เครื่องบันทึกข้อมูลไร้สาย

WIRELESS DATA STORAGE

โดย

นายไชยรัฐ ศรีเมืองบุญ

นายบรรจง สรรพกิจชาญชัย

นายไพฑูรย์ ประภาอภิรัตน์

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตร์บัณฑิต

สาขาวิศวกรรมการวัดคุมทางอุตสาหกรรม

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหาร ลาดกระบัง

ปีการศึกษา 2538

ปริญญานิพนธ์ ปีการศึกษา 2538

ภาควิชา เทคโนโลยีการวัดคุมทางอุตสาหกรรม

คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าคุณทหาร ลาดกระบัง

โครงงาน ระบบบันทึกข้อมูลไร้สาย

โดย

นายไชยรัฐ	ศรีเมืองบุญ	เลขประจำตัว	36012097
นายบรรจง	สรรพกิจชาญชัย	เลขประจำตัว	36012105
นายไพฑูรย์	ประภาอภิรัตน์	เลขประจำตัว	36012110

.....

(อาจารย์ ทรงชัย วีระทวีมาศ)

อาจารย์ที่ปรึกษา

เครื่องบันทึกข้อมูลไร้สาย

WIRELESS DATA STORAGE

โดย

นายไชยรัฐ ศรีเมืองบุญ

นายบรรจง สรรพกิจชาญชัย

นายไพฑูรย์ ประภาอภิรัตน์

อาจารย์ที่ปรึกษา

อาจารย์ ทรงชัย วีระทวีมาศ

บทคัดย่อ

เครื่องบันทึกข้อมูลไร้สาย ประกอบ Microcontroller 8051 เป็นตัวควบคุมการทำงาน และส่วนประกอบอื่น ๆ บนเครื่องได้แก่ แป้นพิมพ์ ขนาด 16 ปุ่ม ชุดแสดงผล LCD ชุดส่ง INFARED เครื่องบันทึกข้อมูลไร้สาย ถูกออกแบบมาเพื่อ นำไปใช้บันทึกข้อมูลนอกสถานที่ เช่น การบันทึกข้อมูล การใช้ไฟฟ้า จาก Meter แล้วนำข้อมูลเข้ามา load ลงบนเครื่อง PC เพื่อให้คำนวณหารราคาค่าไฟฟ้า โดยการส่งข้อมูลระหว่างเครื่องบันทึกข้อมูลไร้สาย กับเครื่อง PC จะใช้การรับส่งข้อมูลโดยใช้ INFARED

Abstract

WIRELESS DATA STORAGE Consist of microcontroller 8051 for controlling operation on board and the other component are key board which 4 x 4 dot , the LCD for display data , the infared transmitter, which designed for record data out door for example, the recorder of electrical consumption or watt hour meter . These data will be loaded in to the personal computure for electrical price calculation . The transfer data process between the wireless data storage and personal computure are used the infared signal to transfer

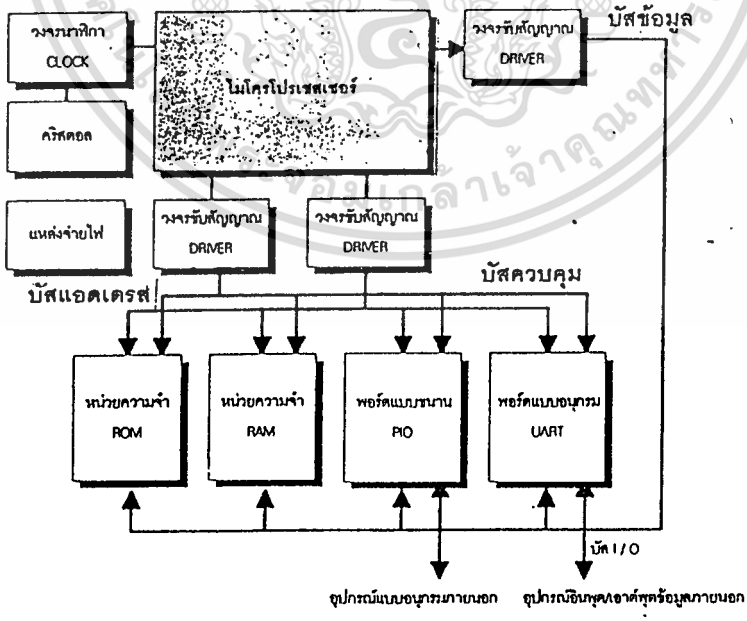
สารบัญ

บทที่ 1
บทนำ
โครงสร้างของ MCS 51

ไมโครคอนโทรลเลอร์ตระกูล 8051

บรรดาไมโครคอนโทรลเลอร์ที่มีการผลิตจากบริษัทต่าง ๆ จำนวนมากนั้น ไมโครคอนโทรลเลอร์จากบริษัท อินเทล (Intel Corporation) ในตระกูล MCS-51 ได้มาการนำไปใช้งานกันแพร่หลายมากนับตั้งแต่ปี ค.ศ. 1980 เป็นต้นมา ได้รับลิขสิทธิ์ในการไปผลิตจำหน่ายและได้มีการเพิ่มประสิทธิภาพและหน่วยการทำงานต่างๆ มากขึ้น ทำให้ในปัจจุบันมีไมโครคอนโทรลเลอร์จากผู้ผลิตต่าง ๆ ที่มีพื้นฐานมาจากไมโครลเลอร์ MCS-51 ของบริษัทอินเทลอยู่เป็นจำนวนมาก

ลักษณะงานที่เหมาะสมกับการนำไมโครคอนโทรลเลอร์ไปใช้งาน มักจะเป็นงานประยุกต์ที่เกี่ยวกับข้องกับการควบคุม หรือจัดการสัญญาณอินพุท / เอาท์พุทของวงจรอิเล็กทรอนิกส์และวงจรดิจิตอลต่าง ๆ เช่น ระบบแสดงผล หรือระบบเตือนภัย ระบบควบคุมภายในเครื่องใช้ไฟฟ้าเป็นต้น ซึ่งงานควบคุมเหล่านี้มักจะไม่มีการคิดคำนวณที่ซับซ้อนมากนัก และต้องการพื้นที่ของแผงวงจรควบคุมที่จำกัด



รูปที่ 1.1 หน่วยการทำงานทั่วไปของระบบไมโครโปรเซสเซอร์

ไมโครคอนโทรลเลอร์ตระกูล MCS-51 ประกอบด้วย ไมโครคอนโทรลเลอร์หลายรุ่น (version) ซึ่งมีสถาปัตยกรรมพื้นฐานที่เหมือนกัน เพียงแต่มีขนาดหรือจำนวนของหน่วยทำงานภายในที่แตกต่างกันออกไป เพื่อความเหมาะสมในงานประยุกต่าง ๆ ตามความต้องการ ดังแสดงให้เห็นในตารางของรูปที่ 1.2 โดยมีทั้งลักษณะที่ใช้เทคโนโลยีการผลิตไอซีวงจรรวมความจุสูงมาก (LSI) แบบ HMOS หรือ CHMOS ซึ่งมีคุณลักษณะที่สูงมากขึ้น และสิ้นเปลืองกำลังไฟฟ้าน้อยกว่ามากอย่างไรก็ตาม การอ้างถึงไมโครคอนโทรลเลอร์ MCS-51 ในหนังสือเล่มนี้ จะเรียกรวมกัน 8051 แทน

EMBEDDED CONTROLLERS

Feature	8051AH	8031AH	8751H	80C51BH	80C31BH	87C51	8052AH	8032AH	8752	8044H
Program Memory (Bytes)	4K	-	4K	4K	-	4K	8K	-	8K	4K
RAM Memory (Bytes)	128	128	128	128	128	128	256	256	256	192
Program Memory Expansion (Off Chip) (Bytes)	64K	64K	64K	64K	64K	64K	64K	64K	64K	64K
Data Memory Expansion (Off Chip) (Bytes)	64K	64K	64K	64K	64K	64K	64K	64K	64K	64K
Max Clock Frequency (MHz)	12	12	12	16	16	16	16	12	12	12
Typical instruction Time (uS)	1	1	1	0.75	0.75	0.75	1	1	1	1
16-Bit Timer / Counters	2	2	2	2	2	2	3	3	3	2
Serial Communications	Synchronous Mode, Asynchronous Mode, 9 or 10 - Bit Programmable									HDLC/SDLC
No. of I/O Lines	32	16	32	32	16	32	32	16	32	32
Interrupt Sources	5	5	5	5	5	5	6	6	6	5
(Two Priority Levels)										
Power Requirements 125	125	250	24	24	29	175	175	175	200	
(ICC Max. mA)										
Programmable Power Modes	-	-	-	4.0 mA	4.0 mA	4.0 mA	-	-	-	-
Idle Powet Down				50 uA	51 uA	52 uA				30 mA

รูปที่ 1.2 ตารางแสดงไมโครคอนโทรลเลอร์ตระกูล MCS-51 ของบริษัทอินเทล

จากแผนภาพในรูปที่ 1.3 แสดงให้เห็นถึงหน่วยการทำงานพื้นฐานของไมโครคอนโทรลเลอร์เบอร์ต่าง ๆ ที่จัดอยู่ภายในตระกูล MCS-51 นี้ ประกอบด้วย

หน่วยประมวลผลกลายขนาด 8 บิต

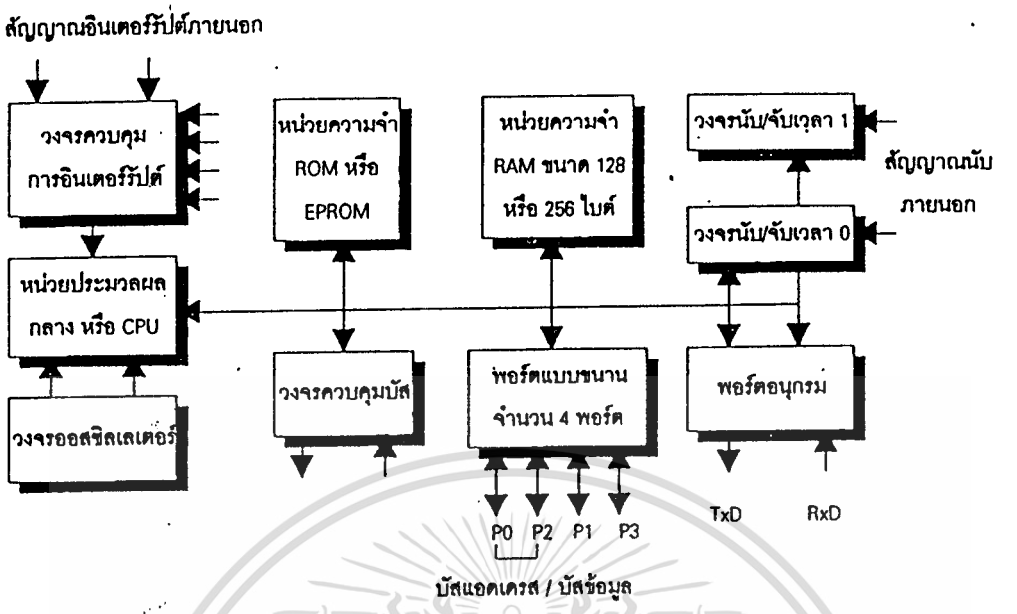
หน่วยประมวลผลสำหรับข้อมูลแบบบิต (Boolean Processor)

ความสามารถในการอ้างตำแหน่งของหน่วยความจำโปรแกรม 64 กิโลไบต์

ความสามารถในการอ้างตำแหน่งของหน่วยความจำข้อมูล 64 กิโลไบต์

หน่วยความจำโปรแกรมภายในขนาด 4 กิโลไบต์ แบบ EPROM (เบอร์ 8751) หรือแบบ ROM (เบอร์ 8051)

หน่วยความจำแบบ RAM ภายในจำนวน 128 ไบต์



รูปที่ 1.3 แผนภาพบล็อกแสดงหน่วยทำงานพื้นฐานของ MCS-51

พอร์ตอินพุท / เอาท์พุทแบบขนานจำนวน 32 เส้น ซึ่งสามารถแยกทำงานได้อย่างอิสระ

วงจรนับ / จับเวลาขนาด 16 บิต จำนวนสองวงจร

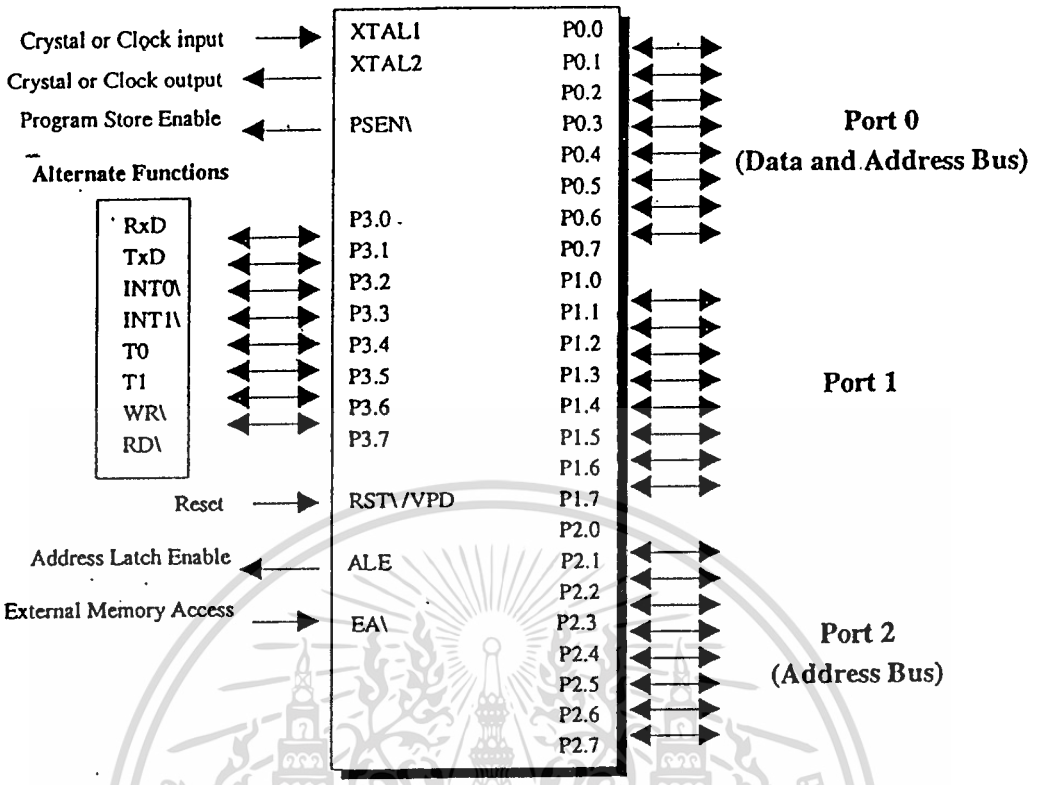
วงจรสื่อสารแบบอนุกรมแบบฟูลดูเพล็กซ์ (Full dulplex)

วงจรควบคุมการอินเตอร์รัปต์จากแหล่งกำเนิดสัญญาณ 6 ประเภท พร้อมการกำหนดลำดับความสำคัญได้สองระดับ

วงจรออสซิลเลเตอร์ภายใน

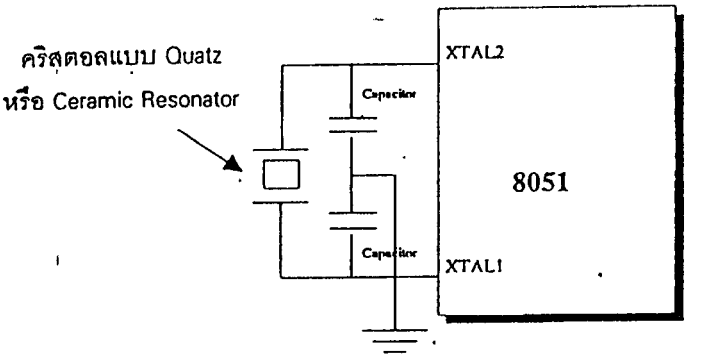
โดยมากแล้วไมโครคอนโทรลเลอร์ตระกูลนี้ มักจะมีรูปร่างของไอซีเป็นแบบ DIP ขนาด 40 ขา ดังแผนภาพในรูปที่ 1.4 ซึ่งแต่ละขาสัญญาณจะมีหน้าที่ที่ระบุชัดเจนตามสัญลักษณ์ชื่อย่อที่กำกับในแต่ละขา อย่างไรก็ตามจะมีบางขาสัญญาณที่อาจจะมีหน้าที่ได้มากกว่าหนึ่งอย่าง (ซึ่งเขียนกำกับไว้ว่า Altemate Funotions ในรูปที่ 1.4) ซึ่งจะไม่สามารถใช้งานในเวลาเดียวกันได้ตัวอย่างเช่น ขาสัญญาณบิต 0 ของพอร์ต 3 (ใช้ตัวย่อเป็น P3.0) อาจจะใช้เป็นขาสัญญาณเอาท์พุท / อินพุทตามปกติ หรืออาจทำหน้าที่เป็นขาสัญญาณอินพุทของข้อมูลสื่อสารแบบอนุกรม (RxD) กับวงจรสื่อสารแบบอนุกรมของ 8051 ได้ซึ่งการกำหนดว่าจะทำงานในลักษณะใดก็ขึ้นอยู่กับการเชื่อมต่อวงจรเข้ากับขาสัญญาณและโปรแกรมควบคุมของระบบนั้น

8051 มีวงจรออสซิลเลเตอร์อยู่ภายใน สำหรับการสร้างพัลส์ของสัญญาณนาฬิกา ซึ่งจะนำไปเป็นฐานเวลา หรือการกำหนดจังหวะการทำงานของหน่วยการทำงานทั้งหมดให้สอดคล้องกัน (Sinchronization) โดยปกติแล้วก็มักจะทำโดยการใช้คริสตอลเชื่อมต่อเข้ากับขาสัญญาณ XTAL1 และ XTAL2 พร้อมกับตัวเก็บประจุดังลักษณะในรูปที่ 1.5 หรืออาจเป็นสัญญาณนาฬิกาจากภายนอกก็ได้



รูปที่ 1.4 การกำหนดหน้าที่ขาสัญญาณของไอซี 8051

พัลส์ความถี่ของสัญญาณนาฬิกาจะเรียกว่า Pulse (ใช้สัญลักษณ์เป็นตัวอักษร P) และคาบของสัญญาณนาฬิกานี้ เรียกว่า **คาบเวลาออสซิลเลเตอร์** (Oscillator period) คาบเวลาออสซิลเลเตอร์จำนวนสองคาบ เรียกว่า State (ใช้สัญลักษณ์เป็นตัวอักษร S) ซึ่งจะนำไปใช้เป็นช่วงเวลาพื้นฐานการทำงานย่อยของไมโครคอนโทรลเลอร์ เช่น การนำคำสั่ง (Fetch) การถอดความหมาย (Decode) การประมวลผล (Execute) และการเขียนข้อมูล (Write) เป็นต้น ดังแสดงเป็นแผนภาพในรูปที่ 1.6 ช่วงเวลาของ State จำนวนหกครั้ง จะเรียกว่า แมชชีนไซเคล (Machine cycle) ดังนั้นค่าหนึ่งแมชชีนไซเคิลจะใช้เวลา 12 คาบเวลาออสซิลเลเตอร์ ค่าของ



รูปที่ 1.5 แสดงการใช้คริสตอลภายนอกต่อเข้ากับวงจรออสซิลเลเตอร์ภายใน 8051

แมชชีนไซเคลซิลนี้จัดว่าเป็นช่วงเวลาที่น้อยที่สุดในการทำคำสั่งใดคำสั่งหนึ่ง ซึ่งหากว่าเป็นคำสั่งที่ซับซ้อนมากก็จะต้องใช้เวลานานสองถึงสามแมชชีนไซเคิล

การคำนวณหาว่าเวลาที่ใช้ในการทำคำสั่งใดจนเสร็จสิ้น จะต้องดูว่าคำสั่งนั้นใช้จำนวนแมชชีนไซเคิลเป็นเท่าไรในการประมวลผล (ขอให้ดูข้อมูลจากภาคผนวก ข) เวลาที่ใช้จะคำนวณตามสูตร

$$T = \frac{C \times 12}{Crystal\ Freduency}$$

โดย C เป็นค่าจำนวนแมชชีนไซเคิลของคำสั่ง

Crystal Frequency เป็นค่าความถี่ของคริสตอลที่ใช้กับ 8051

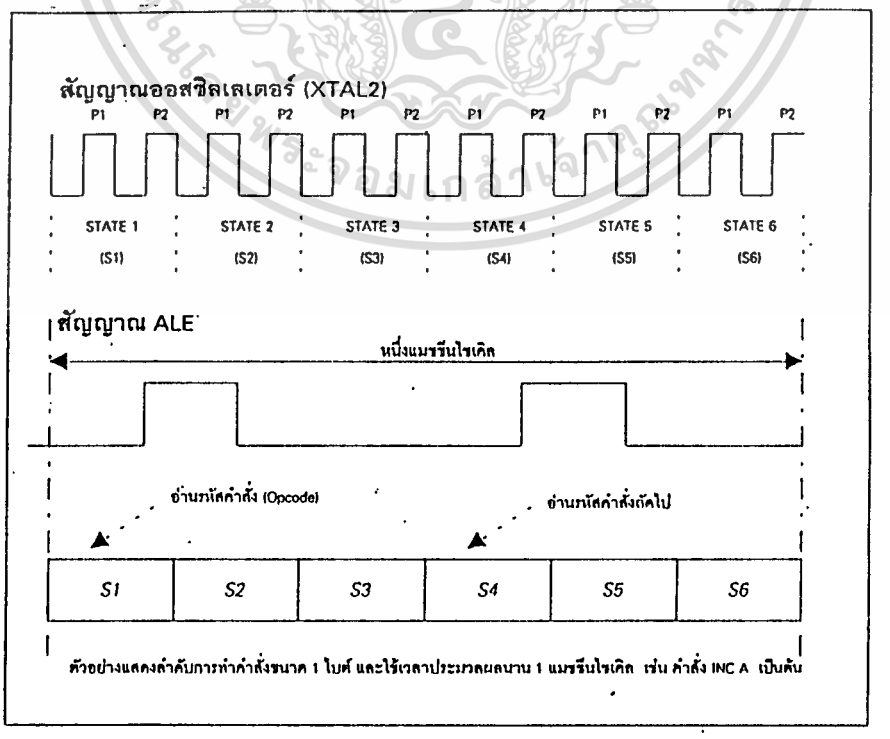
ตัวอย่างเช่น

เวลาในการทำคำสั่ง ADD A,R1 ซึ่งต้องการ 3 แมชชีนไซเคิล

เมื่อใช้คริสตอล 16 เมกะเฮิร์ต จะเป็นเวลานาน 0.75 ไมโครวินาที และ

เมื่อใช้คริสตอล 12 เมกะเฮิร์ต จะเป็นเวลานาน 1 ไมโครวินาที เป็นต้น

อย่างไรก็ตาม ในบางครั้งอาจจะพบเห็นการใช้ค่าของคริสตอลเป็น 11.059 เมกะเฮิร์ต ทั้งนี้โดยมีเหตุผลเนื่องจาก สามารถนำค่าความถี่ที่ได้นี้ ไปใช้ในการเป็นฐานเวลาสำหรับการสร้างความถี่ในการรับ / ส่งข้อมูลอนุกรมซึ่งเป็นหน่วยการทำงานหนึ่งภายใน 8051 เอง โดยจะทำให้ได้ค่าที่ใกล้เคียงกับค่ามาตรฐานคือ 19200, 9600, 4800, 2400, 1200, และ 300 บิต/วินาที ซึ่งจะได้กล่าวถึงอีกครั้งหนึ่งในบทที่ 8 ต่อไป



รูปที่ 1.6 แสดงแผนภาพเวลาพื้นฐานของ 8051 และลำดับของช่วงเวลา state ในการทำคำสั่งหนึ่งไบต์

บทที่ 2

การเชื่อมต่อของอุปกรณ์ 8051

การต่อคีย์บอร์ด

จัดประสงค์

เพื่อรับทราบแนวความคิดในการควบคุมการรับข้อมูลเข้ามาจากคีย์บอร์ด ซึ่งจัดวางไว้ในลักษณะแบบเมตริกซ์ โดยการใช้พอร์ตแบบขนานของ 8051

วงจรประกอบ

โดยปกติแล้วการควบคุมการรับสัญญาณอินพุทจากลุ่มของสวิตช์ (คีย์บอร์ด) ก็มักจะใช้เส้นสัญญาณเดี่ยวในการจัดการเช่นดังในตัวอย่างที่ผ่านมา แต่อย่างไรก็ตามหากว่าจำนวนของสวิตช์มีมากเกินกว่า 8 ตัวแล้ว ก็จะต้องใช้เส้นสัญญาณจากพอร์ตมากกว่าหนึ่งพอร์ตเข้าทำหน้าที่ดูแล ดังนั้น หากว่าได้มีการจัดวางองค์ประกอบของสวิตช์เป็นแบบเมตริกซ์แล้ว ก็จะทำให้สามารถลดจำนวนของเส้นสัญญาณเหล่านี้ลงได้มาก ขอให้ดูการเปรียบเทียบจากตารางข้อมูลในรูปที่ 2.1 ซึ่งหลักในการคำนวณหาจำนวนเส้นสัญญาณที่ต้องใช้งานคือ หากเป็นเมตริกซ์ขนาด M x N ก็จะใช้เส้นสัญญาณจำนวนเพียง M + N เส้นเท่านั้น

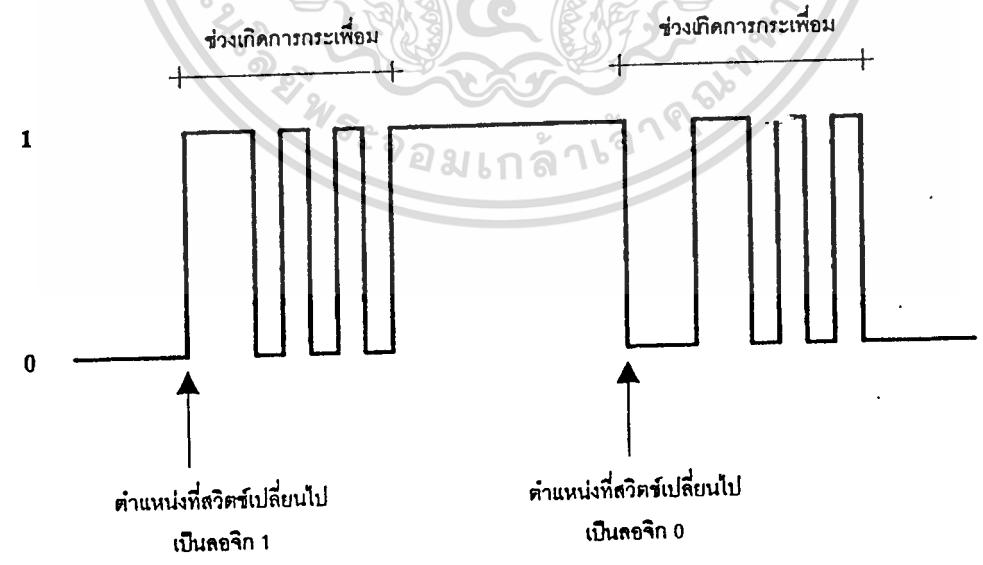
หลักการของโปรแกรมในการตรวจสภาพการกดสวิตช์ จะใช้หลักการ Keyboard Soan โดยเริ่มต้นต้องทำให้สภาวะลอจิกของทุกเส้นสัญญาณในแนวคอลัมน์เป็นค่า 1 เสียก่อน จากนั้นจึงจะเริ่มต้นกำหนดค่าลอจิกของเส้นสัญญาณแถวแรก (Row 0) เป็นลอจิก 0 เมื่อได้อ่านค่าข้อมูลของสัญญาณในแนวคอลัมน์เข้ามาตรวจสอบ หากพบว่ามีเส้นสัญญาณใดมีค่าเป็นลอจิกต่ำแสดงว่ามีการกดสวิทช์ที่ต่ออยู่ระหว่างคอลัมน์นั้นกับเส้นสัญญาณในแนวแถวแรก ทำให้สัญญาณทั้งสองเชื่อมต่อเข้าด้วยกัน หากว่าไม่มีเส้นสัญญาณใดเป็นลอจิกต่ำเลยแสดงว่าไม่มีการกดสวิตช์ใด ๆ ในแถวแรกเลย จากนั้นก็จะดำเนินการซ้ำเช่นนี้กับเส้นสัญญาณในแถวอื่น ๆ อีกในรอบถัดไปจนกระทั่งครบทุกแถว

ขนาดของเมตริกซ์	จำนวนของเส้นสัญญาณ	
	แบบแยกโดยอิสระ	แบบเมตริกซ์
3 x 3	9	6
4 x 4	16	8
4 x 6	24	10
5 x 5	25	10

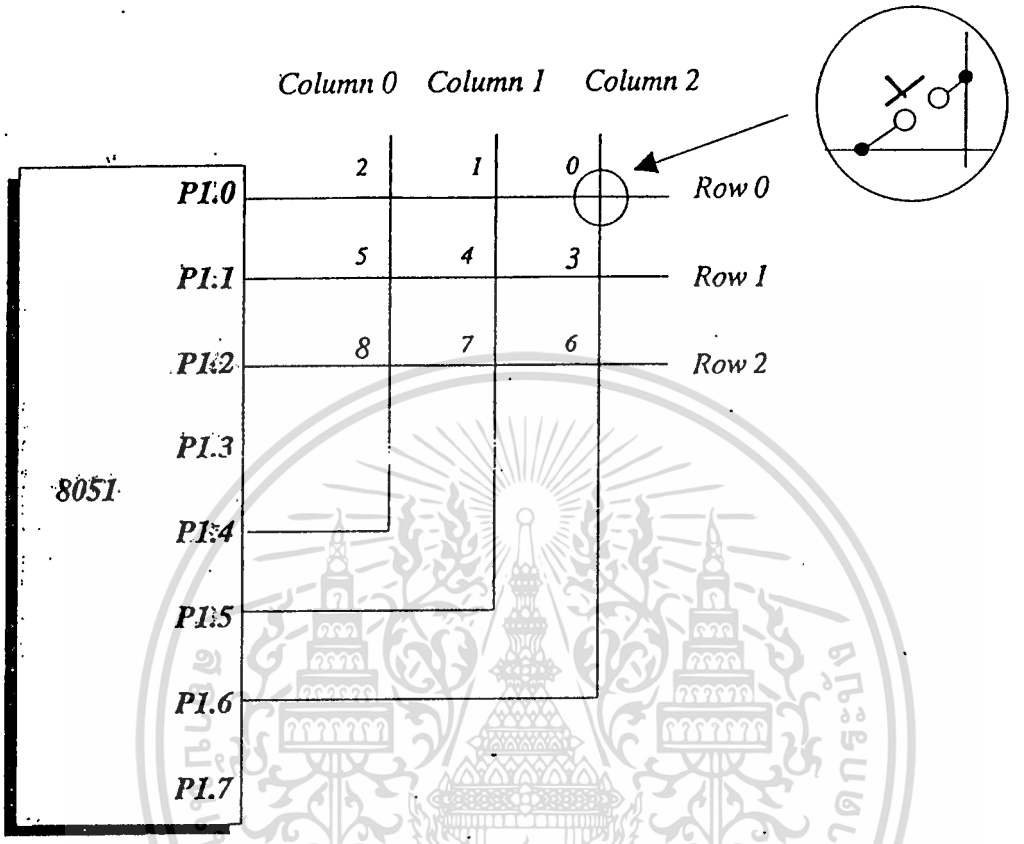
รูปที่ 2.1 ตารางเปรียบเทียบจำนวนสัญญาณจัดการอินพุทข้อมูลจากคีย์บอร์ด ที่จัดวางโดยอิสระและแบบเมตริกซ์

ในส่วนเริ่มต้นของการทำงาน โปรแกรมจะทำการกำหนดค่าเริ่มต้นต่าง ๆ ที่จะใช้งานภายในโปรแกรมให้เสร็จสิ้น ได้แก่ ค่าของคีย์เริ่มต้น เป็น - 1 ค่าจำนวนแถวและคอลัมน์ ค่าของรูปแบบสำหรับการตรวจสอบ การกดคีย์ในแถวแรก (row 0) โดยทำให้บิตอื่นๆ เป็นหนึ่งยกเว้นบิต 0 เป็นค่าศูนย์ จากนั้นจึงนำค่ารูปแบบบิตนี้ไปส่งออกยังพอร์ต P1 และเตรียมการสำหรับค่ารูปแบบที่จะใช้ในแถวถัดไปด้วยการเลื่อนบิตไปทางซ้ายหนึ่งบิต เมื่อทำการอ่านข้อมูลจากพอร์ตเข้ามาอีกครั้งหนึ่งแล้วจะใช้หลักการตรวจสอบว่าบิตใดมีค่าเป็นศูนย์เกิดขึ้นบ้าง ซึ่งในขณะที่ตรวจสอบแต่ละบิตนั้นก็จะทำการเพิ่มค่าของคีย์ในลำดับที่สัมพันธ์กันด้วย หากพบว่าบิตใดเป็นศูนย์ก็จะถือว่าเป็นการสิ้นสุดโปรแกรมทันที แต่หากว่าไม่มีการกดสวิทช์เกิดขึ้นเลยโปรแกรมก็จะทำงานวนรอบการตรวจสอบ เหมือนกับขั้นตอนที่ได้กล่าวมาข้างต้นเรื่อยไป

อย่างไรก็ตาม หากว่าต้องการจะนำไปประยุกต์ใช้งานก็จะต้องเพิ่มโปรแกรมย่อยหน่วงเวลา (Time Delay) สักระยะหนึ่ง ก่อนที่จะได้กลับมาวนรอบการอ่านสภาวะสวิตช์ในรอบใหม่เนื่องจากว่าในสภาพความเป็นจริงแล้ว ลักษณะสมบัติเชิงกลของอุปกรณ์สวิตช์ภายหลังการกดสวิตช์หนึ่งครั้ง ก็จะมีสภาพการกระเพื่อมของหน้าสัมผัส (Contact bounce) ทำให้มีลักษณะเหมือนการเปิดและปิดหน้าสัมผัสหลาย ๆ ครั้งก่อนที่กลับสู่สภาวะปกติ โดยทั่วไปจะใช้เวลาประมาณ 5 - 20 มิลลิวินาที ซึ่งหากว่าเราไม่ให้ความสนใจในเรื่องนี้แล้วก็จะมีผลทำให้โปรแกรมดำเนินการผิดพลาดได้ เนื่องจากมองเห็นไปว่ามีการกดสวิตช์หลายครั้ง (ขอให้ดูรูปของสัญญาณจากรูปที่ 2.2) ดังนั้นจึงต้องทำการหน่วงเวลาโปรแกรมสักระยะหนึ่งเพื่อเป็นการลดปัญหาของการกระเพื่อมดังกล่าวข้างต้น (Debounce) โดยใช้กระบวนวิธีทางด้านซอฟต์แวร์



รูปที่ 2.2 แผนภาพสัญญาณการเปลี่ยนแปลงจากเกิดการกระเพื่อมของหน้าสัมผัสสวิตช์



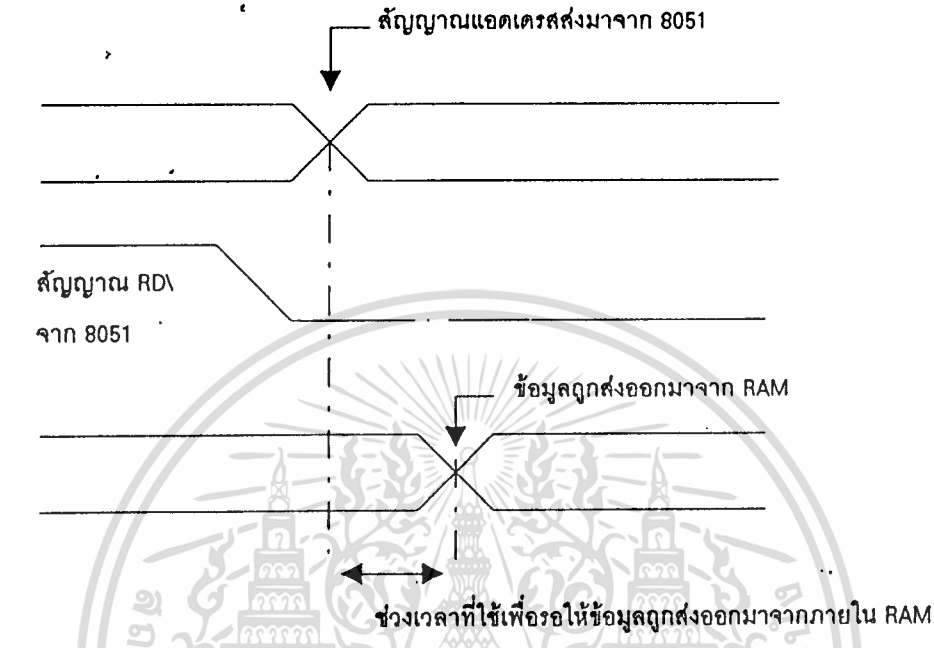
รูปที่ 2.3 แผนภาพวงจรการต่อ 8051 กับสวิตช์แบบกดติดปล่อยดับแบบเมตริกซ์สำหรับโปรแกรมตัวอย่าง

การเชื่อมต่อหน่วยความจำข้อมูลภายนอก (RAM)

การใช้หน่วยความจำข้อมูลภายนอกเป็นวิธีการแก้ปัญหาอย่างหนึ่ง ในกรณีที่มีความต้องการหน่วยความจำสำหรับการเก็บข้อมูลภายใน ซึ่งมีขนาดเพียง 128 หรือ 256 ไบต์เท่านั้น บางครั้งการใช้หน่วยความจำข้อมูลภายนอกยังเหมาะกับงานประยุกต์บางอย่างที่จำเป็น ต้องมีการเก็บสำรองข้อมูลบางอย่างไว้ไม่ให้สูญหายแม้ว่าจะไม่มีการจ่ายไฟฟ้ากับระบบ สามารถทำได้โดยการใช้ไอซีหน่วยความจำ RAM พร้อมแบตเตอร์รี่สำรองประเภทลิเทียมหรือนิเกิล - แคดเมียมเป็นตัวเก็บข้อมูลเหล่านี้ไว้แทน อย่างไรก็ตามไม่ว่าสาเหตุการนำไอซีหน่วยความจำภายนอกมาใช้งานจะเป็นอะไรจะมีผลทำให้พอร์ตอินพุท / เอาท์พุตข้อมูลของ 8051 ถูกนำไปใช้เพื่อติดต่อกับหน่วยความจำเหล่านี้ แทน ดังนั้นจึงอาจจำเป็นต้องมีการใช้วงจรประกอบอื่น ๆ เพ่อชดเชยความสามารถเหล่านั้นของ 8051 แทน

กระบวนการนำข้อมูลเข้าไปจัดเก็บไว้ภายในหน่วยความจำ RAM แบบสแตติก จะเรียกว่า

การเขียน (Write) ไปยังหน่วยความจำ และเมื่อมีการนำข้อมูลที่จัดเก็บไว้นั้นออกมา ก็จะเรียกว่า **การอ่าน (Read)** จากหน่วยความจำ ซึ่งกระบวนการทั้งสองนี้จะต้องกระทำภายในช่วงรอบเวลาที่



รูปที่ 2.4 แผนภาพเวลาแสดงลำดับเหตุการณ์การอ่านข้อมูลจากหน่วยความจำ RAM

กำหนดไว้เช่นนั้น สำหรับลักษณะสมบัติของหน่วยความจำ RAM มีความคล้ายคลึงกับหน่วยความจำ EPROM มาก รวมทั้งประเภทของสัญญาณติดต่อ คือ บัสแอดเดรส บัสข้อมูล และกลุ่มสัญญาณควบคุมทั้งหมดเพียงแต่มีการเพิ่มสัญญาณ WR เพื่อระบุถึงการนำข้อมูลจากบัสข้อมูลลงเก็บไปยังหน่วยความจำตำแหน่งที่ได้ระบุแอดเดรสมาเท่านั้น

จากแผนภาพเวลาของสัญญาณในรูปที่ 2.4 แสดงให้เห็นถึงลำดับเหตุการณ์เมื่อมีการอ่านข้อมูลจากหน่วยความจำ RAM ดังนี้

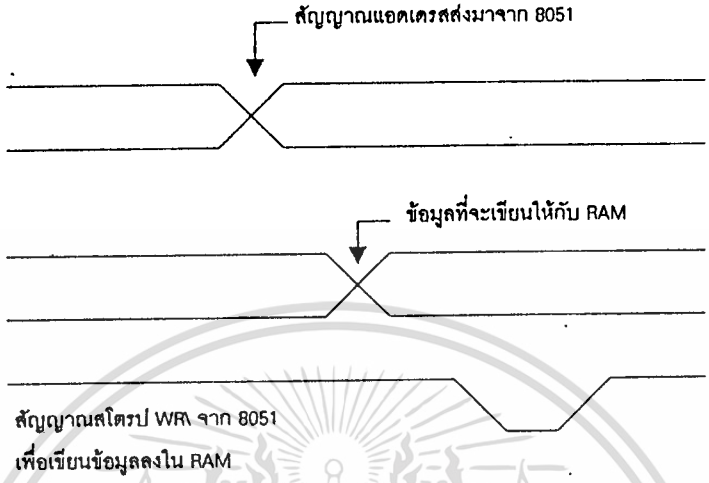
1. ไมโครคอนโทรลเลอร์จะต้องเริ่มต้นด้วยการส่งค่าแอดเดรสของหน่วยความจำที่ ต้องการออกมาทางบัสแอดเดรส

2. สัญญาณ OE จะต้องถูกเปลี่ยนให้เป็นระดับลอจิกต่ำเพื่อระบุว่าต้องการอ่านข้อมูลจากหน่วยความจำ

3. หลังจากนั้นไมโครคอนโทรลเลอร์จะต้องหยุดรอในราวช่วงระยะเวลาหนึ่ง เรียกว่า Read Access time เพื่อให้วงจรภายใน RAM ทำการถอดรหัสแอดเดรสและอ่านข้อมูล

4. หลังจากสิ้นสุดเวลาในข้อ 3 แล้ว ข้อมูลจะถูกส่งออกมาทางขาสัญญาณของบัสข้อมูลและไมโครคอนโทรลเลอร์สามารถอ่านข้อมูลนี้ ไปประมวลผลต่อไป

สำหรับแผนภาพเวลาของสัญญาณในรูปที่ 2.5 แสดงให้เห็นถึงลำดับเหตุการณ์ เมื่อมีการเขียนข้อมูลลงในหน่วยความจำ RAM ดังนี้



รูปที่ 2.5 แผนภาพเวลาแสดงลำดับเหตุการณ์การเขียนข้อมูลลงในหน่วยความจำ RAM

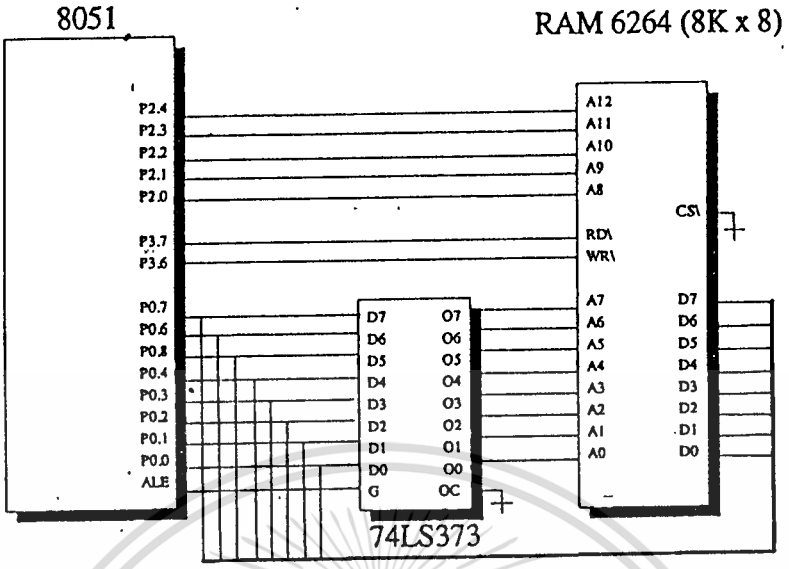
1. ช่วงเริ่มต้นไมโครคอนโทรลเลอร์จะต้องส่งค่าแอดเดรสของหน่วยความจำที่ต้องการออกมาทางบัสแอดเดรส

2. เวลาถัดไปในบัสข้อมูลจะเป็นข้อมูลที่ต้องการเขียนลงในหน่วยความจำ

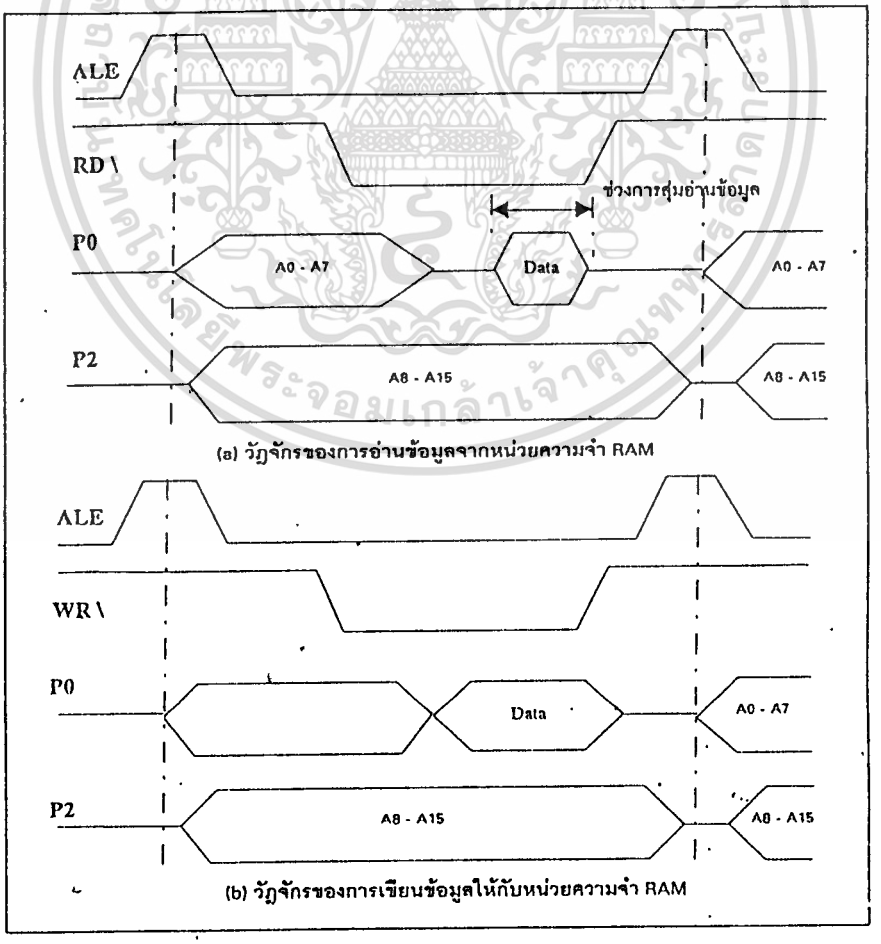
3. ไมโครคอนโทรลเลอร์จะต้องทำการหยุดรอในช่วงระยะเวลาหนึ่งเช่นกัน เรียกว่า Write Access time เพื่อให้วงจรภายใน RAM ทำการถอดรหัสแอดเดรส และเตรียมการเขียนข้อมูล

4. เมื่อสิ้นสุดเวลาที่หยุดรอ ไมโครคอนโทรลเลอร์จะต้องขับสัญญาณ WR ให้เป็นระดับลอจิกต่ำหรือพัลส์ เพื่อให้ข้อมูลภายในบัสข้อมูลถูกนำไปยังตำแหน่งที่ต้องการภายในหน่วยความจำ RAM ต่อไป

การเชื่อมต่อหน่วยความจำ RAM เข้ากับระบบของไมโครคอนโทรลเลอร์ 8051 จะใช้วิธีการเหมือนกับการเชื่อมต่อกับหน่วยความจำ EPROM โดยในรูปที่ 3.10 แสดงให้เห็นถึงวงจรของการเชื่อมต่อเข้ากับหน่วยความจำ RAM แบบสแตติกเบอร์ 6264 ขนาด 8Kx8 ไบต์ มีหลักการทำงานตามที่ได้กล่าวในหัวข้อที่ผ่านมา ไอซี 74LS373 ทำหน้าที่ในการค้างหรือแลตช์ (Latch) ค่าแอดเดรสให้กับอินพุทของหน่วยความจำ RAM ซึ่งมีอยู่เพียงตัวเดียว สำหรับขาสัญญาณ OE (หรือ RD) จะต่อเข้ากับขาสัญญาณ RD(P3.7) และขาสัญญาณ WR(P3.6) ของ 8051 ในที่นี้เนื่องจากว่าภายในวงจรนี้ไอซีหน่วยความจำเพียงตัวเดียว จึงได้ทำการต่อสัญญาณ CS กับสัญญาณกราวด์โดยตรงเพื่อทำการเลือกให้ไอซีทำงานตลอดเวลา



รูปที่ 2.6 วงจรตัวอย่างแสดงการเชื่อมต่อหน่วยความจำข้อมูลภายนอกกับ 8051



รูปที่ 27 แผนภาพแสดงการติดต่อกับหน่วยความจำ RAM ของ 8051

ในรูปที่ 2.7 แสดงให้เห็นถึงแผนภาพเวลาของสัญญาณที่เกี่ยวข้องดังนี้ เมื่อเริ่มการติดต่อกับหน่วยความจำข้อมูลภายนอก จะมีการส่งค่าของแอดเดรสไบต์ต่ำ (A0-A7) ของตำแหน่งหน่วยความจำที่ต้องการออกมายังพอร์ต 0 ในราวก่อนช่วงเลาขอบขาลงของสัญญาณ ALE ซึ่งจะนำไปใช้ในการควบคุมไอซีแลตช์ เพื่อทำการค้างค่าแอดเดรสไบต์ต่ำนี้ไว้ ดังนั้นในเวลาต่อมาพอร์ต 0 ก็จะว่างและพร้อมที่จะนำไปใช้ในฐานะของบัสข้อมูล ส่วนค่าของแอนเดรสไบต์สูง (A8-A150) จะถูกส่งออกมาทางพอร์ต 2 ราวช่วงเวลาประมาณกึ่งกลางระหว่างที่สัญญาณ ALE เป็นระดับลอจิกสูง เมื่อ 8051 ทำการขับสัญญาณ RD หรือ WR ให้เป็นระดับลอจิกต่ำก็จะมีผลทำให้เกิดการส่งและรับ ข้อมูลระหว่างหน่วยความจำ RAM กับบัสข้อมูลขึ้นได้

การแสดงผลด้วยจอ LCD

อุปกรณ์ในปัจจุบันนี้ ใสส่วนแสดงผลนั้นจะใช้ LCD เสียเป็นส่วนใหญ่ไม่ว่าจะเป็นเครื่องเล่น Vedio , เครื่องถ่ายเอกสาร , เครื่องมือวัดคุมต่าง ๆ , เครื่องคอมพิวเตอร์ เราพอจะแบ่ง DOT MATRIX LCD MODULE นี้ออกได้เป็นพวก ๆ ดังนี้ :-

1. CHARACTER LCD MODULE
2. GRAPHIC MODULE
3. SEGMENT DISPLAY TYPE LCD MODULE

โดยในแต่ละแบบนี้ก็จะมีส่วนประกอบใหญ่ ๆ แบ่งได้เป็น

1. DOT MATRIX LCD เป็นตัวแสดงให้เรามองเห็นในลักษณะ การหิดปละเปิดตัวเองกับแสง ก็คือ ส่วนของที่เป็นตัวกระจกบรรจุพลัก

2. DRIVER เป็นตัวรับสัญญาณจากตัวควบคุมมาขับผลัก LCD อีกทีหนึ่งโดยมีเบอร์ที่นิยมใช้ใน LCD MODULE เช่น HD44100H , MSM5259

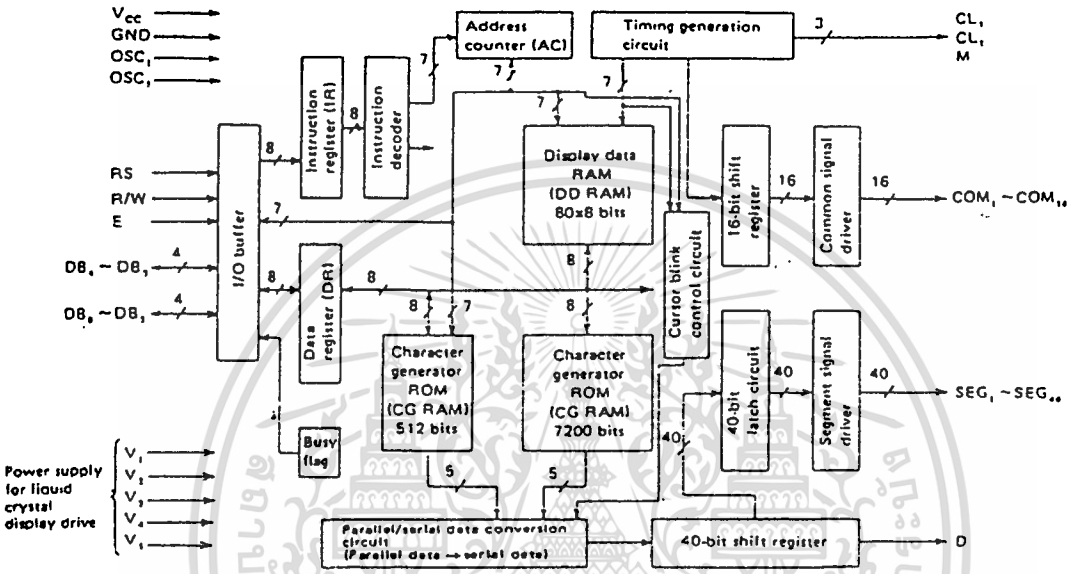
3. CONTROLLER เป็นตัวรับข้อมูลจากอุปกรณ์ภายนอกมาและจัดการควบคุม LCD MODULE ให้ทำงานแสดงผลต่าง ๆ เช่น การลบจอภาพ , การเกิดตัวอักษร เป็นต้น โดยมีเบอร์ IC ที่นิยมใช้กันคือ HD44780 ซึ่งจะใช้แบบ CHARACTER LCD MODULE เป็นส่วนใหญ่ เบอร์ IC HD61830 จะใช้ใน แบบ GRAPHIC LCD MODULE

ในการศึกษาการทำงาน และใช้งาน LCD MODULE นั้นไม่ใช่เรื่องยากเลยถ้าเราสามารถทำความเข้าใจในส่วนของ CONTROLLER ได้ก็เพียงพอแล้วและโดยมาก LCD MODULE ในแต่ละ บริษัทแล้วจะใช้ตัว CONTROLLER ที่มีหลักการทำงานเหมือน ๆ กันเป็นส่วนใหญ่และใน LCD MODULE แต่ละขนาดจำนวนตัวอักษรหรือ จำนวนบรรทัดที่มีหลักการทำงานแบบเดียวกันทั้งหมด IC ที่นิยมมากที่สุด ตัวหนึ่งที่เป็น CONTROLLER LCD ก็คือ HD44780 โดยรูปแบบการทำงานของมันได้เป็นมาตรฐานให้กับ CONTROLLER LCD ตัวอื่น ๆ ด้วย

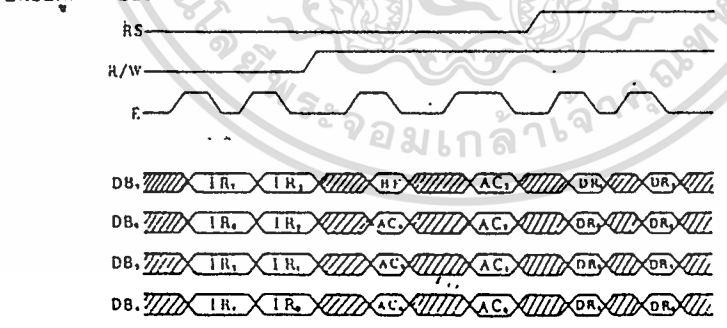
HD44780 เป็นไอซี LSI ตัวหนึ่งใช้ควบคุม LCD โดยแสดงผลในรูปตัวอักษรหรือสัญลักษณ์ต่าง ๆ ตัวมันเองสามารถต่อใช้งานแบบ 4 BIT หรือ 6 BIT ก็ได้ โดยถ้าเราต่อแบบ 4 BIT จะต่อใช้งานที่ DB7 -

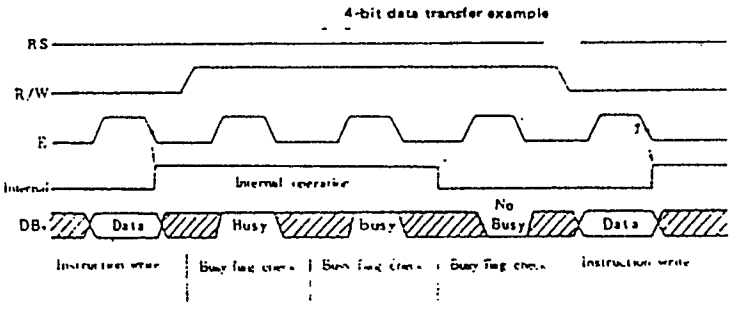
DB4 เท่านั้นโดยข้อมูลครั้งแรกที่ส่งนั้น HD44780 จะถือเป็นข้อมูล 4 BIT บน และข้อมูลที่ส่งต่อ มานั้นเป็นข้อมูล 4 BIT ล่าง

Block diagram of HD44780 interior



เท่านั้นโดยข้อมูลครั้งแรกที่ส่งนั้น HD44780 จะถือเป็นข้อมูล 4 BIT บน และข้อมูลที่ส่งต่อมานั้น
เป็นข้อมูล 4 BIT ล่าง





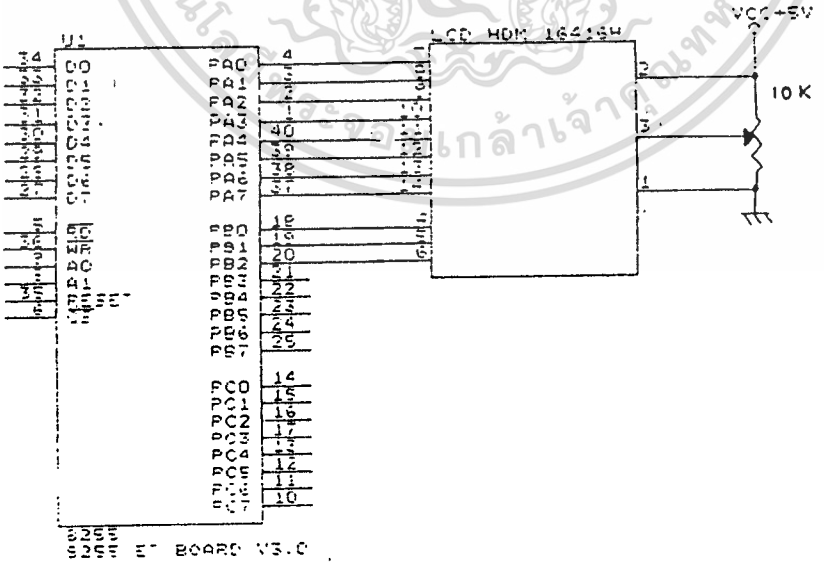
การต่อใช้งานจริงของ 8255 กับ LCD MODULE

จากวงจรเป็นการต่อ 8255 ให้เข้าใช้กับ LCD โดยเราจะจำลองสัญญาณต่าง ๆ ขึ้นมา โดยการใช้ PORT A และ PORT A นั้นเราให้เป็น DATA PORT และ PORT B นั้นเราให้เป็นสัญญาณควบคุมไปใช้

เมื่อเราเริ่มเปิดไฟป้อนให้ HD44780 นั้นก็จะทำการ RESET ตัวมันเองโดยจะใช้เวลาประมาณ 10 ms หลังจากไฟ VDO ถึง 4.5 VOLT แล้ว โดยจะ SET ตัวมันเองดังนี้

1. DISPLAY CLEAR จะทำการลบข้อมูลจอภาพ LCD
2. FUNCTION SET โดยจะ SET ค่าภายใน
   DL = 1 : เป็นการ SET ให้การติดต่อแบบ 8 บิต
   N = 0 : SET เป็น 1 บรรทัดการแสดงผล
   F = 0 : 5X7 DOT ต่อหนึ่งต่ออักษร
3. DISPLAY ON/OFF D = 0 : DISPLAY OFF
   C = 0 : CURSOR OFF
   B = 0 : BLINK OFF
4. ENTRY MODE SET I/D = 1 : +1 (เพิ่มค่า COUNTER ขึ้น 1)
   S = 0 : NO SHIFT

เมื่อเราเปิดเครื่องทำงานแล้วก็จะต้องส่งคำสั่งควบคุมให้มันเริ่มทำงานดังตาราง



ขาต่าง ๆ ในการต่อใช้งาน HD44780

1. RS (REGISTER SELECTION) จะเป็นขาเลือก REGISTER ภายในซึ่งมีอยู่ 2 ตัว คือ INSTRUCTION REGISTER (IR) และ DATA REGISTOR (DR) โดยถ้าเป็น 1 จะเป็นการเลือก DATA และ ถ้าเป็น 0 จะเป็นการเลือก INSTRUCTION

2. R/W (READ/WRITE) เป็นตัวเลือกว่าจะเขียนหรือจะอ่านข้อมูลจากตัว IC โดย อ่านข้อมูล = 1 , เขียนข้อมูล = 0

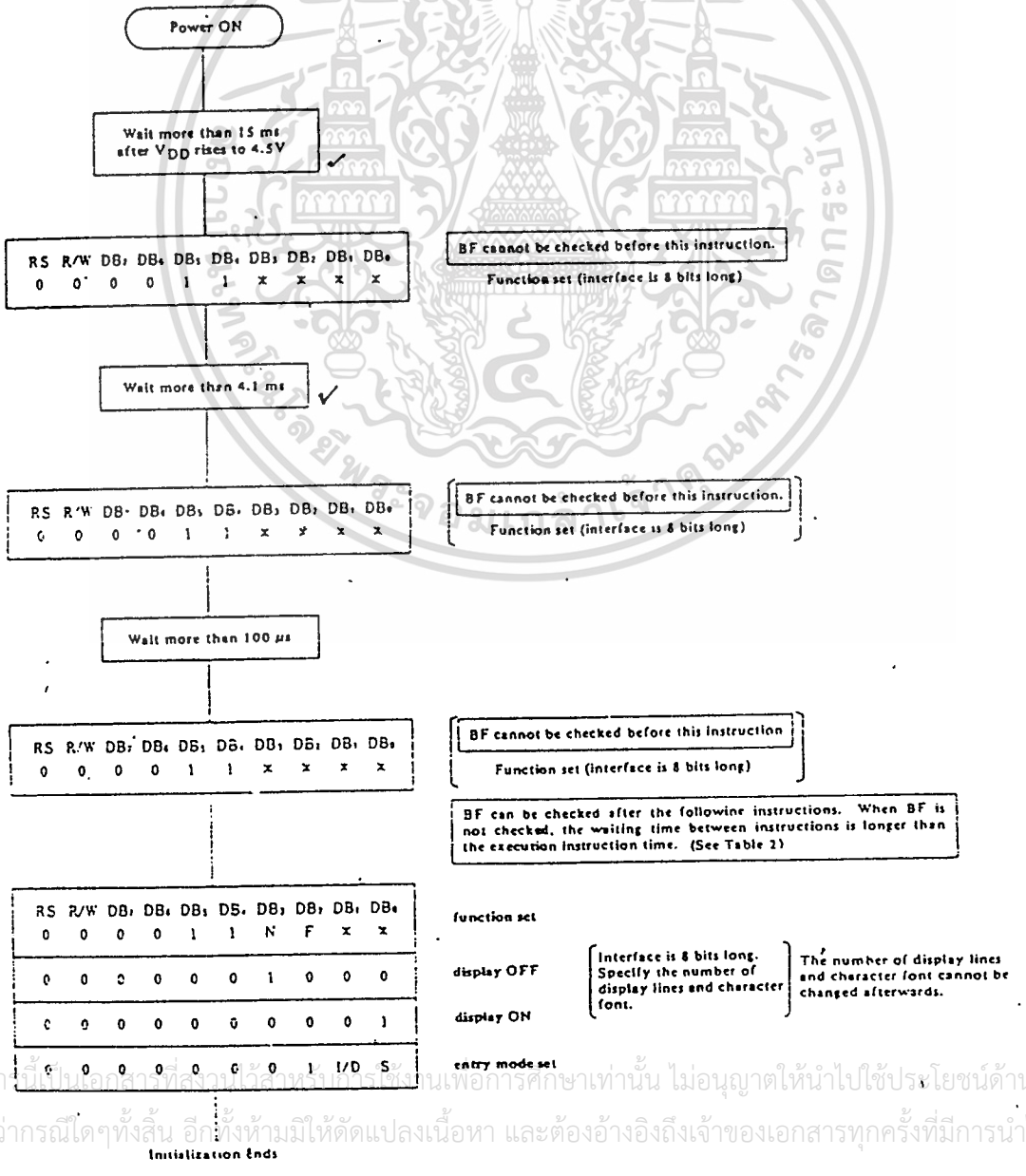
3. E (ENABLE SIGNAL) เป็นขากำหนดสภาพการรับเขียนอ่านข้อมูล

4. DBO - DB7 เป็นขารับส่งข้อมูลจากตัว C

5. VDD ไฟเลี้ยวตัววงจร

6. VSS เป็นขา GND

7. VO เป็นขารับ VOLOTAGE ในการรับ LCD ให้สว่างหรือมืด

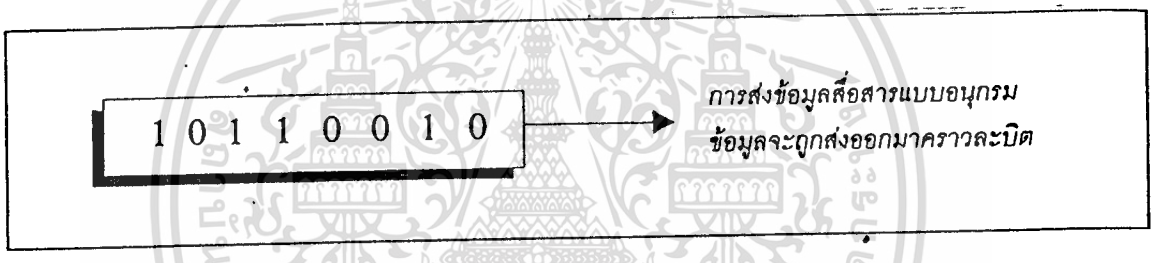


บทที่ 3

การสื่อสารข้อมูลอนุกรมผ่านทาง MCS 51

การสื่อสารข้อมูลอนุกรม

เนื่องจากการสื่อสารแบบอนุกรมเป็นการรับ/ส่งข้อมูล ในลักษณะกลุ่มของบิตข้อมูล (Bit Stream) ดังนั้นจึงต้องให้ความสนใจในการพิจารณาถึงเรื่องของอัตราความเร็วในการรับ/ส่งบิตเหล่านี้เป็นลำดับแรก โดยทั่วไปมักจะระบุกันในหน่วยของจำนวนบิตข้อมูลภายในเวลาหนึ่งวินาที เรียกว่า **อัตราบอด (Baud Rate)** ตามค่ามาตรฐานเหล่านี้ ได้แก่ 110,150,300,1200,2400,4800 และ 9600 จากรูปที่ 3.3 แสดงให้เห็นลักษณะของรูปแบบสัญญาณข้อมูลอนุกรมที่ปรากฏในสายส่งสัญญาณ ข้อมูลที่ 8 บิตนี้หากว่าถูกส่งออกมาด้วยอัตรา 2400 บอดจะใช้เวลาในการส่งข้อมูลหนึ่งบิตมีค่าเท่ากับ 1/2400 หรือ 416 ไมโครวินาที และเวลาในการส่งข้อมูลทั้ง 8 บิตมีค่าเท่ากับ (8 x 416) หรือ 3,328 ไมโครวินาที



รูปที่ 3.1 ข้อมูลสื่อสารแบบอนุกรม ข้อมูลหนึ่งไบต์จะถูกส่งออกคราวละบิตเป็นลำดับไป จนครบจำนวนทั้ง 8

ในบทนี้จะได้กล่าวเกี่ยวกับพอร์ตอินพุทแบบอนุกรมของ 8051 โดยเริ่มกล่าวถึงพื้นฐานของการสื่อสารอนุกรมเป็นลำดับแรก จากนั้นจึงเป็นการใช้งานเพื่อจัดการสื่อสารข้อมูลอนุกรมแบบต่าง ๆ ของ 8051 ทั้งหมด ซึ่งกระทำได้อย่างสะดวกมากด้วยการใช้ชุดคำสั่งเพียงหนึ่งหรือสองคำสั่งเท่านั้น ทั้งนี้เนื่องจาก 8051 มีโครงสร้างด้านฮาร์ดแวร์ที่สนับสนุนโดยตรง ความมีประสิทธิภาพด้านการจัดการสื่อสารข้อมูลอนุกรมนี้ จัดได้ว่าเป็นลักษณะเด่นที่สำคัญประการหนึ่งของไมโครคอนโทรลเลอร์ในตระกูลนี้ทีเดียว จึงควรที่จะได้มีการศึกษาเพื่อให้เกิดความเข้าใจในการนำไปประยุกต์ใช้งานได้อย่างมีประสิทธิภาพต่อไป

วิธีการที่จะทำให้ข้อมูลสื่อสารอนุกรมมีความถูกต้องมากยิ่งขึ้น จะใช้การเพิ่มเติมบิตข้อมูลบางอย่างร่วมไปกับการส่งข้อมูลจริง ได้แก่

1. บิตเริ่มต้น (Start Bit)

บิตเริ่มต้นมีหน้าที่สำหรับการบ่งออกให้วงจรฮาร์ดแวร์ทางด้านรับทราบถึง ตำแหน่งจุดเริ่มต้นของบิตข้อมูลกลุ่มใหม่ เพื่อที่จะทำการปรับจังหวะของสัญญาณการรับข้อมูลให้ตรงกันดังนั้นบิตเริ่มต้นนี้จึงจะถูกเพิ่มเข้าไปก่อนมีการส่งข้อมูลจริง ตามปกติแล้วค่าของบิตเริ่มต้นมักจะเป็นระดับลอจิกที่ตรงข้ามกับระดับ

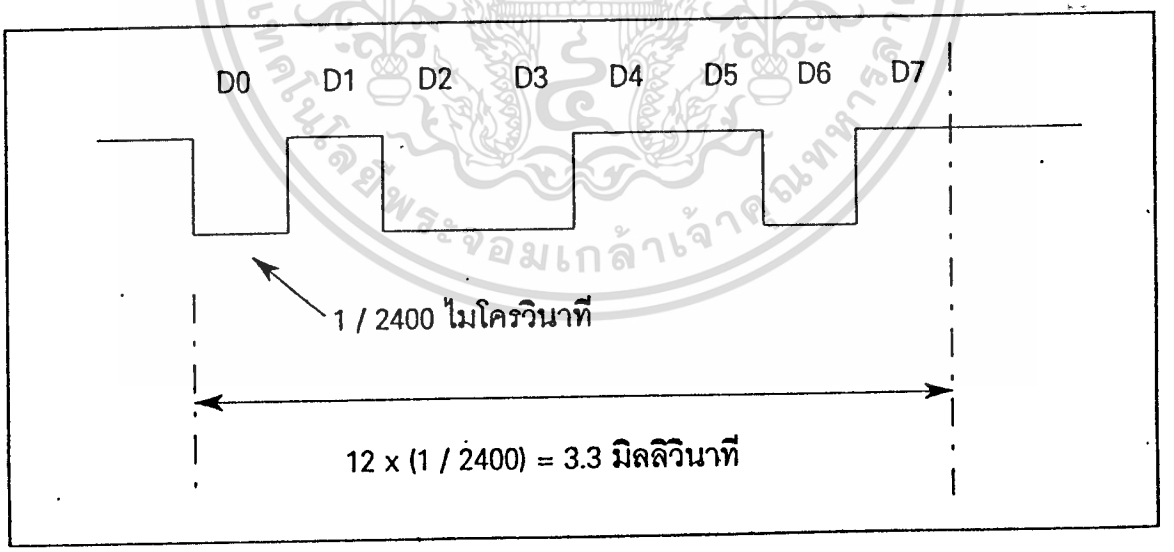
ลอจิกของสภาวะของสายสื่อสาร ขณะเมื่อไม่มีการส่งข้อมูล (Idle State) ตัวอย่างเช่น หากสภาวะของสายเมื่อไม่มีข้อมูลเป็นลอจิกสูง บิตเริ่มต้นก็จะเป็นระดับลอจิกต่ำ เป็นต้น

2. บิตแสดงสภาวะความเป็นเลขคี่ (Parity Bit)

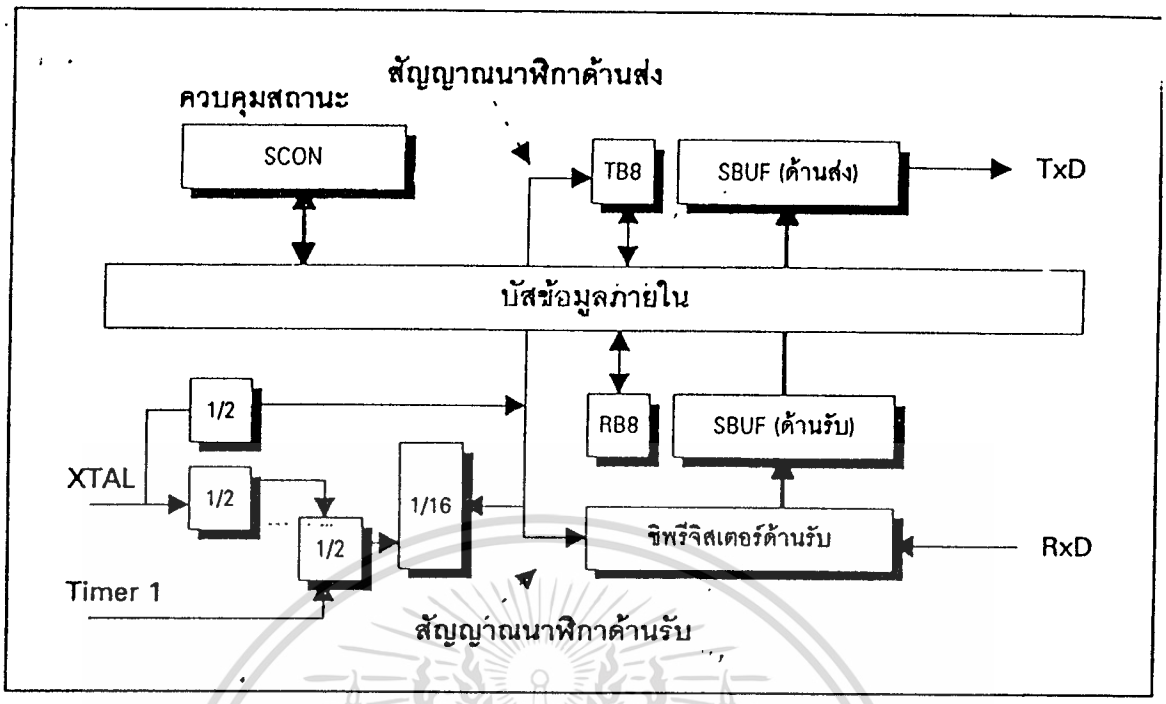
บิตนี้มีหน้าที่เพื่อการตรวจสอบความถูกต้องของข้อมูล โดยทั่วไปมักเรียกว่า **บิตพาริตี้** และจำนำไปแทรกต่อท้ายบิตข้อมูล ค่าของบิตนี้ขึ้นอยู่กับจำนวนค่าของบิตข้อมูลที่เป็น 1 ซึ่งจะเป็นได้สองลักษณะ คือ **พาริตี้คู่ (Even Parity)** หรือ **พาริตี้คี่ (Odd Parity)** ตัวอย่างเช่น ระบบที่ติดต่อกัน โดยระบุว่าจะใช้พาริตี้คู่ (Even Parity) ทางด้านส่งจะนำค่าข้อมูลที่จะส่งมาพิจารณาหากจำนวนของบิตที่มีค่า 1 เป็นเลขจำนวนคู่แล้ว ค่าของบิตพาริตี้จะมีค่าเป็น 0 แต่หากว่าจำนวนของบิตที่มีค่า 1 เป็นเลขจำนวนคี่ ค่าของบิตพาริตี้ ก็จะมีค่าเป็น 1 การพิจารณาทาง

D7	D6	D5	D4	D3	D2	D1	D0
1	**0**	**1**	**1**	**0**	**0**	**1**	**0**

รูปที่ 3.2 บิตข้อมูลสื่อสารแบบขนานภายในไบต์ข้อมูลที่จะถูกส่งออกมาในลักษณะแบบอนุกรม



รูปที่ 3.3 รูปแบบสัญญาณไฟฟ้าของข้อมูลในรูป 8.2 (a) ซึ่งเป็นการส่งข้อมูลแบบอนุกรมด้วยอัตราเร็ว 2400 บิต/วินาที
(สังเกตว่าจะเริ่มส่งจากบิต D0 ซึ่งเป็นบิตนัยสำคัญต่ำออกมาก่อนเป็นลำดับแรก)



รูปที่ 3.5 แผนภาพแสดงการทำงานของวงจรส่วนการรับและส่งข้อมูลอนุกรมของ 8051

SBUF เช่นเดียวกัน แต่ทำหน้าที่เก็บข้อมูลที่นำมาจากส่วนของวงจรเลื่อนบิตหรือชิฟรีจิสเตอร์ (Shift register) ของวงจรจัดการข้อมูลอนุกรมภายใน สัญญาณข้อมูลอนุกรมที่รับเข้าจะผ่านทางขาสัญญาณ RxD (พอร์ต 3.0)

พอร์ตอนุกรมของ 8051 สามารถโปรแกรมให้ทำหน้าในรูปแบบต่าง ๆ กันสี่แบบ (หรือเรียกว่า โหมดการทำงาน) โดยการกำหนดค่าบิต SMO และ SM1 ซึ่งอยู่ภายในรีจิสเตอร์ควบคุมและบอกสถานะ SCON ดังแสดงในรูปที่ 3.6 โหมดการทำงานทั้ง 4 แบบของพอร์ตอนุกรม มีดังนี้

โหมดทำงาน	คำอธิบาย
โหมด 0	เป็นการขยายพอร์ตอินพุทเอาท์พุท โดยทำงานร่วมกับไอซีชิพรีจิสเตอร์ภายนอกประเภททีทีแอลหรือซีมอส
โหมด 1	ใช้สำหรับการเชื่อมต่ออนุกรมแบบ UART (Universal asynchronous receiver / transmitter) โดยการใช้กลุ่มข้อมูลแบบ 10 บิต และสามารถเปลี่ยนแปลงอัตราความเร็วในการส่งข้อมูลได้
โหมด 2	ใช้สำหรับการเชื่อต่ออนุกรมแบบ UART โดยการใช้กลุ่มข้อมูลแบบ 11 บิตและกำหนดอัตราความเร็วในการส่งข้อมูลคงที่
โหมด 3	ใช้สำหรับการเชื่อมต่อนุกรมแบบ UART โดยการใช้กลุ่มข้อมูลแบบ 11 บิต และสามารถเปลี่ยนแปลงอัตราความเร็วในการส่งข้อมูลได้

นอกจากนี้โหมด 2 และ 3 ยังมีการดำเนินการแบบพิเศษออกไป โดยสามารถนำมาใช้ประโยชน์ในการสื่อสารข้อมูลแบบที่มีไมโครเซสเซอร์หลายตัวทำงานร่วมกันได้ ซึ่งจะได้อธิบายรายละเอียดเป็นลำดับไป

จากแผนภาพในรูปที่ 3.5 ชิพรีจิสเตอร์ภายในตัวส่งจะทำหน้าที่ในการเลื่อนบิตข้อมูลออกไปภายนอกโดยไม่มีการบัฟเฟอร์ และเมื่อใดที่มีการเขียนข้อมูลให้กับรีจิสเตอร์ SBUF แสดงว่ามีความต้องการที่จะส่งข้อมูลนี้ออกไปแบบอนุกรม สำหรับชีพรีจิสเตอร์ทางด้านรับจะทำการเลื่อนบิตข้อมูลที่รับเข้ามาเก็บไว้ เมื่อบิตของข้อมูลที่รับมาครบถ้วนตามจำนวนที่กำหนดไว้ตามการย้ายข้อมูลนี้จะเกิดขึ้น ก็ต่อเมื่อรีจิสเตอร์ SBUF นั้นไม่มีข้อมูลที่จะทำการส่งหรือได้ส่งข้อมูลออกไปเสร็จสิ้นแล้ว

เนื่องจากการส่งหรือรับข้อมูลอนุกรมในการส่งข้อมูลไบต์หนึ่ง ๆ ค่อนข้างจะใช้เวลานาน ๆ หลายมิลลิวินาที ดังนั้นเพื่อให้การจัดการเกี่ยวกับการสื่อสารแบบนี้เป็นไปอย่างมีประสิทธิภาพ 8051 จึงได้กำหนดให้บิตหรือแฟล็กสถานะที่เกี่ยวข้องทั้งหมด จัดรวมอยู่ภายในรีจิสเตอร์ SCON เท่านั้น เช่น แฟล็ก TI ซึ่งจะมีค่าเป็น 1 เมื่อแจ้งให้ทราบว่าได้รับข้อมูลผ่านเข้ามาทางพอร์ตอนุกรม เมื่อแฟล็กตัวใดตัวหนึ่งนี้มีค่าเป็น 1 จะมีผลทำให้เกิดการอินเตอร์รัปต์ขึ้น ดังนั้นภายในโปรแกรมจะต้องทำการตรวจสอบจากสภาวะจองแฟล็กเหล่านี้เองว่ามีการอินเตอร์รัปต์ขึ้นด้วยเหตุใด จากนั้นจึงค่อยทำการกำหนดค่า 0 ให้กับแฟล็กนั้น ลักษณะดังกล่าวนี้จะมีความแตกต่างไปจากการอินเตอร์รัปต์จากสัญญาณอื่น ๆ เช่น วงจรนับ/จับเวลา เป็นต้น ซึ่งจะมีการกำหนดค่า 0 ให้กับแฟล็กสถานะที่เกี่ยวข้องโดยอัตโนมัติ ภายหลังจากที่ได้เข้าไปทำงานยังส่วนของโปรแกรมย่อยบริการอินเตอร์รัปต์ ดังนั้นจึงขอให้สังเกตความแตกต่างในส่วนนี้ไว้ด้วย

การส่งข้อมูลออกทางพอร์ตอนุกรม 8051 จะเริ่มต้นขึ้น ภายหลังเมื่อมีการเขียนข้อมูลลงในรีจิสเตอร์ SBUF ข้อมูลนี้จะถูกจัดการด้วยวิธีการทางด้านฮาร์ดแวร์ในการเลื่อนบิตและส่งสัญญาณออกไปภายนอกโดยอัตโนมัติ เมื่อข้อมูลเหล่านี้ได้ส่งออกครบถ้วนแล้ว จึงจะทำการกำหนดค่าแฟล็ก TI ให้เป็น 1 เพื่อแจ้งให้ทราบว่าขณะนี้รีจิสเตอร์ SBUF ว่าง และพร้อมที่จะส่งข้อมูลไบต์ต่อไปแล้ว ในกรณีที่ผู้ใช้เขียนข้อมูลใหม่ลงในรีจิสเตอร์ SBUF โดยไม่รอให้แฟล็ก TI มีค่าเป็น 1 ก่อนจะมีผลทำให้ข้อมูลที่ส่งออกไปผิดพลาดได้

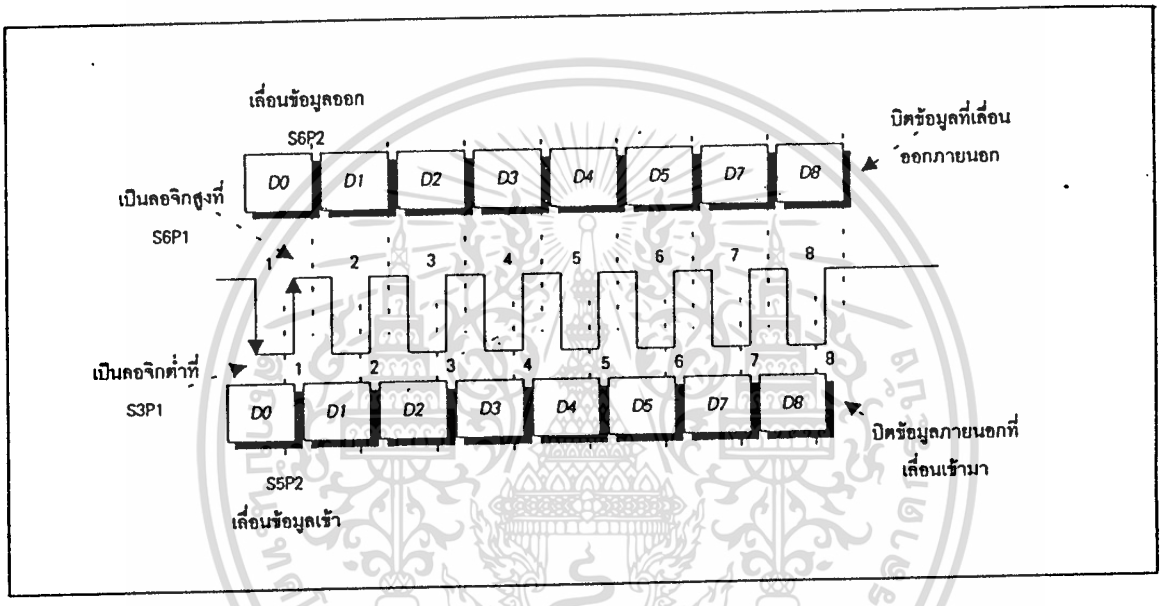
สำหรับการรับข้อมูลจากพอร์ตอนุกรมจะต้องเริ่มต้นโดยการกำหนดค่าบิต REN (Receiver Enable) ให้มีค่าเป็น 1 ก่อน หลังจากนั้นเมื่อมีบิตของข้อมูลถูกส่งเข้ามาจากภายนอกระฮาร์ดแวร์ของ 8051 จึงจะทำการเลื่อนบิตเหล่านี้เข้ามาโดยอัตโนมัติ และเมื่อบิตสุดท้ายถูกเลื่อนเข้ามาเรียบร้อยแล้ว ข้อมูลนั้นจะถูกย้ายมาเก็บไว้ยังรีจิสเตอร์ SBUF และทำการกำหนดให้แฟล็ก RI ให้มีค่าเป็น 1 ซึ่งมีผลทำให้เกิดการอินเตอร์รัปต์โปรแกรมขึ้น

พอร์ตอนุกรมโหมด 0

การทำงานของพอร์ตอนุกรมในโหมด 0 เป็นการขยายพอร์ตอินพุทหรือพอร์ตเอาต์พุท ให้มีจำนวนมากขึ้น โดยจะทำการสร้างสัญญาณนาฬิกาขึ้นเพื่อให้จังหวะของการทำงานที่พร้อมกัน (Synchronizing) สำหรับการเลื่อนบิตเข้าหรือหรือออกจากไอซีรีจิสเตอร์ภายนอก เมื่อมีการโอนย้ายข้อมูลยังริจิสเตอร์ในแต่ละครั้งก็จะมีผลให้เกิดการส่งบิตข้อมูลทั้ง 8 บิตออกมา แม้ว่าแฟล็กสถานะ TI จะยังคงมีค่าเป็น 1 อยู่ก็ตาม

นอกจากนี้แล้วเมื่อใดก็ตามที่ค่าจองแฟล็กสถานะ RI เป็นค่า 1 ก็ควรที่จะย้ายข้อมูลที่รับเข้ามานั้นออกไปจากรีจิสเตอร์ SBUF เสียก่อนที่จะได้มีการกำหนดค่าแฟล็ก RI ให้เป็น 0 เพื่อรับข้อมูลใหม่ต่อไป

การทำงานของพอร์ตอนุกรมในโหมด 0 เป็นการรับและส่งข้อมูลอนุกรมจำนวน 8 บิต โดยใช้เพียงขาสัญญาณ RxD เท่านั้น ส่วนขาสัญญาณ TxD จะนำไปใช้เพื่อเป็นขาสัญญาณนาฬิกาในการให้จังหวะการเลื่อนข้อมูลกับวงจรเลื่อนบิตภายนอก สำหรับอัตราการเลื่อนบิต (หรือ อัตรา บอด) จะถูกกำหนดไว้คงที่ที่ค่า 1/12 ของความถี่ออสซิลเลเตอร์ จากรูปที่ 3.6 แสดงให้เห็นถึงแผนภาพเวลาสัญญาณต่าง ๆ ในโหมด 0 เมื่อมีการรับและส่งข้อมูล 1 ไบต์ โดยสัญญาณนาฬิกาในการเลื่อนบิตนี้จะเกิดขึ้นภายในตัวของ 8051 เอง และมีจุดประสงค์เพื่อนำไปใช้สำหรับวงจรชีพรีจิสเตอร์ภายนอกเท่านั้น



รูปที่ 3.6 แผนภาพเวลาของสัญญาณอนุกรมโหมด 0

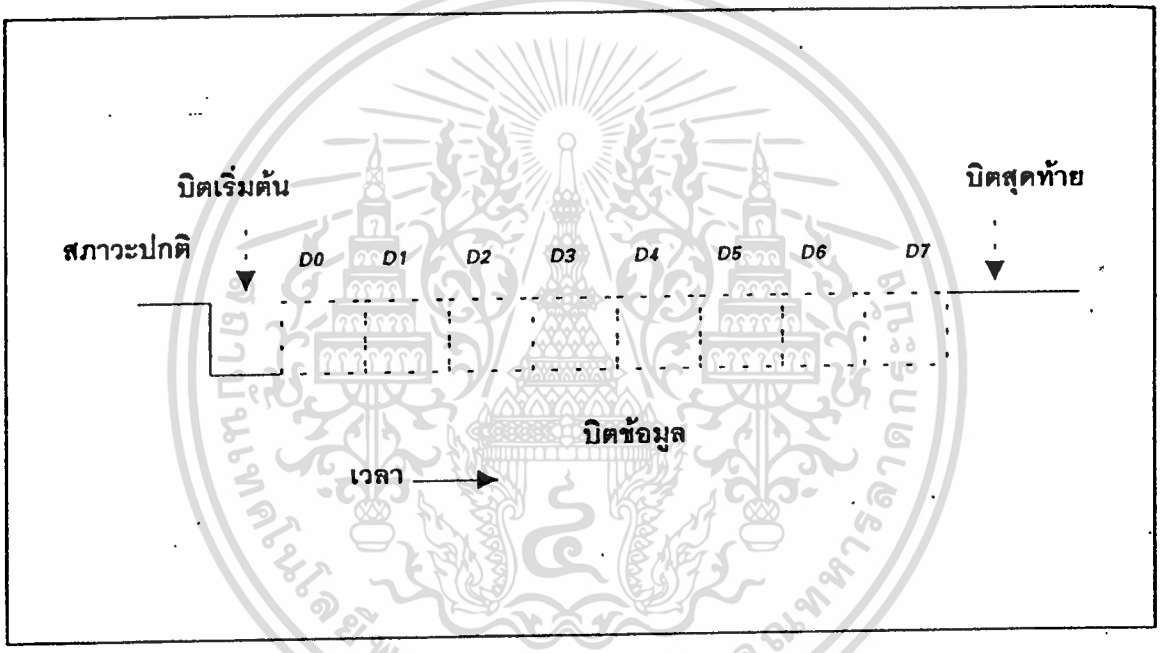
สัญญาณนาฬิกาที่สร้างขึ้นทางขาสัญญาณ TxD นี้จะสลับค่าไปมาจากระดับลอจิกสูงไปต่ำในราวช่วงใกล้เคียงกับเวลาขอบขาลงของสัญญาณ ALE ซึ่งอยู่ในคาบเวลาออสซิลเลเตอร์ที่ 15 ภายหลังจากที่ได้ทำคำสั่งการโอนย้ายข้อมูลมายังรีจิสเตอร์ SBUF หรือคำสั่งที่ทำให้แฟล็กสถานะ RI เป็นค่า 0 หลังจากนั้นสัญญาณนาฬิกานี้ก็จะเปลี่ยนแปลงอีกครั้งราวช่วงใกล้เคียงกับเวลาขอบขาลงสัญญาณ ALE ในคาบเวลาออสซิลเลเตอร์หลังจากนั้นอีก 6 คาบ และจะดำเนินไปในลักษณะเช่นนี้จนกระทั่งข้อมูลทั้ง 8 บิตได้ถูกส่งออกไปเรียบร้อยแล้ว เมื่อสัญญาณขอบขาขึ้นของสัญญาณนาฬิกานี้เกิดขึ้นครบจำนวน 8 ครั้งแล้ว จึงจะมีผลทำให้แฟล็กสถานะ TI หรือ RI มีค่าเป็น 1 ขึ้นและสถานะของขาสัญญาณ TxD ก็จะเป็นระดับลอจิกสูงไปโดยตลอด

ข้อมูลที่จะส่งออกไปภายนอกจะถูกเลื่อนบิตนัยสำคัญต่ำออกไปก่อนเป็นลำดับแรก โดยจะเริ่มขึ้นในเวลาออสซิลเลเตอร์ ภายหลังจากที่ได้ทำคำสั่งการโอนย้ายข้อมูลมายังรีจิสเตอร์ SBUF สำหรับบิตแรกของข้อมูลที่รับเข้ามานั้นจะถูกแล็ตไว้ด้วยขอบขาขึ้นของสัญญาณนาฬิกาในคาบเวลาออสรีจิสเตอร์ที่ 24 ภาย

หลังจากที่มีการกำหนดให้แฟล็กสถานะ RI เป็นค่า 0 หลังจากนั้นในเวลาคาบเวลาออสซิลเลเตอร์อีก 12 คาบถัดมาก็จะได้รับบิตต่อไป ซึ่งจะดำเนินการในลักษณะเช่นนี้จนกระทั่งได้จำนวนบิตข้อมูลครบทั้ง 8 บิต

พอร์ตอนุกรมโหมด 1

การทำงานในโหมด 1 เป็นการสื่อสารข้อมูลอนุกรมจำนวน 10 บิต ประกอบด้วยบิตเริ่มต้นจำนวน 1 บิต บิตข้อมูลจำนวน 8 บิต และบิตสุดท้ายอีก 1 บิต ดังแสดงในรูปที่ 3.7 โดยข้อมูลจะถูกส่งออกทางขาสัญญาณ TxD และรับเข้ามาทางขาสัญญาณ RxD ในส่วนของข้อมูล 8 บิตที่ได้รับหรือทำการส่งออกจะเป็นบิตนัยสำคัญต่ำเป็นลำดับแรก และบิตสุดท้ายของข้อมูลที่รับเข้ามาจะจัดเก็บไว้ในบิต RB8 ภายในรีจิสเตอร์ SCON สำหรับอัตราความเร็วในการส่งข้อมูลของโหมด 1 นั้นสามารถกำหนดเลือกได้



รูปที่ 3.7 รูปแบบของสัญญาณข้อมูลอนุกรมในโหมด 1 ซึ่งมีลักษณะเดียวกับรูปแบบของการสื่อสารแบบอะซิงโครนัส (Asynchronous) ที่ใช้ข้อมูล 8 บิต บิตเริ่มต้นและบิตสุดท้ายอย่างละ 1 บิต

โหมดการทำงานนี้สามารถใช้ในการติดต่อกับพอร์ตสื่อสารอนุกรมแบบ RS - 232C ของเครื่องคอมพิวเตอร์ทั่วไปได้ แต่เนื่องจากว่าจำนวนของข้อมูลในระบบของ RS 232C นั้นอาจจะมีค่าเป็น 7 หรือ 8 บิตได้ ดังนั้นกรณีที่ใช้เป็นข้อมูล 7 บิตและไม่ใช้บิตที่ 8 เป็นบิตพาริตี้ก็อาจจะกำหนดค่าของบิตนี้ให้เป็น 1 ซึ่งจะทำให้ทางด้านรับมองบิตนี้เป็นบิตสุดท้ายไป สำหรับกรณีที่ 8051 เป็นฝ่ายรับข้อมูลของระบบนี้ซึ่งมีเพียง 7 บิต ก็จะมองค่าของบิตสุดท้ายของข้อมูลที่รับมานี้เป็นค่าของข้อมูลบิตที่ 8 แทน และยังคงรอรับบิตสุดท้ายต่อไป อย่างไรก็ตามเนื่องจากว่าระดับสัญญาณของบิตสุดท้ายนี้จะเป็นระดับลอจิกสูงเช่นเดียวกับสภาวะของสายสื่อสารเมื่อไม่มีการส่งข้อมูล ดังนั้นระบบก็จะอ่านค่านี้เข้าไป ซึ่งก็ยังคงถือว่าต้องตามหลักการโดยปริยาย

การส่งข้อมูลจะเกิดขึ้นภายหลังจากเมื่อได้มีการเขียนหรือโอนย้ายข้อมูลเข้าไปยังรีจิสเตอร์ SBUF โดยผู้เขียนโปรแกรมจะต้องทำการตรวจสอบค่าของแฟล็กสถานะ TI ภายในรีจิสเตอร์ SCON ซึ่งจะมีค่าเป็น

1 ภายหลัง ที่ข้อมูลได้เลื่อนบิตออกไปภายนอกเสร็จสิ้นแล้ว สำหรับการรับข้อมูลจะเริ่มต้นขึ้นเมื่อได้มีการกำหนดค่า 1 ให้กับบิต REN และมีการเปลี่ยนแปลงระดับสัญญาณที่ขาสัญญาณ RxD การสุ่มอ่านค่าบิตข้อมูลเข้ามาจะใช้อัตราเดียวเดียวกับอัตราบอดที่ได้กำหนดไว้ในราวช่วงกลางคาบเวลาของบิตหลังจากที่ได้รับข้อมูลครบจำนวน 10 บิตแล้ว และหากมีสภาวะดังในตารางต่อไปนี้เกิดขึ้นก็จะมีผลให้เกิดการย้ายข้อมูลไปเก็บยังรีจิสเตอร์ SBUF

1	แฟล็ก RI มีค่าเป็น 0 (แสดงว่าได้มีการอ่านไบต์ของข้อมูลเข้ามาแล้ว และพร้อมที่จะรับข้อมูลถัดไป) และบิต SM2 มีค่าเป็น 0 เช่นเดียวกัน
2	ค่าของบิตสุดท้ายเป็น 1 (แสดงว่าข้อมูลที่รับเข้ามานั้นถูกต้อง จึงได้โอนย้ายไปเก็บยังรีจิสเตอร์ SBUF โดยไม่สนใจค่าของบิต SM2)

ดังนั้นสรุปได้ว่าข้อมูลที่รับเข้ามาจำนวน 10 บิต นั้น ส่วนของบิตเริ่มต้นไม่ได้มีการนำไปใช้งานต่อไป บิตข้อมูลจำนวน 8 บิต นั้นจะถูกย้ายไปเก็บยังรีจิสเตอร์ SBUF และส่วนของบิตสุดท้ายจะถูกนำไปเก็บในตำแหน่งของบิต RB8 ภายในรีจิสเตอร์ SCON นอกจากนี้ยังมีแฟล็กสถานะ RI ซึ่งจะเป็นค่า 1 เพื่อบอกสถานะว่าได้มีการรับข้อมูลใหม่เข้ามาแล้ว ในกรณีที่โปรมแกรมได้อ่านข้อมูลจากรีจิสเตอร์ SBUF แล้วแต่ไม่ได้กำหนดบิต RI ให้เป็นค่า 0 อีกครั้ง ข้อมูลที่รับเข้ามาใหม่หลังจากนั้นจะสูญหายไป

อัตราการส่งข้อมูลอนุกรมโหมด 1

ดังได้กล่าวแล้วว่าอัตราการส่งข้อมูลอนุกรมโหมด 1 สามารถเปลี่ยนแปลงได้โดยการใช้ Timer 1 หรือ Timer 2 ทำหน้าที่เป็นตัวกำเนิดอัตราการส่งข้อมูลและใช้แฟล็กที่เกิดขึ้นจากการโอเวอร์โฟลว์ ซึ่งในบทนี้จะได้กล่าวเฉพาะการใช้ Timer 1 เท่านั้น สำหรับ Timer 2 ขอให้ดูในหัวข้อที่ 7.6.3.ของบทที่ผ่านมา

กรณีใช้ Timer 1 ทำงานในโหมด 2 (8 - bit automatic reload)

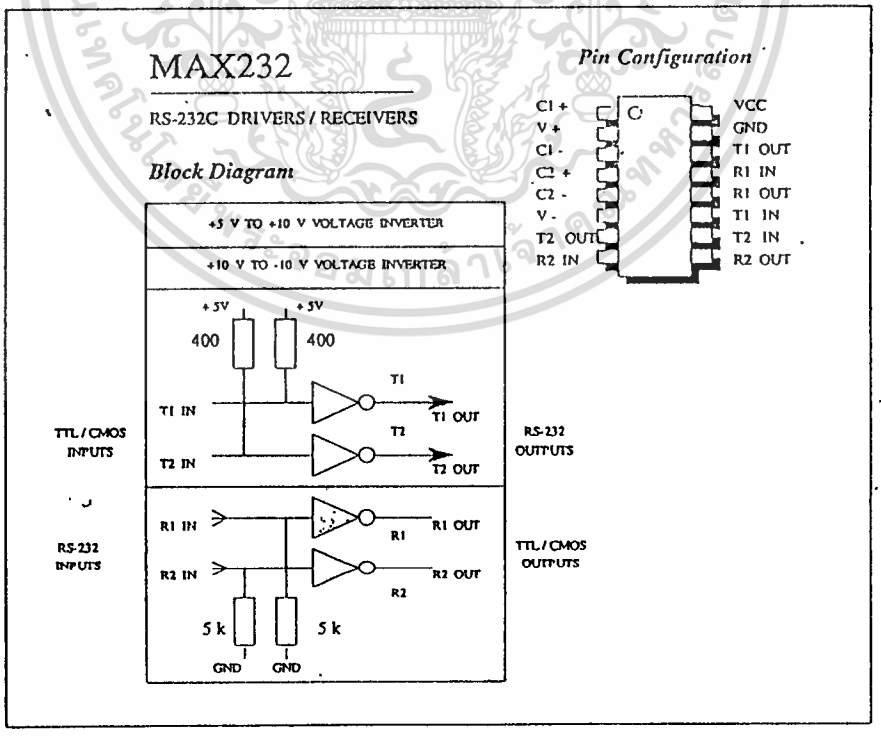
$$\text{ความถี่อัตราบอด} = \frac{2^{\text{SMOD}}}{32} \times \frac{\text{ความถี่ออสซิลเลเตอร์}}{12 \times [256 - TH1]}$$

หมายเหตุ การทำงานของ Timer 1 อาจจะใช้ลักษณะการจับเวลาจากวงจรออสซิลเลเตอร์ภายในหรือการนับสัญญาณภายนอกทางขาสัญญาณ T1 ได้

ที่มาสเตอร์ส่งมา ดังนั้นจึงต้องทำการกำหนดค่าบิต SM2 ให้เป็นค่า 0 มิฉะนั้นจะไม่สามารถรับข้อมูลที่มีบิตที่ 9 เป็นค่าศูนย์เข้ามาได้เลย การสิ้นสุดข้อมูลจะทราบโดยการตรวจสอบว่ามีรหัสเป็นค่า $ ซึ่งทำให้ระบบกลับไปเริ่มต้นการรอรับแอดเดรสในรอบใหม่อีกครั้งหนึ่ง

การเชื่อมต่อแบบมาตรฐาน RS-232C

ในการเชื่อต่อแบบอนุกรมเข้ากับอุปกรณ์คอมพิวเตอร์ต่าง ๆ เช่น คอมพิวเตอร์ เทเล็กซ์หรือโทรพิมพ์ เป็นต้น มักจะกำหนดใช้การเชื่อมต่อตามมาตรฐาน RS-232C ทั้งนี้เพื่อให้มีการใช้งานเส้นสัญญาณหรือรูปแบบของตัวเชื่อมต่อกันทั้งสองด้านให้น้อยลง เนื่องจากระดับโวลเตจที่ใช้และการแทนความหมายของระดับลอจิกตามมาตรฐานนี้แตกต่างไปจากที่ใช้งานกันในระบบดิจิตอลทั่วโดยระดับสัญญาณของ RS-232C เป็นแบบไบโพลาร์ (Bipolar) ระดับโวลเตจทางด้านลบช่วง -3V ถึง -20V แทนค่าลอจิก 1 และโวลเตจทางด้านบวกช่วง +3V ถึง +20V แทนค่าลอจิก 0 ดังนั้นจะเห็นได้ว่ามีความจำเป็นต้องเพิ่มเติมอุปกรณ์หรือวงจรพิเศษเข้าไป เพื่อเปลี่ยนระดับโวลเตจจากระบบ OV ถึง +5V จากขาสัญญาณของ 8051 เป็นระดับโวลเตจที่สูงกว่าค่า +3.0 V หรือต่ำกว่า -3.0 V ดังในรูปที่ 3.8 ซึ่งแสดงให้เห็นว่าระดับสัญญาณแบบ TTL จากขาสัญญาณ TxD และ RxD ของ 8051 จะต้องถูกปรับเปลี่ยนไปเป็นระดับสัญญาณ RS+232 ก่อน ที่จะทำการส่งออกไปใน สายนำสัญญาณต่อไป



รูปที่ 3.8 ไอซีเบอร์ MAX232 ซึ่งเป็นวงจรเชื่อมต่อแบบ RS-232C โดยการใช้ไฟเลี้ยง +5V เพียงชุดเดียว

คุณสมบัติทางไฟฟ้าของสัญญาณ (Electrical Signal Characteristics)

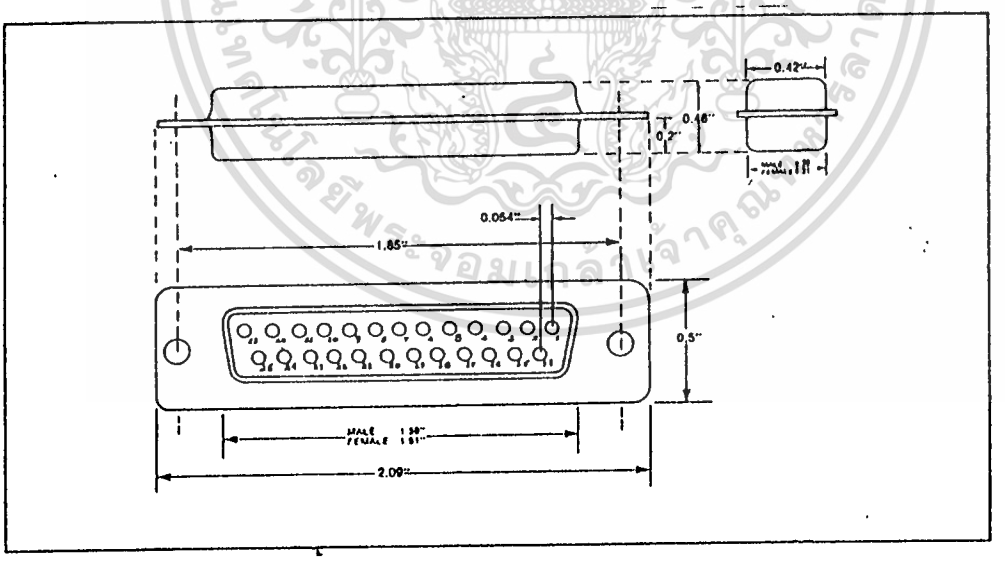
ในส่วนนี้อธิบายถึงรูปแบบของสัญญาณไฟฟ้าที่ตัวอินเตอร์เฟซจะส่งออก และรับเข้ามาจากภายนอก ระดับแรงดันไฟฟ้าที่แสดงถึงตรรกะ 0 และ 1 ก็จะมีกำหนดไว้ในส่วนด้วย

คุณสมบัติทางกลไกการอินเตอร์เฟซะคอน-เน็กเตอร์ (InterfaceMechanical Characteristics : Connectors)

ในหัวข้อนี้กำหนดว่าตัวอินเตอร์เฟซประกอบด้วยส่วนที่เป็นปลั๊ก (plug) และเต้าเสียบ (receptacle) โดยเต้าเสียบจะต้องอยู่บน DCE สำหรับ RS - 232 รุ่น A ถึง C ไม่ได้มีการกำหนดคอนเน็กเตอร์รูปตัว D (D-Shaped) ซึ่งมีใช้กันอยู่ทั่วไปในขณะนั้น ทั้งนี้เพราะว่าอุปกรณ์ตัวนี้ได้รับการคุ้มครองโดยลิขสิทธิ์ และเมื่อสิทธิบัตรนั้นหมดอายุลง ใน RS-232 รุ่น D จึงได้เพิ่มข้อกำหนดคอนเน็กเตอร์ DB-25 เข้าไว้ในมาตรฐานด้วย รายละเอียดของคอนเน็กเตอร์ นี้ได้แสดงไว้ในรูปที่ 3.9

หน้าที่การทำงานของวงจรการแลกเปลี่ยน(Functional Description of Interchange Circuit)

ในส่วนนี้กำหนดหน้าที่และตั้งชื่อให้กับสัญญาณไฟฟ้าต่างๆ ที่นำมาใช้ ตัวอย่างเช่น TRANSMITTED DATA (ข้อมูลส่งออก) ได้ถูกกำหนดไว้ให้กับขา 2 ซึ่งข้อกำหนดนี้มีมากถึง 21 ข้อ แต่มีเพียงไม่กี่ข้อที่เกี่ยวข้องกับไมโครคอมพิวเตอร์



รูปที่ 3.9 แผนภาพคอนเน็กเตอร์ DB-25

มาตรฐานการอินเตอร์เฟซสำหรับระบบการสื่อสารเฉพาะอย่าง (Standard Interfaces Selected Communications System Configurations)

ในส่วนนี้เป็นรายละเอียดต่าง ๆ สำหรับการติดต่อระหว่างโมเด็มกับเทอร์มินัลทั่วไป

บทที่ 4

โครงสร้างของระบบ PC

สถาปัตยกรรมของไอบีเอ็มพีซี

เพื่อความเข้าใจวิธีการทำงานของอะแดปเตอร์อินเตอร์เฟซอนุกรมในระบบพีซี จำเป็นต้องรู้จักอุปกรณ์หลักของไอบีเอ็มพีซีและการออกแบบอุปกรณ์เหล่านั้น หัวข้อต่อไปจะกล่าวถึงชิปโปรเซสเซอร์ บัส การเข้าถึงหน่วยความจำและสายสัญญาณ IRQ

8088 และโปรเซสเซอร์ในตระกูล

ชิปโปรเซสเซอร์ของไอบีเอ็มพีซีรุ่นแรกคือ 8088 ซึ่งเข้ากันได้กับ 8086 ต่อมาไอบีเอ็มพีซีเอทีใช้ 80286 พีซีรุ่นใหม่ส่วนใหญ่ใช้ชิปในอนุกรม 80386 หรือ 80486 ชิปพวกนี้อยู่ในตระกูลเดียวกัน สร้างโดยบริษัท Intel ข้อมูลต่อไปนี้เป็นคุณสมบัติที่มีร่วมกันในชิปทุกตัวในตระกูลนี้

รีจิสเตอร์

รีจิสเตอร์ขนาด 16 บิต มีอยู่สี่ตัวคือ AX,BX,CX และ DX แต่ละตัวสามารถแบ่งแยกออกเป็นรีจิสเตอร์ขนาด 8 บิต ได้เป็น AH, AL, BH, BL, CH, CL, DH และ DL การโหลดไบต์ต่ำของเลข 16 บิต ลงใน AL และไบต์สูงลงใน AH เหมือนกับการโหลดทั้ง 16 บิต ลงใน AX และการโหลด BX, CX และ DX ก็เช่นเดียวกัน

รีจิสเตอร์ขนาดใหญ่ที่สุดมีขนาดเพียง 16 บิต เนื่องจากจำนวน 16 บิต สามารถอ้างหน่วยความจำได้เพียง 64 K การอ้างหน่ายความจำจึงต้องใช้รีจิสเตอร์สองตัว ตัวเลขในรีจิสเตอร์ทั้งสองเรียกว่า เซกเมนต์ (segment) และออฟเซต (offset) ของแอดเดรส เซกเมนต์เป็นแอดเดรสที่มีขอบเขต 16 ไบต์ ภายในหน่วยความจำเซกเมนต์รีจิสเตอร์ 16 บิต สามารถอ้างถึงเซกเมนต์ขนาด 64 K แต่ละเซกเมนต์เริ่มต้นที่ทุก ๆ 16 ไบต์ กล่าวอีกนัยหนึ่งคือ ภายในพื้นที่หน่วยความจำ 1 เมกะไบต์ ออฟเซตเป็นตัวบอกจำนวนไบต์นับนับจากจุดเริ่มต้นของเซกเมนต์

แอดเดรสใด ๆ ภายในเมกะไบต์แรกของหน่วยความจำสามารถอ้างถึงได้โดยใช้เซกเมนต์รีจิสเตอร์ 16 บิต บวกกับออฟเซต 16 บิต ในเซกเมนต์นั้น เซกเมนต์รีจิสเตอร์คือ CS (Code Segment), SS (Stack Segment), DS (Data Segment) และ ES (Extra Segment)

ออฟเซตสามารถใส่ไว้ในรีจิสเตอร์ AX, BX, CX และ DX หรือในรีจิสเตอร์สำหรับออฟเซตภายในโค้ดเซกเมนต์ , SP (Stack Point) และ BP (Base Point) สำหรับออฟเซตภายในสแต็กเซกเมนต์ และ SI (Source Index) และ DI (Destination Index) สำหรับออฟเซตในดาต้าเซกเมนต์

สถาปัตยกรรมบันและการเข้าถึงหน่วยความจำหลัก

บัสเป็นกลุ่มของวงจรที่เชื่อมต่ออุปกรณ์ภายในคอมพิวเตอร์ พีซีมีบัสสามประเภท คือ

บัสข้อมูลมีความกว้าง 8 บิต ซึ่งข้อมูลเดินทางผ่านในแบบขนาน

บัสแอดเดรส มีความกว้าง 20 บิต สำหรับการส่งแอดเดรส

บัสควบคุม ประกอบด้วยวงจรสำหรับควบคุมอุปกรณ์ โดยทั่วไปการควบคุมเหล่านี้เป็นคำสั่ง บอกว่า ควรทำอะไรกับข้อมูลบนบัสข้อมูลและบัสควบคุมเมื่อโปรเซสเซอร์ต้องการอ่านหน่วยความจำแอดเดรสจะถูกส่งออกมาตามบัสแอดเดรสและเซนสัญญาณบานสายสัญญาณอ่านของบัสควบคุม อุปกรณ์หน่วยความจำที่เกี่ยวข้องจะใส่ข้อมูลในแอดเดรสนั้นลงบนบัสข้อมูลเพื่อส่งกลับไปให้โปรเซสเซอร์

ในกระบวนการส่งข้อมูลไปยังหน่วยความจำจะกลับกันกับกระบวนการนี้สัญญาณบนสายสัญญาณเขียนบนบัสควบคุมจะถูกเซต แอดเดรสถูกใส่บนบัสแอดเดรสและข้อมูลที่ต้องการเขียนจะถูกใส่ลงบนบัสข้อมูล

แอดเดรส I/O

นอกจากหน่วยความจำหลัก 1 เมกะไบต์ ซึ่งเข้าถึงได้ในลักษณะเซกเมนต์และออฟเซตแล้ว ยังมีหน่วยความจำ I/O หรือแอดเดรสของพอร์ตอีก 768 ตำแหน่ง สำหรับใช้กับอุปกรณ์ต่าง ๆ

หน่วยความจำ I/O สามารถเข้าถึงได้โดยการส่งคำสั่ง IN และ OUT ไปให้ซีพียู ตัวอย่างเช่น การส่งค่าในรีจิสเตอร์ AL ไปที่พอร์ต 3F8H ใช้คำสั่ง OUT 3F8H, AL

การอ่านจากพอร์ตซึ่งแอดเดรสของมันอยู่ใน DX และใส่ผลลัพธ์ลงในรีจิสเตอร์ AL คำสั่งคือ IN AL, DX

วิธีการทำงานคำสั่ง IN และ OUT ในภาษาต่าง ๆ จะแตกต่างกัน ในบทต่อไปจะอธิบายเกี่ยวกับวิธีเหล่านี้

เมื่อซีพียูได้รับสั่งดังกล่าว มันจะส่งสัญญาณไปตามสายสัญญาณอ่าน I/O หรือสายสัญญาณเขียน I/O ตามความเหมาะสม เมื่ออุปกรณ์ I/O ตรวจพบสัญญาณบนสายสัญญาณเหล่านี้ มันต้องตรวจสอบบัสแอดเดรสว่าคำสั่งนั้นสำหรับมันหรือเปล่า เนื่องจาก I/O มีแอดเดรสที่สูงที่สุดคือ 300 H การเชื่อมต่อบัสแอดเดรสกับอุปกรณ์ I/O จึงต้องการเพียง 9 บิต ล่างเท่านั้น

อุปกรณ์ I/O สามารถใส่ข้อมูลบนบัสข้อมูลได้เช่นเกี่ยวกับการเข้าถึงหน่วยความจำหลัก เมื่อได้รับสัญญาณอ่าน และรับข้อมูลจากบัสข้อมูลเมื่อได้รับสัญญาณเขียน อย่างไรก็ตามอุปกรณ์ I/O อาจทำมากกว่าการใส่ข้อมูลบนบัสข้อมูล เช่น เมื่อ 8250 รับสัญญาณอ่านที่อ้างถึงบัฟเฟอร์รับข้อมูล มันไม่เพียงส่งข้อมูลในบัฟเฟอร์ออกไปเท่านั้น แต่ยังเคลียร์บัฟเฟอร์ด้วยเพื่อไม่ให้ข้อมูลเดียวกันนี้ถูกอ่านอีกในครั้งต่อไป UART อาจรีเซตอินเตอร์รัปต์ที่มันสร้างขึ้นด้วย

อินเตอร์รัปต์

ในการที่ผ่านมาพูดถึงความแตกต่างระหว่างฮาร์ดแวร์และซอฟต์แวร์อินเตอร์รัปต์ สำหรับการโปรแกรมในระดับของระบบ เกี่ยวข้องกับฮาร์ดแวร์อินเตอร์รัปต์เท่านั้น

บนพีซีมีสายสัญญาณอินเตอร์รัปต์จำนวนหนึ่ง อินเตอร์รัปต์ถูกสร้างโดยการทำให้แรงดันไฟฟ้าบนวงจรที่เหมาะสมเพิ่มขึ้น และรักษาระดับแรงดันนั้นไว้จนกว่าอินเตอร์รัปต์ถูกรีเซต หรือถูกตอบรับ

ไอบีเอ็มพีซีรุ่นแรกมีสายสัญญาณ IRQ 8 เส้น ดังตารางที่ 4.1 พีซีเอทีและพีซีรุ่นหลังมีสายสัญญาณ IRQ 16 เส้น ดังในตารางที่ 4.2

ตารางที่ 4.1 สายสัญญาณ IRQ ของไอบีเอ็มพีซี

สายสัญญาณ IRQ	อุปกรณ์
NMT	Nonmaskable interrupt
0	ไทเมอร์
1	แป้นพิมพ์
2	สงวนไว้
3	พอร์ตอนุกรม 2
4	พอร์ตอนุกรม 1
5	ฮาร์ดดิสก์
6	ฟลอปปีดิสก์
7	พอร์ตขนาน

8259 A PIC

สายสัญญาณ IRQ ไม่ได้ต่อโดยตรงกับพีซียูแต่ต่ออยู่กับชิปควบคุมอินเตอร์รัปต์ ชิปตัวนี้คือ 8259 A PIC (Programmable Interrupt Control) ไอบีเอ็มพีซีเอทีและพีซีรุ่นหลังมีตัวควบคุมนี้สองตัวต่อกันแบบดีซีเชน (daisy - chain) PIC เป็นตัวจัดลำดับความสำคัญของอินเตอร์รัปต์และป้องกันความวุ่นวายที่จะเกิดขึ้นถ้ามีอินเตอร์รัปต์เกิดขึ้นหลายตัวพร้อมกัน

ตารางที่ 4.2 สายสัญญาณอินเตอร์รัปต์ของไอบีเอ็มพีซีเอที

สายสัญญาณ IRQ	อุปกรณ์
NMT	Nonmaskable interrupt
0	ไทเมอร์
1	แป้นพิมพ์
2	เชื่อมโยงจากตัวควบคุม 1 ไป 2

ตารางที่ 4.2 (ต่อ) สายสัญญาณอินเตอร์รัปต์ของไอบีเอ็มพีซีเอที

สายสัญญาณ IRQ	อุปกรณ์
3	พอร์ตอนุกรม 2
4	พอร์ตอนุกรม 1
5	พอร์ตขนาน 2
6	ฟลอปปีดิสก์
7	พอร์ตขนาน 1
8	นาฬิกา
9	เปลี่ยนทิศทางจาก IRQ 2 เดิม
10	สงวนได้
11	สงวนได้
12	สงวนได้
13	โปรเซสเซอร์ร่วม (Coprocessor)
14	ฮาร์ดดิสก์
15	สงวนไว้

RQo ถูกกำหนดให้มีลำดับความสำคัญสูงสุด และ IRQ7 มีลำดับความสำคัญต่ำสุด ถ้าอุปกรณ์หลายตัวส่งสัญญาณอินเตอร์รัปต์พร้อมกัน อินพุทที่เข้าสู่ตัวควบคุมจะเป็นได้หลายตัว ซึ่งมันจะถูกส่งไปให้ซีพียูตามลำดับความสำคัญ

การอีนาเบิลอินเตอร์รัปต์

PIC มีรีจิสเตอร์ตัวหนึ่งสำหรับอีนาเบิลอินเตอร์รัปต์ต่างๆ IRQ3 และ IRQ4 จะไม่ถูกอีนาเบิลโดยปริยาย ดังนั้นถ้าต้องการสื่อสารโดยใช้อินเตอร์รัปต์ ต้องสั่ง PIC ให้อีนาเบิลอินเตอร์รัปต์ที่เหมาะสม โดยการอ่านรีจิสเตอร์ด้วยคำสั่ง IN จากพอร์ต 21 H เซตบิตที่ต้องการเป็นศูนย์ (บิต 4 สำหรับ IRQ4 และบิต 3 สำหรับ IRQ3) และเขียนค่ากลับลงไปที่รีจิสเตอร์ด้วยคำสั่ง OUT ไปยังพอร์ต 21 H เมื่อ PIC ได้รับสัญญาณอินเตอร์รัปต์ โดยที่ไม่มีอินเตอร์รัปต์อื่นค้างอยู่มันจะส่งแรงดันไฟฟ้าบวกไปบนสายสัญญาณ INT ที่ต่อกับซีพียู ซีพียูตอบรับการอินเตอร์รัปต์ด้วยการส่งสัญญาณไปยัง PIC จะส่งหมายเลขซึ่งเกิดจากการนำเอาหมายเลข IRQ บวกกับ 8 ไปให้ซีพียู เช่น ถ้าเป็น IRQ4 มันจะส่ง 12 (OCH) ถ้าเป็น IRQ4 มันจะส่ง 12 (OCH) ถ้าเป็น IRQ3 มันจะส่ง 11 (OBH) จากนั้นซีพียูจึงทำงานตามคำสั่งสำหรับอินเตอร์รัปต์นั้น โดยจัดเก็บแอดเดรสของโปรแกรมในขณะนั้นไว้ในสแต็ก และทำคำสั่ง far call ไปยังตำแหน่งของหน่วย

ความจำที่ถูกชี้โดยอินเตอร์รัปต์เวกเตอร์สำหรับอินเตอร์รัปต์นั้น ดังนั้นอินเตอร์รัปต์บน IRQ4 จึงมีผลเหมือนกับการทำคำสั่ง INT OCH ในซอฟต์แวร์

ถ้าไม่ต้องการให้ซีพียูถูกอินเตอร์รัปต์ เนื่องจากกำลังทำงานที่สำคัญมากก็สามารถสั่งให้มันไม่สนใจอินเตอร์รัปต์ได้โดยการใช้คำสั่ง CLI (olear Interrupt Fialg) การอินเตอร์รัปต์จะเกิดขึ้นได้อีกด้วยคำสั่ง STI (Set Interrupt Flag) ผลจากคำสั่งทั้งสองจะทำให้ IF (Interrupt Flag) ในซีพียูเปลี่ยนไป

การเซต IF ไม่ได้ทำให้เกิดความเปลียนแปลงภายใน PIC ซึ่งจะยังคงส่งสัญญาณมาบนสายสัญญาณ INT ตามปกติเมื่อมันได้รับอินเตอร์รัปต์จากสายสัญญาณ IRQ การเซต IF เพียงแต่ป้องกันซีพียูไม่ให้ตอบรับการอินเตอร์รัปต์เท่านั้น อย่างไรก็ตาม สายสัญญาณ INT จะยังคงเป็นได้ และซีพียูจะสามารถจัดการกับอินเตอร์รัปต์ได้อีกเมื่อทำคำสั่ง STI

อินเตอร์รัปต์อื่นที่ค้างอยู่ก็ไม่จำเป็นต้องสูญหายไป อุปกรณ์จะดูแลสายสัญญาณ IRQ ให้เป็นได้ และ PIC ก็รู้ว่าอุปกรณ์นั้นต้องการความสนใจเมื่ออินเตอร์รัปต์ถูกวีนาเบิลอีกครั้ง PIC จะส่งอินเตอร์รัปต์ ที่ค้างอยู่ไปตามลำดับความสำคัญ อย่างไรก็ตามความสามารถที่จะห้ามอินเตอร์รัปต์ชั่วคราวนี้อาจทำให้ข้อมูลสูญหายได้ เนื่องจากอุปกรณ์จะไม่สร้างสัญญาณอินเตอร์รัปต์ใหม่ (เช่น ถ้ามีตัวอักษรใหม่ได้รับเข้ามาโดย UART) จนกว่า อินเตอร์รัปต์แรกจะถูกตอบรับ

มีสองสาเหตุที่ทำให้อินเตอร์รัปต์จากอุปกรณ์อาจไม่ได้รับความสนใจจากพีซียูโดยทันที คือมันอาจจะซ้ำซ้อนกับอินเตอร์รัปต์ของอุปกรณ์อื่นที่มี ลำดับความสำคัญสูงกว่า หรืออินเตอร์รัปต์อาจถูกห้ามไว้ ถึงกระนั้นก็ตามการใช้วิธีอินเตอร์รัปต์โดยปกติก็เร็วกว่าการรอคอยการทำงานในรูปของโปรแกรมที่ซับซ้อน

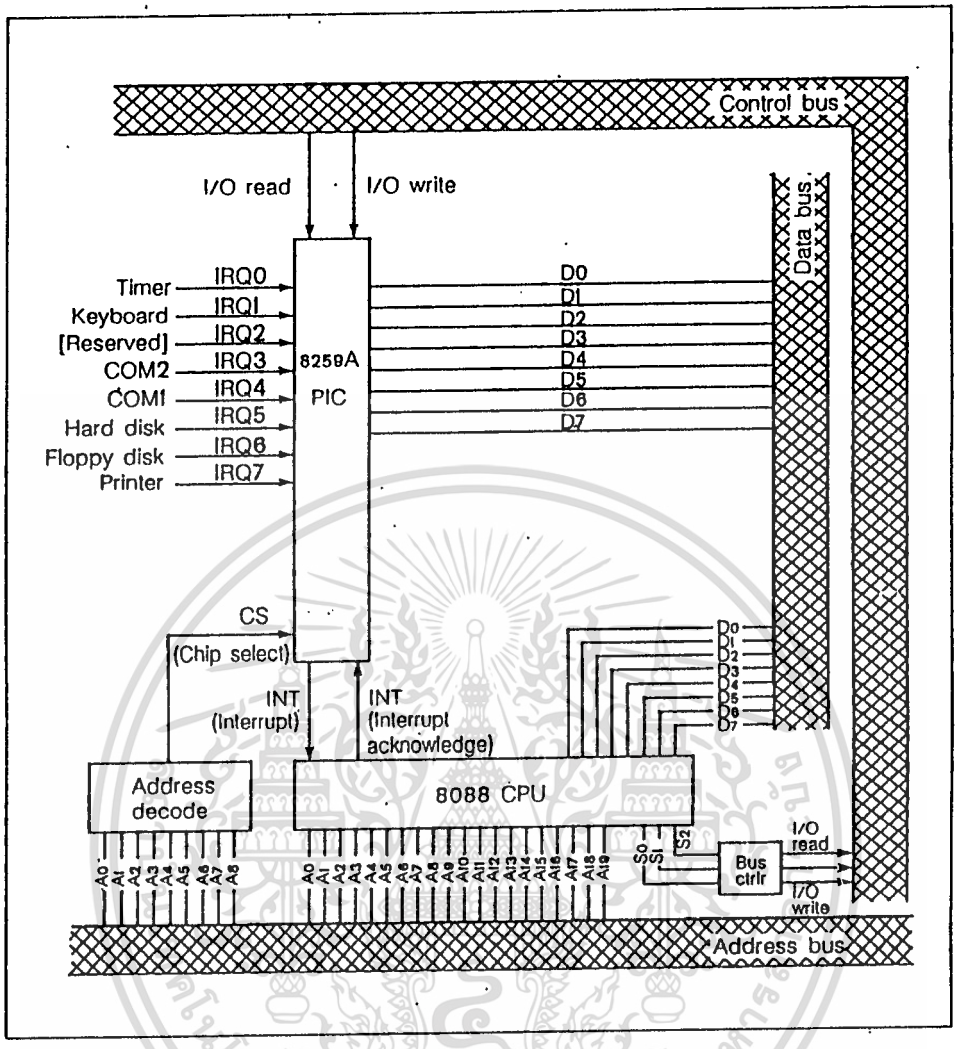
การตอบรับการอินเตอร์รัปต์

ซีพียูตอบรับการอินเตอร์รัปต์จาก PIC เพื่อสั่งให้ PIC แจ้งว่าอินเตอร์รัปต์ใดที่เกิดขึ้น ซอฟต์แวร์จัดการอินเตอร์รัปต์ (ซึ่งมีแอดเดรสอยู่ในตารางออนเตอร์รัปต์เวกเตอร์) ต้องตอบรับการอินเตอร์รัปต์ไปยัง PIC ด้วยตัวมันเอง โดยการส่งค่า 20 H (รู้จักกันในชื่อ End of Interrupt หรือ EOI) ไปยังพอร์ต 20 H ถ้าไม่ทำเช่นนี้ PIC จะไม่ส่งอินเตอร์รัปต์มายังซีพียูอีก)

ในกรณีของเครื่องที่มี PIC สองตัว IPQ8 ถึง 16 ได้รับมาจาก PIC ตัวที่สอง โดยผ่านทาง IPQ2 ของ PIC ตัวแรก จากนั้น PIC ตัวแรกจะแจ้งไปยังซีพียู ตัวจัดการอินเตอร์รัปต์ที่จัดการกับ IPQ8 ถึง 15 จึงต้องส่ง EOI สองครั้ง สำหรับ PIC แต่ละตัว

รูปที่ 13.1 เป็นสรุปของลอจิกในการจัดการอินเตอร์รัปต์บนพีซี ภาพนี้เป็นเวอร์ชั่นอย่างง่ายมีพื้นฐานจากพีซีรุ่นแรก โดยการเชื่อมต่อขาถูกจัดเรียงใหม่ เพื่อความชัดเจน วงจรที่สมบูรณ์สามารถดูได้จากคู่มืออ้างอิงทางเทคนิคของเครื่อง

รูปที่ 4.1 โครงสร้างอินเตอร์รัปต์ของพีซี



แผงวงจรอินเตอร์เฟซอนุกรม

พีซีเกือบทั้งหมด ใช้ **UART** ตัวใดตัวหนึ่งที่อธิบายไว้ในบทที่ 12 ส่วนประกอบอื่นที่เกี่ยวข้องกับการจัดการ **UART** ได้แก่

การเชื่อมต่อทางกายภาพกับแผงวงจรหลักของคอมพิวเตอร์

การเชื่อมต่อกับโลกภายนอกผ่านซ็อกเกตที่เหมาะสม

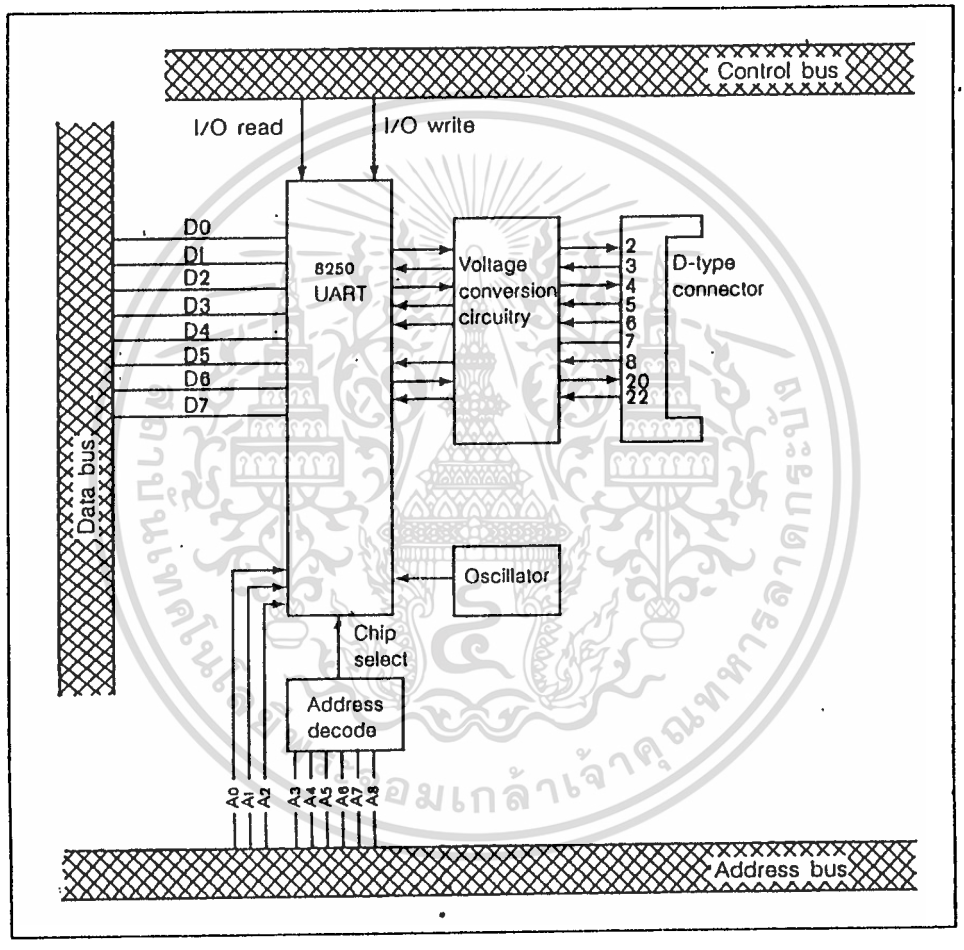
การแปลงสัญญาณไฟฟ้าระหว่างระดับแรงดันไฟฟ้า ที่ใช้ใน คอมพิวเตอร์กับที่ใช้ในวงจร **RS** - 232 - **C**

สำหรับนักเขียนโปรแกรมแล้ว ไม่จำเป็นต้องต้องสนใจส่วนประกอบเหล่านี้ในรูปที่ 4.2 แสดงวงจรหลักของอะแดปเตอร์อินเตอร์เฟซอนุกรม ซึ่งวงจรที่สมบูรณ์จะดูได้จากคู่มืออ้างอิงทางเทคนิค คู่มือเหล่านี้บรรจุข้อมูลอ้างอิงที่มีประโยชน์มากเกี่ยวกับอะแดปเตอร์อะซิงโครนัส และอะแอปเตอร์อนุกรมกับเครื่องพิมพ์ของไอบีเอ็ม ซึ่งนำไปใช้ได้กับแผงวงจรอนุกรมอื่น

อินเตอร์รัปต์ของอะแดปเตอร์อนุกรม

ฮาร์ดแวร์อินเตอร์รัปต์หมายเลข 4 ถูกกำหนดให้กับอะแดปเตอร์การสื่อสารแบบอะซิงโครนัสตัวแรก อินเตอร์รัปต์หมายเลข 3 ถูกกำหนดให้กับอะแดปเตอร์การสื่อสารแบบอะซิงโครนัสตัวที่สอง ดังนั้นแผงวงจรอนุกรมหรือโมเด็มภายในที่ถูกทำให้เป็นอะแดปเตอร์ตัวแรกอินเตอร์รัปต์ที่ UART ของมันสร้างขึ้นจะถูกส่งไปตาม IRQ4

รูปที่ 4.2 แผงผังลอจิกของอินเตอร์เฟซอนุกรม



ลำดับความสำคัญของอินเตอร์รัปต์ภายใน INS 8250 และ UART อื่น ที่เข้ากันได้ ถูกกำหนดไว้ตายตัว Receive Line Status มีลำดับความสำคัญสูงสุด ตามด้วย Received Data Available จากนั้นเป็น Trans - mitter Holding Register Empty และสุดท้ายคือ Modem Status อย่าลืมว่าเหตุการณ์ที่เกิดขึ้นจะไม่กระตุ้นการอินเตอร์รัปต์ หากไม่มีการโปรแกรมรีจิสเตอร์อีนาเบิลอินเตอร์รัปต์เพื่ออีนาเบิลได้ก่อน

อินเตอร์รัปต์อื่นจะไม่ถูกสร้างขึ้นจนกว่าอินเตอร์รัปต์ตัวแรกจะถูกรีเซตอินเตอร์รัปต์ถูกรีเซต เมื่อ

Received Line Status ถูกรีเซตด้วยการอ่านรีจิสเตอร์แสดงสถานะสายสื่อสาร

Received Data Available ถูกรีเซตด้วยการอ่านรีจิสเตอร์พักข้อมูลรับ

Transmitter Holding Empty ถูกรีเซตด้วยการอ่านรีจิสเตอร์จำแนกอินเตอร์รัปต์ (ถ้าอินเตอร์รัปต์เกิดจากสาเหตุนี้หรือด้วยการเขียนข้อมูลไปยังรีจิสเตอร์พักข้อมูลส่ง)

Modem Status ถูกรีเซตด้วยการอ่านรีจิสเตอร์แสดงสถานะโมเด็ม

แอดเดรส I/O ของอะแดปเตอร์นุกรม

พอร์ต I/O สำหรับการสื่อสารแบบอนุกรมในพีซี เริ่มต้นที่แอดเดรส 3F8H สำหรับอะแดปเตอร์ตัวแรก และ 2F8 สำหรับอะแดปเตอร์ตัวที่สอง การอ้างถึงรีจิสเตอร์แต่ละตัวใน UART จะใช้การบวกออฟเซตเข้ากับเบสแอดเดรสนี้

ดังที่กล่าวมาแล้วว่า การอ้างแอดเดรส I/O นั้นใช้บัสแอดเดรสเพียง 9 บิตล่างเท่านั้น และทั้ง 3F8H และ 2F8H มีสามบิตล่างเป็นศูนย์ ดังนั้นออฟเซตที่บวกเข้าไปจึงมีได้ตั้งแต่ 0 ถึง 7 และบิตที่เปลี่ยนแปลงจะไม่เกินบิต 2 วงจรแยกแอดเดรสบนอะแดปเตอร์จึงตรวจสอบเฉพาะบิต 3 ถึง 8 บนบัสแอดเดรส ส่วนบิต 0 ถึง 2 จะถูกส่งต่อโดยตรงกับชิป UART

เมื่อวงจรแยกแอดเดรสพบแอดเดรสในช่วง 3F8H ถึง 3FFH หรือ 2F8H ถึง 3FFH หรือ 2F8H ถึง 2FFH ถ้าเป็น CON2 มันจะส่งสัญญาณไปที่ขา Chip Seleot Pin ของ UART แล้ว UART จึงดูที่บิต 0, 1 และ 2 ของบัสแอดเดรส ว่า เป็นแอดเดรสใดและดูสัญญาณการเขียนและการอ่าน I/O บนบัสควบคุม เพื่อตัดสินว่าควรทำอะไรต่อไป แบสแอดเดรสและออฟเซตของ COM1 และ COM2 แสดงไว้ในตารางที่ 4.3

ตารางที่ 4.3 แอดเดรส I/I สำหรับ COM1 และ COM1

ออฟเซต	อะแดปเตอร์ตัวแรก	อะแดปเตอร์ตัวที่สอง	รีจิสเตอร์
0	3F	2F8	รีจิสเตอร์พักข้อมูลส่ง
0	3F8	2F8	รีจิสเตอร์พักข้อมูลรับ
0	3F8	2F8	แลตช์ตัวหาร LSB
1	3F9	2F9	แลตช์ตัวหาย MSB
1	3F9	2F9	รีจิสเตอร์อีนาเบิลอินเตอร์รัปต์
2	3FA	2FA	รีจิสเตอร์จำแนกอินเตอร์รัปต์
3	3FB	2FB	รีจิสเตอร์ควบคุมสายสื่อสาร
4	3FC	2FC	รีจิสเตอร์ควบคุมโมเด็ม
5	3FD	2FD	รีจิสเตอร์แสดงสถานะสายสื่อสาร
6	3FE	2FE	รีจิสเตอร์แสดงสถานะโมเด็ม

สังเกตว่ารีจิสเตอร์บางตัวใช้แอดเดรสร่วมกัน ความสับสนระหว่างรีจิสเตอร์พักข้อมูลส่งและรีจิสเตอร์พักข้อมูลรับไม่เกิดขึ้น เพราะคำสั่ง OUT จะเข้าถึงรีจิสเตอร์พักข้อมูลส่ง และคำสั่ง IN จะเข้าถึงรีจิสเตอร์พักข้อมูลรับแลตช์ตัวหารก็ใช้แอดเดรสร่วมกับรีจิสเตอร์อื่นเช่นกัน การเลือกแลตช์ตัวหารใช้บิตควบคุมการเข้าถึงแลตช์ตัวหารก็ใช้แอดเดรสร่วมกับจีริสเตอร์อื่นเช่นกัน การเลือกแลตช์ตัวหารใช้บิตควบคุมการเข้าถึงแลตช์ตัวหาร (DLAB) ในรีจิสเตอร์ควบคุมสายสื่อสาร เมื่อ DLAB ถูกเซตเป็น 1 จะเป็นการเข้าถึงแลตช์ตัวหารเมื่อ DLAB เป็น 0 จะเป็นการเข้าถึงรีจิสเตอร์อื่นในแอดเดรสเดียวกัน

โพลลิ่ง

การเขียนโปรแกรมโดยใช้โพลลิ่งต้องแน่ใจว่าโปรแกรมจะทำคำสั่ง IN เป็นวงรอบเพื่อตรวจสอบการรับข้อมูลหรือเหตุการณ์อื่นที่เกี่ยวข้องกันว่าเกิดขึ้นหรือไม่ โดยอ่านจากรีจิสเตอร์แสดงสถานะที่เหมาะสมของ UART ถ้ามีการรับตัวอักษรโปรแกรมควรสั่งคำสั่ง IN อีกครั้งไปยังแอดเดรสของรีจิสเตอร์พักข้อมูลรับ ถ้าตัวอักษรถูกส่งออกไปแล้ว ตัวอักษรตัวต่อไปจะถูกส่งด้วยการใช้คำสั่ง OUT ไปยังรีจิสเตอร์พักข้อมูลส่ง

โปรแกรมบริการอินเตอร์รัปต์

พีซีมีตารางอินเตอร์รัปต์เวกเตอร์ ซึ่งเก็บแอดเดรส 4 ไบต์ ของแต่ละอินเตอร์รัปต์ การกำหนดค่าในอินเตอร์รัปต์เวกเตอร์ ตอนแรกต้องตรวจสอบว่ามีการอินเตอร์รัปต์สำหรับอินเตอร์รัปต์นั้นติดตั้งอยู่ก่อนหรือไม่โดยใช้อินเตอร์รัปต์ 21H ฟังก์ชัน 35H และใส่หมายเลขอินเตอร์รัปต์ลงใน AL, 35H ลงใน AH และสั่ง INT 21H ค่าแอดเดรสของโปรแกรมบริการอินเตอร์รัปต์ (interruptservice routine) จะถูกส่งกลับมาใน ES:BX เหตุผลที่ต้องการตรวจสอบว่ามีตัวจัดการอินเตอร์รัปต์ถูกติดตั้งอยู่ก่อนหรือไม่มีสองประการ ประการแรก เมื่อเลิกใช้ตัวจัดการอินเตอร์รัปต์ของคุณแล้วต้องคืนค่าตัวชี้ไปยังตัวจัดการอินเตอร์รัปต์เดิมไว้ในตารางอินเตอร์รัปต์เวกเตอร์ ประการที่สองพีซีบางเครื่องสนับสนุน การใช้ IRQ ร่วมกันหมายความว่า อุปกรณ์สองตัวขึ้นไปสามารถใช้สายสัญญาณ IRQ ร่วมกันได้ ในกรณีนี้อุปกรณ์แต่ละตัวอาจมีตัวจัดการอินเตอร์รัปต์ของมันเองและตัวจัดการอินเตอร์รัปต์ของคุณควรจะเรียกใช้ตัวจัดการตัวเก่าทุกครั้งที่เกิดอินเตอร์รัปต์

เมื่อได้ตรวจสอบแอดเดรสของตัวจัดการที่มีอยู่ก่อนแล้ว ก็สามารถติดตั้งตัวจัดการใหม่ลงไปได้ โดยใช้อินเตอร์รัปต์ 21H ฟังก์ชัน 25H แอดเดรสของตัวการอินเตอร์รัปต์นำไปใส่ไว้ใน DS:DX ใส่ค่า 25H ลงใน AH, ใส่อินเตอร์รัปต์เวกเตอร์ลงใน AL แล้วสั่ง INT 21H

เมื่อ ISR ถูกเรียกใช้ เริ่มต้นมันจะเก็บค่าของรีจิสเตอร์ไว้บนสแต็ก เพื่อที่ว่าเมื่อจบงานแล้วจะได้คืนสถานะของโปรเซสเซอร์ให้เหมือนเดิมก่อนเกิดอินเตอร์รัปต์ ต่อจากนั้น ISR ต้องเรียกตัวจัดการอินเตอร์รัปต์ที่ถูกติดตั้งอยู่ก่อน แล้วจึงตรวจหาสาเหตุของการอินเตอร์รัปต์ และตอบสนองให้สอดคล้องกัน IRQ ต้องตอบรับการอินเตอร์รัปต์ไปให้ PIC (ทั้งสองตัวถ้าหมายเลข IRQ ตั้งแต่ 8 ขึ้นไป) สุดท้ายมันต้องคืนค่าของรีจิสเตอร์และจบด้วยคำสั่ง IRET หรือ Interrupt Return

ปัญหาบนพีซีความเร็วสูง

ถ้าหากพีซีทำงานเร็วกว่า 8250 มากเกินไปอาจทำให้คำสั่งที่ส่งติดต่อกันเร็วกว่าที่ 8250 สามารถประมวลผลได้ จากสาเหตุนี้จึงควรหลีกเลี่ยงการส่งคำสั่งติดต่อกันไปให้ 8250 ควรใส่คำสั่ง Short Jump ไว้ระหว่างคำสั่งเข้าถึงชิปที่ต่อเนื่องกันเพื่อลดความเร็ว วิธีนี้ใช้ได้เฉพาะกับการเขียนโปรแกรมด้วยภาษาแอสเซมบลีเท่านั้น

บทที่ 5

การสื่อสารของ PC โดย INT 14

ฟังก์ชันภายในจำนวนหนึ่งซึ่งสามารถถูกเรียกใช้ได้จากโปรแกรมได้ถูกจัดเตรียมมาพร้อมกับพีซี ฟังก์ชันเหล่านี้เรียกว่า BIOS funotions ซึ่งถูกเก็บไว้ในรอม (ROM) ของพีซีโปรแกรมไม่จำเป็นต้องใช้ BIOS funotion เสมอไป บางครั้งอาจใช้วิธีการอื่นซึ่งมักจะเร็วกว่าแต่ให้ผลลัพธ์เช่นเดียวกัน ตัวอย่างเช่น มีโปรแกรมทางการค้าน้อยมากที่ใช้ BIOS funotion ในการเขียนหน้าจอเพราะว่าการเคลื่อนย้ายข้อความไปยังบัฟเฟอร์ของจอภาพโดยตรง จะทำให้แสดงผลได้เร็วกว่ามากกับเรื่องการสื่อสารแบบอนุกรม การมองข้าม BIOS และควบคุมพอร์ตอนุกรมโดยตรงก็กลายเป็นวิธีมาตรฐานไปแล้วถึงกระนั้นก็ตาม BIOS funotion ก็มีประโยชน์เมื่อไม่มีเวลาพอที่จะเขียนโปรแกรมให้ใช้ฟังก์ชันในระดับต่ำ และเมื่อประสิทธิภาพไม่ใช่สิ่งสำคัญ

ในบทนี้อธิบายข้อดีและข้อเสียของ BIOS funotion ที่เกี่ยวกับการสื่อสารแบบอนุกรม คือ BIOS funotion ในอินเตอร์รัปต์ 14 ซึ่งโปรแกรมสื่อสารอื่นบางตัวก็ใช้อินเตอร์รัปต์เดียวกันนี้เป็นทางติดต่อกับโปรแกรมแอปพลิเคชัน เช่น LAN Workplace DOS ของ Novell ออกแบบ มาเพื่อทำให้พีซีสามารถสื่อสารกับโฮสต์คอมพิวเตอร์ที่ใช้ยูนิกซ์ได้ เมื่อทำการติดตั้งซอฟต์แวร์ โปรแกรม TSR จะถูกโหลดเพื่อทำให้พีซีสามารถสื่อสารข้ามแลนได้ แพจเกจแลนอื่น ๆ หลายตัวก็ใช้อินเตอร์รัปต์ 14 เช่นกัน โปรแกรมเหล่านี้สามารถปรับตั้งเพื่อให้ซอฟต์แวร์อื่นใช้อินเตอร์รัปต์ 14 เพื่อเข้าถึงทรัพยากรของพวกมันได้ ข้อมูลในบทนี้จึงอาจมีประโยชน์แม้ว่าคุณไม่ได้ใช้ BIOS funotions เพื่อการสื่อสารข้อมูลแบบอนุกรมก็ตาม

ซอฟต์แวร์อินเตอร์รัปต์

ซอฟต์แวร์อินเตอร์รัปต์เป็นวิธีการสั่งให้โปรเซสเซอร์กระโดดไปยังตำแหน่งต่าง ๆ เพื่อไปทำงานตามคำสั่งที่อยู่ในตำแหน่งนั้น เมื่อโปรเซสเซอร์อ่านคำสั่งซอฟต์แวร์อินเตอร์รัปต์เข้ามา มันจะไปอ่านข้อมูลบนหน่วยความจำที่เรียกว่า ตารางอินเตอร์รัปต์เวกเตอร์ (interrupt veotor table) พื้นที่นี้บรรจุแอดเดรสของอินเตอร์รัปต์ที่เป็นไปได้ทั้งหมด แต่ละแอดเดรสมีขนาด 4 ไบต์ ดังนั้นการทำคำสั่งอินเตอร์รัปต์ 21H โปรเซสเซอร์จะดูดที่ออกเซต (4 x 21H) ในตารางอินเตอร์รัปต์เวกเตอร์ จากนั้นจึงกระโดดไปทำงานตามคำสั่งที่เก็บไว้ที่แอดเดรสนั้น จึงกระทั่งพบ IRET แล้วโปรสเซสเซอร์จึงทำงานคำสั่งต่อไปนี้ในโปรแกรมเดิม

ฟังก์ชันการสื่อสารแบบอนุกรมของ BIOS

การเข้าถึงฟังก์ชันของ BIOS ที่เกี่ยวข้องกับการสื่อสารแบบอนุกรม ต้องเรียกผ่านอินเตอร์รัปต์ 14H โดยใส่ค่า 0 ถึง 3 ลงในรีจิสเตอร์ AH เพื่อรุบุฟังก์ชันที่ต้องการและใส่หมายเลขพอร์ตลงในรีจิสเตอร์ DX พอร์ต หมายเลข 0 สำหรับ COM1,1 สำหรับ COM2,2 สำหรับ COM3 และ 3 สำหรับ COM4

ฟังก์ชันตั้งค่าพารามิเตอร์การสื่อสาร

ฟังก์ชัน 0 ใช้สำหรับการตั้งค่าพารามิเตอร์การสื่อสาร การเข้าถึงฟังก์ชันนี้ทำได้โดยทำให้ค่าในรีจิสเตอร์ AH เป็น 0, รีจิสเตอร์ DX เป็นหมายเลขพอร์ต, ใส่ไบต์ที่แทนพารามิเตอร์ในรีจิสเตอร์ AL และสั่งอินเตอร์รัปต์ 14 H บิต 0 และ 1 ของไบต์ที่เป็นพารามิเตอร์แทนค่าความยาวเวิร์ด สำหรับเวิร์ดขนาดแปดบิต ทั้งสองบิตต้องเป็น 1 สำหรับเวิร์ดขนาดเจ็ดบิต บิต 1 เป็น 1 และบิต 0 เป็น บิต 2 ระบุจำนวนของบิตจบ ค่า 0 แทนหนึ่งบิตจบ และค่า 1 แทนสองบิตจบ บิต 3 และ 4 ระบุพาริตี้ตามตารางที่ 5.1 บิต 5 ถึง 7 ระบุอัตราบอด ดังในนารางที่ 5.2

ตารางที่ 5.1 การตั้งพาริตี้ของ BIOS Interrupt 14H

บิต 4	บิต 3	พาริตี้
0 หรือ 1	0	None
0	1	Odd
1	1	Even

ตารางที่ 5.2 การตั้งอัตราบอดของ BIOS Interrupt 14H

บิต 7	บิต 6	บิต 5	อัตราบอด
0	0	0	110
0	0	1	150
0	1	0	300
0	1	1	600
1	0	0	1200
1	0	1	2400
1	1	0	4800
1	1	1	9600

ตารางที่ 5.3 เป็นตัวอย่างของพารามิเตอร์สำหรับ 1200 บอด แปดบิตข้อมูล หนึ่งบิตจบ ไม่มีพาริตี้

ตารางที่ 5.3 ตัวอย่างของไบต์ที่เป็นพารามิเตอร์

บิต	การเซต	ค่า	ความหมาย
7	1	128	
6	0		1200 บอด
5	0		
4	0		ไม่มี
3	0		พาริตี้
2	0	2	แปด
1	1	1	บิตข้อมูล
ค่าของไบต์	13	131	

เมื่อตั้งค่าพารามิเตอร์แล้ว ฟังก์ชันตั้งค่าพารามิเตอร์การสื่อสารจะส่งสถานะปัจจุบันขอพอร์ตมาให้ในรีจิสเตอร์ AX

ฟังก์ชันส่งตัวอักษร

ฟังก์ชันสำหรับส่งตัวอักษรคือ ฟังก์ชัน 1 การเข้าถึง ทำได้โดยเซต AH เป็น 1 และ DX เป็นหมายเลขพอร์ต ตัวอักษรที่จะส่งอยู่ในรีจิสเตอร์ AL แล้วสั่งอินเตอร์รัปต์ 14H สังเกตว่า ตัวอักษรจะไม่ถูกส่งจนกระทั่งสายแฮนด์เช็คกิ้งบาเข้าไปได้ แม้แต่ใน BIOS Funotions ระดับล่างก็ไม่สามารถยกเลิกฮาร์ดแวร์แฮนด์เช็คกิ้งได้

ในทางปฏิบัติของการเขียนโปรแกรมจะเรียกใช้ฟังก์ชันอ่านสถานะของพอร์ต (ฟังก์ชัน) และเรียกใช้ฟังก์ชันการส่งตัวอักษรก็ต่อเมื่อรู้ว่าเงื่อนไขถูกต้อง (สายแฮนด์เช็คกิ้งขาเข้าเป็นไฮ)

ค่าที่ส่งกลับในรีจิสเตอร์ AH จะแสดงความผิดพลาดที่เกิดขึ้น ถ้าบิต 7 ของ AH เป็น 0 แสดงว่าการส่งสำเร็จ ถ้าบิต 7 ของ AH เป็น 1 บิตที่เหลือจะแสดงความผิดพลาดที่เกิดขึ้นโดยใช้รหัสเหมือนกับที่อธิบายไว้สำหรับฟังก์ชันอ่านสถานะของพอร์ต

ฟังก์ชันรับตัวอักษร

ฟังก์ชัน 2 ใช้สำหรับรับตัวอักษรโดยการเซต AH เป็น 2, DX เป็นหมายเลขพอร์ต และสั่งอินเตอร์รัปต์ 14H BIOS จะรอจนกระทั่งได้รับตัวอักษรจากพอร์ตอนุกรม หรือหมดเวลาไทม์เอาต์ ตัวอักษรที่รับได้จะถูกใส่ไว้ใน AL และการเกิดความผิดพลาดพูกรายงานไว้ใน AH

ถ้าไม่เกิดความผิดพลาด AH จะเป็น 0 ถ้าไม่เป็น 0 บิต 0 ถึง 7 จะแสดงเงื่อนไขการเกิดข้อผิดพลาด ดังที่อธิบายไว้ในฟังก์ชัน 3 อย่างไรก็ตามถ้าบิต 7 ถูกเซตเพื่อแสดงไทม์เอาต์บิตที่เหลือจะไม่มีความหมาย

ในทางปฏิบัติตามปกติ จะไม่เรียกฟังก์ชันรับตัวอักษรซ้ำ ๆ กัน แต่เรียกให้ฟังก์ชันอ่านสถานะของพอร์ตเพื่อตรวจสอบก่อน และจะเรียกฟังก์ชัน 2 ก็ต่อเมื่อรู้ว่ามีตัวอักษรเข้ามารออยู่ วิธีนี้ทำให้ควบคุมได้มากกว่า เนื่องจากสามารถใช้เวลาไทม์เอาต์ของตัวเองได้ และสามารถทำอย่างอื่นในขณะรอรับข้อมูลได้ และยังหลีกเลี่ยงปัญหาที่เกิดจากความผิดพลาดใน BIOS ของไอบีเอ็มพีซีรุ่นแรก ๆ ซึ่งรายงานไทม์เอาต์เป็นข้อผิดพลาดทางพาริตี้

ฟังก์ชันอ่านสถานะของพอร์ต

ฟังก์ชัน 3 เข้าถึงได้โดยการเซต AH เป็น 3, DX เป็นหมายเลขพอร์ตและส่งอินเตอร์รัปต์ 14H ฟังก์ชันนี้ให้ข้อมูลหลายอย่างเกี่ยวกับสถานะปัจจุบันของพอร์ตอนุกรม และส่งสถานะนั้นกลับมาในรีจิสเตอร์ AX โดยแต่ละบิตจะมีความหมายดังในตารางที่ 6.4

Delta หมายความว่าสัญญาณที่เกี่ยวข้องเปลี่ยนแปลงไปหลังจากการอ่านสถานะของพอร์ตครั้งล่าสุด ตัวอย่างเช่น บิต 5 ของไบต์ที่ส่งกลับใน AL แสดงว่า สัญญาณ DSR เป็นไฮหรือโล บิต 1 แสดงว่าสถานะของสัญญาณ DSR ได้เปลี่ยนไปจากการอ่านครั้งล่าสุดหรือไม่ ข้อมูล Delta มักจะถูกใช้ในการเชื่อมต่อกับ I/O และเนื่องจากวิธีนี้ทำไม่ได้โดยผ่านทางดอสหรือ BIOS ดังนั้น บิต Delta จึงไม่มีประโยชน์ในการใช้งานจริง

ตารางที่ 5.4 รหัสสถานะของพอร์ตตามอินเตอร์รัปต์

บิตของรีจิสเตอร์ AH	ความหมายถ้าถูกเซต
7	เลยเวลาไทร์เอาต์
6	รีจิสเตอร์เลื่อนข้อมูล (Transmitter Shift register) ว่าง
5	รีจิสเตอร์พักข้อมูลส่ง (Transmitter Holding register) ว่าง
4	พบสัญญาณแบรก
3	ความผิดพลาดทางเฟรม
2	ความผิดพลาดทางพาริตี้
1	ความผิดพลาดโอเวอร์รัน
0	มีข้อมูลเข้า

ตารางที่ 5.4 (ต่อ) รหัสสถานะของพอร์ตตามอินเตอร์รัปต์ 14

บิตของรีจิสเตอร์ AL	ความหมายถ้าถูกเซต
7	พบ Receive line signal
6	มีสัญญาณกริ่ง
5	Data set ready
4	Clear to send
3	Delta receive line signal
2	ขอบขาลงของสัญญาณกริ่ง
1	Delta data set ready
0	Delta clear to rend

แฮนด์เช็คกิ้งภายใต้ BIOS

BIOS ทำงานแปลกประหลาดมากเกี่ยวกับสัญญาณแฮนด์เช็คกิ้ง ฟังก์ชันตั้งค่าพารามิเตอร์ ไม่มีการเซตสัญญาณแฮนด์เช็คกิ้งใด ๆ ฟังก์ชันรับตัวอักษรเปิดสัญญาณ DTR และปิดสัญญาณ RTS แล้วจึงรอรับตัวอักษรที่เข้ามา หรือส่งค่าแสดงข้อผิดพลาดถ้าไม่ได้รับตัวอักษรภายในช่วงเวลาที่กำหนด ฟังก์ชันส่งตัวอักษรเซต DTR และ RTS แล้วระให้ฟัง DSR และ CTS ถูกเซตโดยอุปกรณ์อีกตัวจึงจะส่งตัวอักษร ถ้าทั้งคู่ไม่ถูกเซตภายในเวลาไทม์เอาต์ก็จะออกจากฟังก์ชัน

เรื่องแปลกก็คือ ขณะที่ BIOS ต้องการสัญญาณแฮนด์เช็คกิ้งสองสัญญาณสำหรับการส่ง มันกลับให้สัญญาณแก่อุปกรณ์อื่นเพียงสัญญาณเดียวในการรับ ถ้าพีซีอีกเครื่องหรืออุปกรณ์อื่นที่ต้องการแฮนด์เช็คกิ้งสองสัญญาณจะต้องสร้างสัญญาณที่หายไปด้วยการเชื่อม RTS กับ DTR ที่ปลายสายทางด้านอุปกรณ์ที่อยู่ระยะไกล้ และปลดการเชื่อมต่อของ RTS ที่ปลายด้านพีซีก่อนการเรียกใช้ฟังก์ชันรับหรือส่งตัวอักษร สัญญาณแฮนด์เช็คกิ้งขาออกจะถูกปิด ถ้าอุปกรณ์อีกตัวต้องการสัญญาณแฮนด์เช็คกิ้ง ต้องมีการเรียกฟังก์ชันส่งหรือรับตัวอักษรก่อน หลังจากนั้น อย่างน้อยจะมี DTR ที่ยังคงเป็นไฮ

อย่างไรก็ตาม ถ้าทำตามวิธีการปกติในการเรียกฟังก์ชันอ่านสถานะของพอร์ต เพื่อดูว่ามีตัวอักษรที่รับได้หรือไม่ ก่อนจะเรียกใช้ฟังก์ชันรับตัวอักษร อาจทำให้เกิดปัญหาไม่ได้รับอะไรเลย เนื่องจากไม่มีการเรียกฟังก์ชันรับตัวอักษรจนกว่าจะมีตัวอักษรเข้ามา แต่ตัวอักษรจะไม่เข้าเนื่องจากสายแฮนด์เช็คกิ้งเป็นโล เพราะว่ายังไม่มีการเรียกฟังก์ชันรับตัวอักษร

ดังนั้นในขั้นตอนการเริ่มต้น หลังขากตั้งค่าพารามิเตอร์แล้ว ขอแนะนำให้เรียกฟังก์ชันการอ่านตัวอักษรหนึ่งครั้งเพื่อเปิด DTR และอย่าคิดว่าไม่จำเป็นเชื่อมสายเข้าด้วยกัน เพราะคิดว่าฟังก์ชันส่งตัวอักษรจะเซต RTR ให้ เนื่องจากว่าหลังจากใช้ฟังก์ชันรับตัวอักษร RTS จะถูกปิด

นอกจากนี้ยังไม่มีวิธีที่จะปิดสัญญาณแฮนด์เช็คกิ้งเมื่อเสร็จสิ้นการสื่อสารอีกด้วย ดังนั้นเชื่อถือการยกเลิกการเชื่อมต่อของโมเด็ม เมื่อออกจากโปรแกรม ควรตรวจสอบการวางสายด้วยตัวเองอีกครั้ง

ข้อเสียของการสื่อสารแบบอนุกรมด้วย BIOS

โชคไม่ดีที่การสื่อสารแบบอนุกรมด้วย BIOS มีข้อจำกัดมากกว่าการโปรแกรมในระดับระบบ ดังต่อไปนี้

I/O แบบกระตุ้นด้วยอินเตอร์รัปต์

ชิป UART ที่ใช้ในพีซีสามารถโปรแกรมให้สร้างสัญญาณอินเตอร์รัปต์ เมื่อมีเหตุการณ์ที่กำหนดเกิดขึ้น เช่น เมื่อมีตัวอักษรเข้า ข้อดีของวิธีการนี้และเทคนิคที่เกี่ยวข้องจะกล่าวอย่างละเอียดในบทต่อไป การใช้วิธีนี้ต้องโปรแกรมชิปโดยตรง ไม่สามารถใช้ฟังก์ชัน BIOS ได้ ซอฟต์แวร์สื่อสารเกือบทั้งหมดที่ทำงานด้วยอัตราบอดสูงจำเป็นต้องใช้วิธีนี้

แฮนด์เช็คกิ้ง

ฟังก์ชันทั้งหมดของดอสต้องการให้สายแฮนด์เช็คกิ้งขาเข้าทั้งคู่เป็นไฮ จึงจะส่งได้ ในบางกรณีเช่น แต่ตามปกติมักจะสร้างสัญญาณแฮนด์เช็คกิ้งหลอกขึ้นได้ด้วยการเชื่อมต่อสายแฮนด์เช็คกิ้งขาออกเพียงเส้นเดียว การโปรแกรม UART โดยตรง สามารถควบคุมหรือไม่สนใจแฮนด์เช็คกิ้งได้ตามต้องการ

เบรก

สัญญาณเบรกเป็นวิธีการขัดจังหวะโปรแกรมของเมนเฟรมบางตัว การส่งสัญญาณเบรกไม่สามารถทำได้โดยใช้ฟังก์ชันของดอสและ BIOS มันทำได้ด้วยการโปรแกรม UART โดยตรงเท่านั้น

ผู้ออกแบบ BIOS ให้ความสำคัญกับฟังก์ชันการสื่อสารของ BIOS ค่อนข้างน้อย บางทีอาจเป็นเพราะตั้งใจที่จะนำไปใช้กับเครื่องพิมพ์อนุกรมเป็นหลัก มากกว่าสื่อสารกับโมเด็มและอุปกรณ์อนุกรมอื่น การใช้ความสามารถทางการสื่อสารแบบอนุกรมของพีซีให้ได้เต็มที่จำเป็นต้องใช้การโปรแกรมในระดับที่ต่ำกว่านี้

บทที่ 6

การสื่อสาร อนุกรมโดยภาษา C

บทนี้จะอธิบายวิธีใช้ภาษาซีสำหรับเขียนโปรแกรมสื่อสารแบบอนุกรมสำหรับพีซี โดยใช้ทั้งสามวิธี คือ การใช้ BIOS (อิน - เตอร์รัปต์ 14 ป การใช้คำสั่ง I/O เพื่อโพล UART และการใช้ตัวจัดการอินเตอร์-รัปต์ ควรมีความรู้เกี่ยวกับการเขียนโปรแกรมด้วยภาษาซีมาก่อน

วิธีที่คอมไพเลอร์ (Compiler) แต่ละตัวใช้ในการเข้าถึงฟังก์ชันในระดับต่ำจะแตกต่างกัน ดังนั้น ภาษาซีในส่วนที่เกี่ยวข้องกับการสื่อสารอนุกรมต้องสัมพันธ์กับคอมไพเลอร์ตัวใดตัวหนึ่ง ในบทนี้อ้างอิงกับ Microsoft C เวอร์ชั่น 7 สำหรับพีซี และใช้กับการเขียนโปรแกรมภายใต้ดอส

เฮดเดอร์ไฟล์สำหรับการสื่อสารแบบอนุกรม

ผู้ที่เขียนโปรแกรมด้วยภาษีซีคงคุ้นเคยกับการใช้ประโยค # define เพื่อนิยามค่าคงที่ วิธีนี้ทำให้โปรแกรมภาษาซีอ่านง่ายขึ้น ประโยค # define ถูกใช้บ่อยในเฮดเดอร์ไฟล์ รูปที่ 6.1 เป็นเฮดเดอร์ไฟล์สำหรับการสื่อสารแบบอนุกรมที่เราใช้ในบทนี้

การคัดลอดไฟล์นี้ไปใช้และรวมมันเข้ากับโปรแกรมของตนเองจะช่วยให้เกิดความสะดวกมากขึ้น ฟังก์ชันตัวอย่างในบทนี้ใช้ค่าที่นิยามไว้ในเฮดเดอร์ไฟล์นี้เช่นกัน

รูปที่ 6.1 เฮดเดอร์ไฟล์ชื่อ sercom.h สำหรับการสื่อสารอนุกรม

```
// บิตต่าง ๆ ในรีจิสเตอร์ควบคุมสายสื่อสาร
# define    LCONT_ LSB                  1
# define    LCONT_ MSB                  2
# define    LCONT_ STOP                 4
# define    LCONT_ PARITY _ ENABLE      8
# define    LCONT_ PARITY _ SELECT      10
# define    LCONT_ BREAK                32
# define    LCONT_ DLAB                 128
// แฟล็กสำหรับแฮนด์เช็คกิ้งในรีจิสเตอร์ควบคุมโมเด็ม

# define    MCONT_ DRR                  1
# define    MCONT_ RTS                  2
# define    MCONT_ OUT1                 4
# define    MCONT_ OUT2                 8
```

รูปที่ 6.1 (ต่อ) เฮดเดอร์ไฟล์ชื่อ seroom.h สำหรับการสื่อสารอนุกรม

```
// บิตต่าง ๆ ในรีจิสเตอร์แสดงสถานะสายสื่อสาร

# define    LSTAT_DATA_READY          1
# define    LSTAT_OVERRUN             2
# define    LSTAT_PARITY_ERROR        4
# define    LSTAT_FRAME_ERROR         8
# define    LSTAT_BREAK_INT           16
# define    LSTAT_THRE                32
# define    LSTAT_TSRE                64

// บิตต่าง ๆ ในรีจิสเตอร์อีนาเบิลอินเตอร์รัปต์

# define    ENABLE_DATA_AVAILABLE     1
# define    ENABLE_THRE               2
# define    ENABLE_RCVLINESTAT        4
# define    ENABLE_MODEM_STAT         8
# define    ENABLE_BREAK              32
# define    ENABLE_ALL (1 | 2 | 4 | 8 )

// บิตต่างๆ ในรีจิสเตอร์แสดงสถานะโมเด็ม

# define    MSTAT_DELTA_CTS           1
# define    MSTAT_DELTA_DSR           2
# define    MSTAT_TERI                4
# define    MSTAT_DELTA_RSLD          8
# define    MSTAT_CTS                 16
# define    MSTAT_DSR                 32
# define    MSTAT_RI                  64
# define    MSTAT_RLSD                128

// ค่าพาริตี้

# define    PARITY_NONE               0
# define    PARITY_ODD                1
# define    PARITY_EVEN               2
```

รูปที่ 6.1 (ต่อ) เฮดเดอร์ไฟล์ชื่อ sercom.h สำหรับการสื่อสารอนุกรม

```
# define    PARITY_ERROR            1
# define    FRAME_ERROR             2
# define    BREAK_ERROR             3
exterm  int port_ads ;    // เบสแอดเดรสของ UART

// ตำแหน่งของรีจิสเตอร์ใน UART
# define    REG_RX          port_ads
# define    REG_TX          port_ads
# define    REG_INT_EN      port_ads + 1
# define    REG_INT_ID      port_ads + 2
# define    REG_LCONT       port_ads + 3
# define    REG_MCONT       port_ads + 4
# define    REG_STAT        port_ads + 5
# define    REG_MSTAT       port_ads + 6
# define    REG_SEG_SCR     port_ads + 7
# define    REG_FIFO        port_ads + 2
          // บัฟเฟอร์แบบวงกลม
# ifdef NOTHING
typedef     struct tag_ring   {
   int      count;      /*จำนวนตัวอักษรที่อยู่บัฟเฟอร์*/
   int      start:      /*ตำแหน่งตัวอักษรที่จะถูกดึงเป็นตัวถัดไป*/
   int      chext;      /*ตำแหน่งที่ตัวอักษรถัดไปจะถูกเก็บไว้*/
   int      size;       /*ขนาดของบัฟเฟอร์*/
   char *   buffer;     /*แอดเดรสของบัฟเฟอร์ตัวอักษร*/
} RING;
# endif

        // ค่าสมมูลสำหรับอินเตอร์รัปต์ 14

# define    COM1        0
# define    COM2        1
# define    COM3        2
# define    COM4        3

void comparm ( int parity, int stop , int databits) ;
void baudset ( long baudset ) ;
```

การจัดการบิตในภาษาซี

ไม่ว่าจะใช้ฟังก์ชันของดอสหรือ ROM BIOS หรือโปรแกรม UART โดยตรงก็ตาม เมื่อเขียนโปรแกรมสื่อสารมักจะต้องมีเข้าถึงแต่ละบิตภายในไบต์ เนื่องจากต้องเกี่ยวข้องกับการอ่านและการเซตรีจิสเตอร์ที่แต่ละบิตมีความหมายต่างกัน แม้ว่าหนังสือเล่มนี้จะไม่ได้เขียนมาสำหรับผู้ใช้ภาษีซีมือใหม่ แต่ก็จะกล่าวถึงการจัดการบิตอย่างกว้าง ๆ เพราะว่าหัวข้อนี้มักถูกละเลยในหนังสือภาษาซีมาตรฐาน และมันจะจำเป็นต่อการเขียนโปรแกรมสื่อสารแบบอนุกรม

ขั้นแรกจำเป็นต้องทำการคุ้นเคยกับค่าของไบต์ที่มีเพียงหนึ่งบิตถูกเซตค่านี้เรียกว่า ค่าบิต (Bit value) ซึ่งมีค่าสำหรับบิต 0 ถึง 7 ดังนี้

บิต	ค่าบิต
0	1
1	2
2	4
3	8
4	16
5	32
6	64
7	128

การทดสอบบิต

การสอบว่าบิตในบิตหนึ่งถูกเซตหรือไม่ ทำได้โดยการนำจำนวนนั้นมา AND กับค่าบิตโดยโดยใช้ตัวดำเนินการ & ดังนั้นถ้า status แทนสถานะของพอร์ต เมื่อต้องการทราบบิต 4 ถูกเซตหรือไม่ เขียนเป็นภาษาซีได้ดังนี้

```
if (status & 16 ) {
.....
}
```

ในตัวอย่างข้างต้น ถ้า status เป็น 40 ฐานสิบ ซีจะทำการคำนวณดังนี้

Status	0	0	1	0	1	0	0	0
รูปแบบที่ใช้ทดสอบ	0	0	0	1	0	0	0	0
ผลจากการ AND	0	0	0	0	0	0	0	0

ซึ่งให้ผลลัพธ์เป็นศูนย์ แสดงว่า บิต 4 ในไบต์แสดงสถานะไม่ถูกเซต ถ้า status เป็น 52 ฐานสิบ ซีจะทำการคำนวณดังนี้

Status	0	0	1	1	0	1	0	0
รูปแบบที่ใช้ทดสอบ	0	0	0	1	0	0	0	0
ผลจากการ AND	0	0	0	1	0	0	0	0

ผลลัพธ์ที่ไม่เป็นศูนย์ แสดงว่า บิต 4 ในไบต์แสดงสถานะถูกเซต

การทำให้บิตเป็นหนึ่ง

ดำเนินการ | ทำงานตามตรรก OR เมื่อ OR สองจำนวนเข้าด้วยกันจะเกิดจำนวนใหม่ที่มีบิตถูกเซต เมื่อมีบิตหนึ่งหรือทั้งสองบิตที่ตรงกันในจำนวนทั้งสองถูกเซต ตัวอย่างเช่น ถ้า comparm แทนพารามิเตอร์ไบต์และต้องการเซตบิต 3 เป็นหนึ่ง เขียนเป็นโปรแกรมได้ดังนี้

```
comparm |= 8;
```

ถ้า comparm มีค่า 37 การคำนวณจะเป็นดังนี้

```
comparm   0 0 1 0 0 1 0 1
8         0 0 0 0 1 0 0 0
OR        0 0 1 0 1 1 0 1
```

ให้ทำให้บิตเป็นศูนย์

การทำให้บิตเป็นศูนย์ทำได้โดยใช้ตัวดำเนินงาน - ซึ่งจะกลับค่าบิตทุกบิตเป็นค่าตรงข้าม เมื่อทำการ AND ไบต์หนึ่งกับค่าคอมพลีเมนต์ของค่าบิตที่ต้องการทำให้เป็นศูนย์จะให้ผลลัพธ์เป็นไบต์ที่มีบิตที่ต้องการเป็นศูนย์ เช่น ถ้าต้องการให้บิต 5 ของ parm เป็นศูนย์ สามารถใช้นิพจน์

```
parm &= -32;
```

ถ้า parm เป็น 106 ฐานสิบ ขั้นแรกซีจะคำนวนไบต์ที่เป็น 1-comple - ment ของ 106 ดังนี้

```
30    0 0 1 0 0 0 0 0
-32   1 1 0 1 1 1 1 1
```

จากนั้นซีจะ AND ไบต์นี้กับ parm ดังนี้

```
Parm      0 1 1 0 1 0 1 0
-32       1 1 0 1 1 1 1 1
AND       0 1 0 0 1 0 1 0
```

บิต 5 จะถูกเซตเป็นศูนย์ตามต้องการ

วิธีการเขียนเพื่อจัดการบิต

ตามปกติเรามักจะใช้ประโยค # define เพื่อบอกค่าบิตภายในไบต์ตัวอย่างเช่น กำหนดให้ Isatus เป็นไบต์แสดงสถานะสายสื่อสาร เราสามารถใช้ประโยค # define ในไฟล์ seroom.h เพื่อตรวจสอบแต่ละบิตในรีจิสเตอร์แสดงสถานะสายสื่อสารได้ดังนี้

```
if (Istatus & LSTAT_DATA_READY) { // data ready }
if (Istatus & LSTAT_PARITY_ERROR) { // parity ready }
if (Istatus & LSTAT_BREAK_INT) { // break interrupt }
```

การเรียกฟังก์ชันของ BIOS จากภาษาซี

ไลบรารีของ MSC มีฟังก์ชันในการจัดการอินเตอร์รัปต์ 14 คือ ฟังก์ชัน _bios_serialcom() ซึ่งมีการประกาศไว้ดังนี้

unsigned _bios_serialcom (unsigned service, unsigned serial_port, unsigned data)

พารามิเตอร์ service เป็นค่าใดค่าหนึ่งต่อไปนี้ (นิยามไว้ใน bios.h ซึ่งต้องถูกรวมเข้ากับโปรแกรม)

_COM_INIT	เซตพารามิเตอร์ของพอร์ต (อัตราบอด, จำนวนบิตข้อมูล เป็นต้น)
_COM_SEND	ส่งตัวอักษรหนึ่งตัว
_COM_RECEIVE	รับตัวอักษรหนึ่งตัว
_COM_STATUS	รายงานสถานะของพอร์ต

พารามิเตอร์ port เป็น 0 สำหรับ COM1, 1 สำหรับ COM2 เป็นต้น

พารามิเตอร์ data แทนอัตราบอดและค่าอื่น ๆ (ถ้า service เป็น _COM_INIT) หรือเป็นตัวอักษณที่จะส่ง (ถ้า service เป็น _COM_SEND)

พารามิเตอร์การสื่อสาร เลือกได้จากค่าที่ถูกนิยามไว้ใน bios.h ดังต่อไปนี้

_COM_CHR7	7 บิตจับข้อมูล
_COM_CHR8	8 บิตจับข้อมูล
_COM_STOP1	1 บิตจบ
_COM_STOP2	2 บิตจบ
_COM_NOPARITY	ไม่มีพาริตี้
_COM_EVENPARITY	พาริตี้คู่
_COM_ODDPARITY	พาริตี้คี่
_COM_110	110 บอด
_COM_150	150 บอด
_COM_300	300 บอด
_COM_600	600 บอด
_COM_1200	1200 บอด
_COM_2400	2400 บอด
_COM_4800	4800 บอด
_COM_9600	9600 บอด

ค่าเหล่านี้สามารถนำมา OR กันเพื่อสร้างเป็นพารามิเตอร์ เช่น การเซต COM1 ให้เป็นแปดบิตข้อมูล หนึ่งบิตจบ ไม่มีพาริตี้ และ 2400 บอดเขียนได้ดังนี้

```
_bios_serialcom (_COM_INIT,0, _COM_CHR8
| _COM_STOP 1 | _COM_NOPARITY | _COM_2400);
```

ค่าที่คืนกลับมาจะเป็นตัวอักษรที่รับได้ (ในกรณีของ _COM_RECEIVE) หรือสถานะ (ในกรณีของ _COM_STATUS) สถานะเป็นค่า 16 บิต ที่ประกอบด้วยค่าของรีจิสเตอร์แสดงสถานะสายสื่อสารในไบต์สูงและรีจิสเตอร์แสดงสถานะโมเด็มในไบต์ต่ำ

การใช้แอสเซมเบลอร์แบบอินไลน์

สำหรับผู้ที่คุ้นเคยกับการเขียนโปรแกรมภาษาแอสเซมบลีก็สามารถใช้คุณสมบัติของแอสเซมเบลอร์แบบอินไลน์ (inline assembler) ของ MSC เพื่อเข้าถึง BIOS ได้ ตัวอย่างเช่น การส่งตัวอักษร chr ผ่าน COM2 สามารถเขียนเป็นฟังก์ชัน ได้ดังนี้

```
void send14 ( unsigned char chr )
_asm {
    mov  ah, 1        // ฟังก์ชันสำหรับส่ง
    mov  al, cher     // ตัวอักษรที่จะส่ง
    mov dx, 1         // 1 = COM2
    int  14  H
    }
```

ตัวอย่างโปรแกรมที่ใช้ BIOS

โปรแกรมในรูปที่ 16.2 ให้ฟังก์ชันของ BIOS เพื่อดาวน์โหลดไฟล์ที่ได้รับจากคอมพิวเตอร์อีกเครื่องหนึ่ง มันแสดงให้เห็นลักษณะร่วมกันหลายอย่างของการเขียนโปรแกรมสื่อสารแบบอนุกรม รวมทั้งแนวความคิดของลูปที่มีการทดสอบมากกว่าหนึ่งสถานะ (ในกรณีนี้คือ การกดแป้นพิมพ์และการรับตัวอักษร) และการประมวลผลที่ตามมา กล่าวอีกนัยหนึ่งคือมันแสดงให้เห็นวิธีการโพลลิ่ง โปรแกรมนี้แสดงวิธีการเซตพารามิเตอร์จากภายในโปรแกรมโดยใช้ฟังก์ชันของ ROM BIOS ด้วยเช่นกัน

รูปที่ 6.2 โปรแกรมดาวน์โหลดไฟล์โดยใช้ BIOS

```
# define  < stdio.h >
# define  < bios.h >
# define  PORTNO 1    // สำหรับ COM2, COM1 ค่านี้จะเป็น 0
main ( )
{
FILE *f ;
int result, status ;
```

รูปที่ 6.2 (ต่อ) โปรแกรมดาวน์โหลดไฟล์โดยใช้ BIOS

```
f = fopan ( CAPTURE", "W" ) ;
if ( f == NULL ) {
  printf ( Could not open capture file \ n" ) ;
  exit ( 1 ) ;
}
_bios_serialcom (_COM_INIT , PORTNO ,
_COM_1200 | _COM_STOP1 | _COM_NOPARITY  | _COM_CHR8);

printf ('Waiting for input \n" );
for  ( ;; ) {
  if ( kbhif ( ) ) {
    if ( getch ( ) == 27 )
      break  ;
  }
  status = _bios_serialcom ( _COM_STATUS , PORT_NO, 0);

  if ( ! ( status & 266 ))    // ตรวจสอบบิตศูนย์ในไบต์สูง
      continue;

  chr = ( unsigned char ) _bios_serialcom
      ( _ COM_RECEIVE, PORTNO. 0) ;

  if ( chr === 26 )          // สิ้นสุดเมื่อได้รับ  Ctrl-z
    break
  putchar ( chr ) ;          // แสดงตัวอักษร
  fputc ( chr, f ) ;         //เขียนตัวอักษรลงสู่ไฟล์
}
fclose ( f ) ;
exit ( 0 ) ;
}
```

โปรแกรม UART โดยตรงแบบโพลลิ่ง

ถ้าไม่ต้องการใช้ BIOS ก็สามารถควบคุม UART ได้โดยตรง ซึ่งสามารถใช้ได้ทั้งวิธีโพลลิ่ง (polling) หรือใช้วิธีกระตุ้นด้วยอินเตอร์รัปต์ (Interrupt driven) วิธีกระตุ้นด้วยอินเตอร์รัปต์จะอธิบายในหัวข้อต่อไป

การอ่านและเขียนรีจิสเตอร์ของ UART

การที่จะสื่อสารโดยตรงกับ UART จะต้องสามารถอ่านและเขียนรีจิสเตอร์ของมันได้ รีจิสเตอร์แต่ละตัวจะมีแอดเดรส I/O ของตัวเอง การเข้าถึงแอดเดรส I/O บนพีซี ใช้คำสั่ง IN และ OUT ใน MSC มีฟังก์ชันสามฟังก์ชันที่ทำงานนี้คือ

int_inp (port)	อ่าน 1 ไบต์ จาก (unsigned int) port
int_outp (port, value)	ส่ง (int) value เป็นไบต์เดี่ยวไปยัง (unsigned int) port และคืนค่า value
int_putpw (port, value)	ส่ง (unsigned int) value เป็นค่า 2 ไบต์ ไปยัง (unsigned int) port และคืนค่า value

ถ้ากลับไปดูตารางที่ 5.3 ซึ่งแสดงแอดเดรส I/O ของรีจิสเตอร์สำหรับ UART ที่ COM1 และ COM2 จะเห็นว่ารีจิสเตอร์แสดงสถานะสายสื่อสารสำหรับ COM1 อยู่ที่ 3FDH ซึ่งอยู่สูงกว่าเบสแอดเดรสอยู่ห้าตำแหน่งในรูปที่ 6.1 REG_LSTAT ถูกนิยามให้เท่ากับ port_ads+5 และ post_ads นิยามไว้เป็นตัวแปรกลาง (global variale) สมมติว่าได้มีการเซตค่า post_ads เป็นเบสแอดเดรสของพอร์ตที่ต้องการเข้าถึง (เช่น ox3f8 สำหรับ COM1 หรือ ox2f8 สำหรับ COM2) การอ่านรีจิสเตอร์แสดงสถานะสายสื่อสารสามารถเขียนได้ดังนี้

```
int lstatus;
lstatus  =  _inp (REG_LSTAT ) ;
```

รีจิสเตอร์แสดงสถานะโมเด็มอยู่สูงกว่าเบสแอดเดรสสี่ตำแหน่ง ตามที่นิยามไว้ในรูปที่ 6.1 การส่งค่า modbyte ไปยังรีจิสเตอร์ใช้

```
_out (REG_MCONT, modbyte ) ;
```

การตั้งอัตราบอด

ตัวอย่างในรูปที่ 6.3 แสดงการตั้งอัตราบอดโดยการโปรแกรม 8250 โดยตรง

รูปที่ 6.3 การตั้งอัตราบอด

```
# include "seroom.h"
/*-------------------------------------------------------------------*/
void baudset ( long baudrate )
{
unsigned int divisor;
unsigned  char lsb, msb;

   divisor  =  1152001  /  baudrate ;
   msb  =  divisor  >> 8;
   lsb  =  (divisor << 8)>> 8;          // ให้ผลเหมือน &oxFF
```

รูปที่ 6.3 (ต่อ) การตั้งอัตราบอด

```
        _out ( REG_LCONT, LCONT_DLAB) :  // ทำให้สามารถเข้าถึง
                                         //แลตช์ตัวหาร
                                         // ด้วยการตั้งค่าบิตเข้าถึง
                                         //แลตช์ตัวหาร
                                         //ในรีจิสเตอร์ควบคุมสาย
                                         //สื่อสาร
        _outp (port_ads, lsb );          //ไบต์นัยสำคัญต่ำสุดของ
                                         //ตัวหาร
        _outp (port_ads + 1, msb );      //ไบต์นัยสำคัญสูงสุดของ
                                         //ตัวหาร
    }
    /*-------------------------------------------------------------------*/
```

เริ่มจากกาคำนวณตัวหารอัตราบอด คือ 115,200 หารด้วยอัตราบอดจากนั้นแยกตัวหารออกเป็นไบต์นัยสำคัญสูงสุดและไบต์นัยสำคัญต่ำสุดเราต้องการเขียนค่าเหล่านี้ลงใน UART แต่ก่อนอื่นต้องอีนาเบิลการเข้าถึงแลตช์ตัวหาร โดยการเซตบิต 7 ในรีจิสเตอร์ควบคุมสายสื่อสาร หลังจากนั้นค่าอีกสองไบต์ที่ถูกส่งให้แอดเดรส I/O ตำแหน่งแรกและตำแหน่งที่สองจะเป็นตัวกำหนดอัตราบอด

การตั้งความยาวเวิร์ด บิตจบ และพาริตี้

รูปที่ 6.4 แสดงวิธีการตั้งความยาวเวิร์ด จำนวนของบิตจบ และพาริตี้

```
# include seroom.h "
/*-------------------------------------------------------*/
void compam (int parity , int stop , int databits )
{
BYTE parmbyte ;
   parmbyte  =  databits - 5 ;  // ค่าเริ่มต้น : เซตบิต 1 และ 2
   if ( stop  == 2 )
     parmbyte  | = LCONT_STOP;
   if ( parity ! = PARITY_NONE )
     parmbyte  | = LCONT_PARITY_ENABLE;
   if ( parity ! = PARITY_EVEN )
     parmbyte  | = LCONT_PARITY_SELECT;
   _outp (REG_LCONT, parmbyte );
}
/*-------------------------------------------------------*/
```

ตัวอย่างโปรแกรมที่ใช้การควบคุม UART โดยตรง

โปรแกรมในรูปที่ 6.5 ใช้การควบคุม UART โดยตรง ผ่านทางคำสั่ง IN และ OUT

รูปที่ 6.5 การดาวน์โหลดไฟล์โดยการโปรแกรม UART

```
# include       < stdio.h >
# include       < bios.h >
# include       seroom.h "

int  port_ads;
/*-------------------------------------------------------------*/
main ( )
{
FILE * f ;
unsigned char chr ;
int result, status;
unsigned int lstat , mstat ;

   f = fopen ( "CAPTURE" , "W" ) ;
   if   ( f == NULL ) {
        printf ( " Could not open capture file\n\" ) ;
        exit ( 1 ) ;
}
port_ads = 0x2f8                        // สำหรับ COM2, COM1 ใช้
                                        // 3f8
baudset ( ( long ) ) 2400 ) ;           // ตั้งค่าอัตราบอดเป็น 2400
comparm ( PARITY_NONE, 1, 8 ); // ไม่มีพาริตี้ , 1 บิตจบ , 8 บิต
                                        // ข้อมูล
_outp ( REG_MCONT , MCONT | MCONT_RTS ;  // เปิด
                                        // สัญญาณแฮนด์เช็คกิ้ง
printf ("Waiting for input\n" ) ;
for ( ;; ) {
   if ( kbnit ( ) ) {
        if ( getch ( ) == 27 )
            break ;

   }
   lstat = _inp ( REG_LSTAT ) ;
   if (  lstat & LSTAT_OVERRUN )
        printf ( "Over-run error\n" ) ;
   if (  lstat & LSTAT_PARITY_ERROR )
        printf ( "Parity error\n"  ) ;
   if (  lstat & LSTAT_FRAME _ERROR )
        printf ( "Framing error\n" ) ;
   if (  lstat & LSTAT_BREAK_INT )
        printf ( "Break received" ) ;
```

รูปที่ 6.5 (ต่อ) การดาวน์โหลดไฟล์โดยการโปรแกรม UART

```
// ถ้าต้องการทดสอบสถานะโมเด็ม ก็สามารถทำได้ ณ จุดนี้
   if ( ! ( lstat & LSTAT_DATA_READY ) )
       continue ;
   if ( ohr == 26 )          // สิ้นสุดเมื่อได้รับ Ctrl - z
       break ;
   putchar ( ohr ) ;         // แสดงตัวอักษร
   fputc ( ohr ) ;           // เขียนตัวอักษรลงสู่ไฟล์
}
_outp (REG_MCONT , 0 ) ;  // ปิดสัญญาณแฮนด์เช็คกิ้ง
fclose ( f ) ;
exit ( 0 ) ;
}
/*-------------------------------------------------------------*/
```

I/O แบบกระตุ้นด้วยอินเตอร์รัปต์

การรับและส่งข้อมูลโดยใช้อินเตอร์รัปต์จำเป็นต้องสร้าง ISR (interrupt service routine) ขึ้นมาตัวหนึ่ง ISR นี้ถูกเรียกเมื่อ UART สร้างอินเตอร์รัปต์เพื่อแจ้งว่าเกิดเหตุการณ์เกี่ยวกับการสื่อสารอนุกรม ตามปกติจะเป็นการรับหรือการส่งตัวอักษร ในกรณีการรับตัวอักษร ISR จะใส่ตัวอักษรลงในบัฟเฟอร์เพื่อให้โปรกแกรมหลักดึงไปใช้ ในกรณีการส่งตัวอักษร ISR จะส่งตัวอักษรตัวถัดไปที่ค้างอยู่ในบัฟเฟอร์ข้อมูลออกตัวถัดไป

บัฟเฟอร์แบบวงกลม

การเขียนโปรแกรมแบบกระตุ้นด้วยอินเตอร์รัปต์ ต้องใช้บัฟเฟอร์แบบ FIFO ซึ่งตัวอักษรที่รับมาได้จะถูกนำมาเก็บไว้เพื่อรอให้โปรแกรมหลักดึงออกไปประมวลผล และบัฟเฟอร์แบบเดียวกันสำหรับเก็บตัวอักษรที่รอการส่งวิธีทั่วไปในการสร้างบัฟเฟอร์แบบ FIFO คือใช้บัฟเฟอร์แบบวงกลมหรือวงแหวน ฟังก์ชันสำหรับการสร้างและการใช้งานบัฟเฟอร์แบบวงกลมถูกแสดงไว้ในรูปที่ 6.6

รูปที่ 6.6 ฟังก์ชันสำหรับสร้างบัฟเฟอร์แบบวงกลม

```
void initbuf ( RING *thering , char * addr , int len )
{
// กำหนดค่าเริ่มต้น
    thering -> count  =  0;
    thering -> start  =  0;
    thering -> cnext  =  0;
    thering -> buffer =  addr;
```

รูปที่ 6.6 (ต่อ)ฟังก์ชันสำหรับสร้างบัฟเฟอร์แบบวงกลม

```
    thering -> size    =   len ;
    }
    /*-------------------------------------------------------------*/
    void putbuf ( RINT * theing , char ch )
    {
    // ใส่ตัวอักษรหนึ่งตัวลงในบัฟเฟอร์แบบวงกลม
thering ->buffer [ thering ->  cnext++]  = ch ;
if (thering -> count++ > = thering - > size ) { // โอเวอร์โฟลว์
                                                // หรือไม่
    thering -> countfi ;
    if (thering -> start++>= thering -> size )
    therint -> start -= thering ->  size ;       // เลื่อนจุดเริ่มต้น
}
// ต้องเลื่อนไปทับจุดเริ่มต้นของบัฟเฟอร์หรือไม่ ?
    if ( thering -> cnext >= thering-> size )
      thering ->cnext  -= thering-> size )
}
/*-------------------------------------------------------------*/
BOOLEAN getbuf ( RING / thering , char *ch )
{
// ดึงข้อมูลออกจากบัฟเฟอร์
    if (thering ->count <= 0 )
      return ( FALSE ) ;                    //ไม่มีข้อมูล
    *ch = ( char ) thering ->buffer [thering-> start++];
    thering -= count -- ;                   //ปรับจำนวนของ
                                            //ตัวอักษรที่เหลือ
    if (thering -> start > = thering -> size )
      thering -> start -= thering ->size;
    return ( TRUE ) ;
}
/*-------------------------------------------------------------*/
```

บัฟเฟอร์ประกอบด้วยพื้นที่ส่วนหนึ่งของหน่วยความจำที่ต่อเนื่องกัน ตัวอักษรที่ติดต่อกันถูกใส่ในตำแหน่งติดต่อกันบัฟเฟอร์จนกระทั่งถึงปลายบัฟเฟอร์ แล้วจึงวนกลับมาที่จุดเริ่มต้นอีกครั้ง โปรแกรมนี้ดูแลตัวชี้ที่บอกตำแหน่งของตัวอักษรที่จะเขียน หรือตัวอักษรที่จะอ่านเป็นตัวถัดไป และบันทึกจำนวนของตัวอักษรในบัฟเฟอร์ขณะนั้น

รูปที่ 6.7 แสดงฟังก์ชันสำหรับสร้างบัฟเฟอร์สองชุด และคืนหน่วยความจำเมื่อใช้งานเสร็จ บัฟเฟอร์ถูกสร้างโดยสร้างโดยเรียกฟังก์ชันพร้อมกับค่า TRUE และขนาดของบัฟเฟอร์ การทำลายบัฟเฟอร์ที่ใช้ฟังก์ชันพร้อมกับ FALSE

รูปที่ 6.7 ฟังก์ชันสำหรับเริ่มสร้างบัฟเฟอร์สองชุด

```
static BOOLEAN com_buffers (BOOLEAN create, int
 inbufsixe, int outbufsize )
{
static  char  *out_buf ;
static  char  *in_buf ;
static  BOOLEAN combuf_made =  FALSE ;

   if  (create ) {
      if ( combuf_made )
          return ( TRUE );
      out_buf = malloc ( outbufsize ) ;
      if ( out_bof  ==  NULL )
          return ( FALSE);
      in_buf + malloc ( inbufsize ) ;
      if ( in_buf == NULL ) {
          free ( out_buf );
          return ( out_buf ) ;
          }
       initbuf ( &outring, out_bof , outbufsize ) ;
       initbuf ( &inring, in_bof , inbufsize ) ;
       combuf_made = TRUE ;
       return (TRUE ) ;
   }
   else {                    // ทำลายบัฟเฟอร์
      if (combuf_made ) {
          free ( in_buf ) ;
          free ( out_buf );
       combuf_made = FALSE ;
     }
     return ( TRUE ) ;
  }
  return ( TRUE ) ;
 }
 /*-------------------------------------------------------------------*/
```

การเริ่มต้นโปรแกรมบริการอินเตอร์รัปต์

เมื่อสร้างบัฟเฟอร์แล้วขั้นตอนต่อมาคือ การเริ่มต้นส่วนของโปรแกรมบริการอินเตอร์รัปต์ ซึ่งมีขั้นตอนดังนี้

การบันทึกแอดเดรสของโปรแกรมเดิม (ถ้ามี) สำหรับอินเตอร์รัปต์นั้น

การเก็บแอดเดรสโปรแกรมของเราเองไว้ในตารางอินเตอร์รัปต์เวกเตอร์

การอีนาเบิลอินเตอร์รัปต์บน UART

การอีนาเบิล IRQ บน PIC

การตั้งค่าพารามิเตอร์ เช่น อัตราบอดและพาริตี้

การตั้งค่าสายสัญญาณฮาร์ดแวร์แฮนด์เช็คกิ้ง

รูปที่ 16.8 แสดงฟังก์ชัน intinit ( ) ซึ่งทำงานทั้งหมดนี้ ฟังก์ชัน intinit ( ) เรียกใช้ฟังก์ชัน fifo_init ( ) เพื่อตรวจสอบว่ามีบัฟเฟอร์ FIFO บน UART หรือไม่ ถ้ามีก็ทำการตั้งค่าเริ่มต้นของบัฟเฟอร์ตามรูปที่ 6.9

โปรแกรมบริการอินเตอร์รัปต์

ฟังก์ชันในรูปที่ 6.10 intinit ( ) เป็นส่วน ISR ที่แท้จริงซึ่งถูกเรียกเมื่อมีอินเตอร์รัปต์อนุกรมเกิดขึ้น

ฟังก์ชันเริ่มต้นด้วยคำสั่ง STI เพื่ออีนาเบิลอินเตอร์รัปต์ ต่อจากนั้นก็เรียกใช้ฟังก์ชัน check_init ( ) เพื่อดูว่าเหตุการณ์ใดที่กระตุ้นให้เกิดอินเตอร์รัปต์และจัดการตามเหตุการณ์นั้น ถ้าไม่มีเหตุการณ์ใดอินเตอร์รัปต์อาจเกิดจาก UART ตัวอื่นที่ใช้ IRQ เดียวกัน ดังนั้นฟังก์ชันจะไปเรียกตัวจัดการอินเตอร์รัปต์ที่ติดตั้งไว้ก่อนโปรแกรมของเรา จากนั้นจะทำงานเป็นลูป เพื่อจัดการกับเหตุการณ์อื่น จนกระทั่งไม่มีเหตุการณ์ที่ทำให้เกิดอินเตอร์รัปต์เหลืออยู่ สุดท้ายมันจะรีเซต PIC เพื่อให้ PIC รู้ว่าอินเตอร์รัปต์นี้ถูกให้บริการไปแล้วไปแล้วและสามารถรับอินเตอร์รัปต์อื่นได้

รูปที่ 6.8 ฟังก์ชันสำหรับเริ่มต้นโปรแกรมย่อยบริการอินเตอร์รัปต์

```
# inclue  "seroom.h"

static  int port_ads ;
static  BYTE inchr ;
static  BYTE piomask ;
static  int comerror , picaddr ;
static  BYTE lstat, mstat ;
static  BOOLEAN fifo_exist = FALSE ;
static  char max_send = 1;
static RING outring , inring ;
static BOOLEAN inxoff FALSE ;
void  (*oldhand ) ( ) ;
void _interrupt _far inthand ( );
```

รูปที่ 6.8 ฟังก์ชันสำหรับเริ่มต้นโปรแกรมย่อยบริการอินเตอร์รัปต์

```
BOOLEAN was _enabled = FALSE ;
int intinit ( int irqno )
{
BYTE oldva1 ;
   irq_nbr  =  irqno ;          // ใช้เป็นตัวแปรร่วม เพื่อให้ ISR สามารถ
                                // อ่านได้
   oldhand =  _dos_getveot (irq_nbr + 8 ) ;
   _dos_setveot (irq_nbr+8 , inthand ) ;
// เซต MCONT_DTR, MCONT_RTS, CCONT_OUT1,MCONT_OUT2
//บนรีจิสเตอร์ควบคุมโมเด็ม
   _outp (REG_MCONT, MCONT_DTR | MCONT_RTS |
   MCONT_OUT1 | MCONT_OUT2 );

//คำนวนหาบิตที่แทน irq นั้นในรีจิสเตอร์อีนาเบิลอินเตอร์รัปต์ของ PIC
   piomask  =  1 < < ( ird_nbr % 8 ) ;
   pioaddr   =  ( irq_nbr > 7 ) ? Oxa1 : Ox21 ;
_asm cli
   oldval    = _inp ( O x 21 ) ; // อ่านค่ารีจิสเตอร์อีนาเบิลอินเตอร์รัปต์
                                 // ของ PIC
_asm sti
   was_enabled  =  ! ( oldval  &  piomask ) ;
   _oulp ( REG_INT_EN,  O ) ;  // ดัสเอเบิลอินเตอร์รัปต์บน UART
fifo_init ( ) ;                // เริ่มการทำงานของบัฟเฟอร์แบบ FIFO
                               // ถ้ามี
// อ่านค่ารีจิสเตอร์ของ UART เพื่อล้างค่า
   mstat     =  _inp ( REG_MSTAT ) :
   lstat      =  _inp ( REG_LSTAT ) :
 ( void ) _inp ( REG_RX ) ;        //อ่านตัวอักษรที่ค้างอยู่
 ( void ) _inp ( REG_INT_ID );     //ล้างรีจิสเตอร์กำหนดอินเตอร์รัปต์
_outp (REG_INT,ENABLE_ALL) ;  // อีนาเบิลอินเตอร์รัปต์บน UART
_outp ( 0x20 , 0x20 ) ;                  //รีเซต PIC
   insoff  =  FALSE ;
   rt\eturn ( 1 ) ;
}
/*-------------------------------------------------------------*/
```

รูปที่ 6.9 ฟังก์ชันสำหรับเริ่มการทำงานของ บัฟเฟอร์ FIFO

```
void fifo_init ( )
{
//ตั้งค่าโดยปริยาย
   fifo_exist  =  FALSE ;
   max_send  =  1 ;
// ขั้นแรก ดูว่าเป็น  8250  หรือไม่
   _outp ( REG_SCR, 0x55 ) ;
   if ( _inp *( REG_SCR ) ! = 0x55 )     // ถ้าไม่มีรีจิสเตอร์แครตชี
     return ;                            // แสดงว่าเป็น  8250
// ดูว่า UART มีบัฟเฟอร์ FIFO หรือไม่ ถ้ามีให้อีนาเบิลมัน
     _outp (REG_FIFO, OxOf ) ;
     if  ( _inp REG_INT_ID ) )
        return ;
   fifo_exist  =  TRUE ;
   max_send  =  15 ;
  _outp ( REG_FIFO, 1 | 2 | 4 | 64 | 128  ) ;  // 14 ไบต์เป็นระดับตัวทริก
                                                // ในการรับ
}
/*-------------------------------------------------------*/
```

รูปที่ 6.10 โปรแกรมย่อยบริการอินเตอร์รัปต์อนุกรม

```
static BOOLEAN check_int ( ) ;
/*-------------------------------------------------------------------------*/
void_interrupl _far inthand ( )
{
_asm sti

   if  ( check _int ( ) == FALSE ) {   //ไม่มีอินเตอร์รัปต์ที่ระบุ
       if ( was_enabled )              //IRQ ใช้ร่วมกับตัวจัดการอื่น
          _chain_intr (ol dhand ) ;
   }
   else                               // ทำต่อไปจนกระทั่งไม่มีเหตุการณ์
       while  ( check_int ( ) ) ;     // ค้างอยู่
   _outp ( PIC1, EOI ) ;              // ส่ง end of interrupt ไปยัง
                                      // PIC ตัวแรก
   if ( irw_nbr > 7 )
```

รูปที่ 6.10 (ต่อ) โปรแกรมย่อยบริการอินเตอร์รัปต์อนุกรม

```
        _outp ( PIC2 , EOI ) ;      // PIC ตัวที่สอง
    }
    else                            // ทำต่อไปจนกระทั่งไม่มีเหตุการณ์
        while ( check_int ( ) ) //ค้างอยู่
      _outp ( PIC1, EOL ) ; //ส่ง end of interrupt ไปยัง
                                    // PIC ตัวแรก
      if ( irw_nbr > 7 )
        _outp ( PIC2, EOL ) ;    // PIC ตัวที่สอง
    }
    /*-------------------------------------------------------------*/
    static BOOLEAN check _int ( )
    }
    BYTE  intreg ;
    BYTE  chr ;
    int i ;
       intreg  =  _inp ( REG_INT_ID );
       if ( intreg & 1 )            // มีอินเตอร์รัปต์ค้างอยู่เหรือไม่
          return ( FALSE ) ;
       switch  ( intreg ) (
       case  0 :                    // สถานะโมเด็มเปลี่ยนแปลง
          mstat  =  _inp (REG_MSTAT ) ;
          return ( TRUE ) ;
       cade  2 :                    //รีจิสเตอร์พักข้อมูลส่งว่าง
          lstat  =  _inp ( REG_MSTAT ) ;
          for ( I = 0 ; ( i < max_ send ) && outring.count ;
              ++ I ) {
             ( void ) getbuf ( &outring, &inchr ) ;
             _outp ( REG_TX, inchr ) ;
       }
       return ( TRUE ) ;

          case  4  :                // รับข้อมูล
          while  (_inp (REG_LSTAT)  & LSTAT_DATA_READY ) {
             chr  =  _inp ( REG_RX ) ;
             putbuf ( &inring, chr );
          }
          return ( TRUE ) ;
```

รูปที่ 6.10 (ต่อ) โปรแกรมย่อยบริการอินเตอร์รัปต์อนุกรม

```
        case 6:                    // สถานะสายสื่อสารเปลี่ยนแปลง
            lstat = _inp ( REG_LSTAT ) ;
            if ( lstat & LSTAT_PATITY_ERROR )
               comerror = PARITY_ERROR;
            else if ( lstat & LSTAT_FRAME_ERROR )
               comerror = BREAK_ERROR;
            return ( TRUE ) ;
        }
        return ( FALSE ) ;              // เกิดความผิดพลาดจริง ๆ
    }
    /*-------------------------------------------------------------------*/
```

การรวมทั้งหมดเข้าด้วยกัน

รูปที่ 6.11 เป็นโปรแกรมที่เรียกฟังก์ชันที่เราพูดถึงมาแล้วทั้งหมดเริ่มจากการเซตค่าเริ่มต้นของการสื่อสาร แล้วจึงเข้าสู่ลูปที่จัดการทั้งแป้นพิมพ์และอินพุตจากพอร์ตอนุกรม เมื่อผู้ใช้กดปุ่ม Esc จะทำให้ออกโปรแกรม โปรแกรมนี้ทำงานได้และสามารถใช้เป็นพื้นฐานสำหรับโปรแกรมที่ซับซ้อนยิ่งขึ้นไป

รูปที่ 6.11 ฟังก์ชันหลักสำหรับโปรแกรมสื่อสารอนุกรมแบบกระตุ้นด้วยอินเตอร์รัปต์

```
main ( )
{
char c;
    port_ads = 0x2f8 ;          //COM2, ถ้าเป็น COM1 ใช้ ox3f8
    baudset (2400) ;
    comparam ( PARITY_NONE, 1,8 ) ;
    com_buffers (TRUE, 2048 , 2048 ) ;
    intinit ( 3 ) ;   // IRQ3 สำหรับ COM2, IRQ4 สำหรับ COM1
    for ( ;; ) {
        if ( kbhit ( ) ) {
            c = getch ( ) ;
            if ( c == 27 )
                break ;          // วิธีออกจากโปรแกรมที่ง่ายที่สุด
            comput ( c ) ;
        }
        if ( getbuf ( &inring , &c ) ) //    อ่านตัวอักษรที่รับได้
```

รูปที่ 6.11(ต่อ) ฟังก์ชันหลักสำหรับโปรแกรมสื่อสารอนุกรมแบบกระตุ้นด้วยอินเตอร์รัปต์

```
            putch ( c ) ;
        }
    }
    /*-------------------------------------------------------------------*/
    comput ( char c )
    {
        if ( outring.count || ! ( _inp ( REG_LSTAT ) &LSTAT_THRE ) )
            putbuf ( &outring, c ) ;          // ใส่ลงบัฟเฟอร์ส่งข้อมูล

        else
            _outp (REG_TX, c ) ;
    }
    /*-------------------------------------------------------------------*/
```

บทที่ 7

การออกแบบและการประยุกต์ใช้งาน

จากอดีตที่ผ่านมาการไปบันทึกตัวเลขกำลังไฟฟ้าที่ถูกใช้ไปของบ้านแต่ละหลังจะใช้วิธีการไปจดบันทึกลงสมุด ซึ่งจะเสียเวลาในการจดบันทึกและเวลาในการคำนวณออกมาเป็นจำนวนเงินที่ผู้ใช้ต้องจ่ายให้การไฟฟ้า ในยุคต่อมาการบันทึกค่ากำลังไฟฟ้ายังใช้วิธีเดิมอยู่ส่วนการคำนวณออกมาเป็นจำนวนเงินจะเป็นการใช้คอมพิวเตอร์ช่วยในการคิด แต่ก็ยังเสียเวลาในการข้อมูลเข้าคอมพิวเตอร์อยู่ดี ในยุคปัจจุบันนี้ไม่ต้องมีการจดบันทึกลงลงสมุดอีกแล้ว เพราะคอมพิวเตอร์มีขนาดเล็กลงสามรถคือเคลื่อนที่ไปไหนมาไหนได้สดวก ดังนั้นจึงง่ายต่อการบันทึกข้อมูลเข้าคอมพิวเตอร์ที่มีขั้นตอนเพียงขั้นตอนเดียว ในตอนบันทึกที่มิเตอร์วัดค่ากำลังไฟฟ้าที่ถูกใช้ไปแต่ก็ยังมีที่เสียอยู่คือ เครื่องคอมพิวเตอร์ขนาดเล็กที่ว่านี้มีราคาค่อนข้างสูง

แต่เครื่องบันทึกค่าไฟฟ้าที่จะกล่าวถึงในบทนี้ จะมีราคาถูกกว่าและเล็กกว่ามากโดยการนำไมโครโทรลเลอร์ตระกูล MCS 51 เบอร์ 89C51 มาใช้ในการบันทึกและจดจำข้อมูล มีหลักการใช้งานอยู่ว่าเครื่องนี้จะถือติดตามตัวไปบันทึกข้อมูลแล้วไว้ในหน่วยความจำ ซึ่งหน่วยความจำนี้จะใช้หน่วยความจำแบบพิเศษ ประเภท Non - volatile (NV REM) มีแบตเตอรี่ แบคอัพภายในตัวเอง ทำให้สามารถตัดแหล่งจ่าย (Powes Supply) ออกจากตัว NV RAM โดยไม่ทำให้ข้อมูลที่บรรจุอยู่ภายใน NV RAM สูญหาย อีกทั้งยังใช้กำลังไฟต่ำ เป็นข้อได้เปรียบหน่วยความจำชนิดอื่น (statio Random Aoces Memory) ทั่ว ๆ ไป ซึ่งใช้ power สูงกว่า NV RAM อีกทั้งยังต้องมีไฟ Baole - up ไว้ตลอด เพื่อกันการสูญหายของข้อมูลทำให้อายุการใช้งาน Battery สั้นลง

การใช้งานเครื่องบันทึกข้อมูลไร้สาย

เมื่อ ON power ของเครื่องบันทึกข้อมูลไร้สาย จะปรากฏ ข้อความบนจอ LCD ถึง Displary (1) เป็นเวลาประมาณ 3 Seo

Display (1)

NO TE BOOK FOR
POWER METER

After 3 Seo

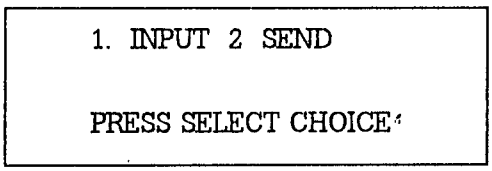
Display (2)

INSTRUMENT KMITL
PROJECT 3I 2538

3 Seo

จากนั้นเครื่องบันทึกข้อมูลไร้สายจะแสดงข้อความบน LCD ดังใน Display 3

Display (3)



เพื่อให้ผู้ทำการบันทึกเลือกการทำงาน โดยในกรณี ที่กด Key 2 [SEN D] เครื่องบันทึกข้อมูลไร้สายจะทำการส่งข้อมูลในลักษณะ อนุกรมออกทางระบบส่งซึ่งใช้ INFARED RAY เป็นค่าส่ง ข้อมูลจะถูกส่งออกไปจนกว่าจะครบตามจำนวนของข้อมูลที่ถูกบันทึก ในขณะนี้เครื่องกำลังทำการส่งข้อมูล LCD จะแสดงผลดังใน Display (4)

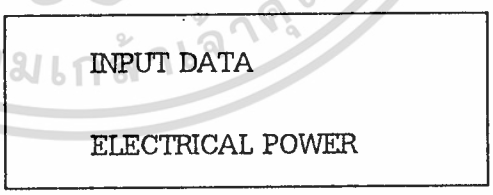
Display (4)

เมื่อข้อมูลถูกส่งออกไปจนหมดแล้ว LCD บนเครื่องบันทึกข้อมูลจะแสดงผลดังใน Display (5) เป็นเวลา 3 Sec.

Display (5)

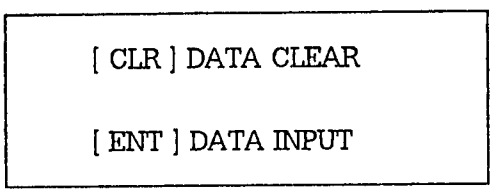
จากนั้นจะกลับเข้าสู่การแสดงผลของ Display (3) อีกครั้งหนึ่ง
ในกรณี ที่กด Key [1] เครื่องบันทึกข้อมูลไร้สายจะแสดง Display (6) เป็นเวลา 3 Sec.

Display (6)



และจะเข้าสู่ Display (7) ในเวลาต่อมา

Display (7)



เพื่อให้ผู้ทำการบันทึก เลือกใช้การทำงานโดยในกรณี กด Key [CLR] เครื่องบันทึกจะทำการลบข้อมูลภายในหน่วยความจำทิ้งทั้งหมด และจะกลับเข้าสู่ Display (7) อีกครั้ง

ในกรณีกด Key [ENT]

ผู้ทำการบันทึกจะสามารถทำการบันทึกค่าของตัวเลขต่าง ๆ ตามลำดับดังนี้

เมื่อกด Key [ENT] เครื่องบันทึกข้อมูลไร้สายจะแสดงผลดังใน Display (8) เพื่อให้ผู้ใช้สามารถป้อนค่า วัน , เดือน , ปี โดย เมื่อป้อนจำนวนวัน ครบ 2 หลัก เครื่องก็จะทำการ step มาที่การป้อนเดือน โดยอัตโนมัติ เพื่อให้ทำการป้อนจำนวนเดือนต่อไป หลักจากป้อนตัวเลขในเดือน [MONT.] ครบสองหลักแล้วก็จะกระโดดไปตำแหน่งของ [year] ซึ่งมีตัวเลข 4 หลัก

Display (9)

DATE _ _ MONTH _ _
YEAR _ _ _ _

เมื่อทำการบันทึกวัน / เดือน / ปี เรียบร้อยแล้ว LCD จะแสดง USO ดังใน Display (9)

Display (9)

NUMBER _ _ _ _ _
POWER _ _ _ _ _

เพื่อให้ผู้บันทึกได้ทำการบันทึก รหัส ของ USER ลงในตำแหน่ง NUMBER ซึ่งมีอยู่ด้วยกัน 4 หลัก และจำนวนของการใช้ไฟใน ช่องของ POWER ซึ่งมีอยู่ 4 หลัก เช่นกัน (เช่นค่า code ของ หม้อไฟฟ้า มี 7 หลัก ค่า POWER มี 5 หลัก แต่ในการทำงานนี้ เนื่องจากเป็น Projeot ทดลอง จึงใช้จำนวนของ NUMBER และ POWER เพียงอย่างละ 4 หลัก เป็นแนวทางในการพัฒนาเครื่องบันทึกข้อมูลไร้สายไปใช้กับงาน ในด้านอื่น ๆ อีกต่อไป)

หลังจากป้อนค่า code ของ USER ในช่องตำแหน่ง NUMBER เป็นจำนวน 4 หลักแล้ว เครื่องบันทึกก็จะไปแสดงผลในตำแหน่งช่อง POWER เพื่อให้ผู้บันทึกทำการบันทึก ปริมาณการใช้งานของ USER นั้น ๆ ในตำแหน่ง ของ USER code นั้น ๆ

เมื่อทำการป้อนปริมาณการใช้ของ USER ครบทั้ง 4 หลัก ให้ผู้บันทึกทำการกดปุ่ม [ENT] เพื่อเป็นการยืนยันว่าจะเก็บค่าที่กำลังแสดงผลอยู่บนจอ LCD ในปัจจุบัน เข้าไปเก็บไว้ในหน่วยความจำของเครื่อง

และ LCD ก็จะกลับไปแสดงผลดังใน Display (9) อีกครั้งหนึ่งเพื่อทำการป้อนค่าปริมาณการใช้ของ USER

หากผู้บันทึกได้ทำการบันทึกปริมาณการใช้ USER ต่างๆ เป็นที่เรียบร้อยแล้วต้องการออกจากระบบการบันทึกให้ทำการกด [CLP/END] จอ LCD จะกลับเข้ามาแสดงผลดังในรูป

Display (9)

END INPUT DATA
[ENT] [ESC]

ทำการกด [ENT] เพื่อสิ้นสุดกดใช้เครื่องบันทึก

ทำการกด [ESC] เพื่อกลับเข้าสู่ Menu Disploy (9) อีกครั้ง

การจัด Memory map

ในเครื่อง WIRBLESS DATA STORAGE จะจัดแบ่ง Memory ออกเป็น 4 ส่วนด้วยกัน อันได้แก่

1) COUNTER BUFFER ซึ่งจะเริ่มทาง Address 17394 - 17398 H จำนวน 4 byte โดยค่าที่เก็บ ใน Memory ในส่วนนี้จะเกิดจำนวนของ ข้อมูลที่ได้บันทึกเข้ามาไว้ในเครื่อง เพื่อ ให้ทราบถึง จำนวนของข้อมูลที่ถูกจัดเก็บไว้ภายใน Memory

2) Power consumption data area

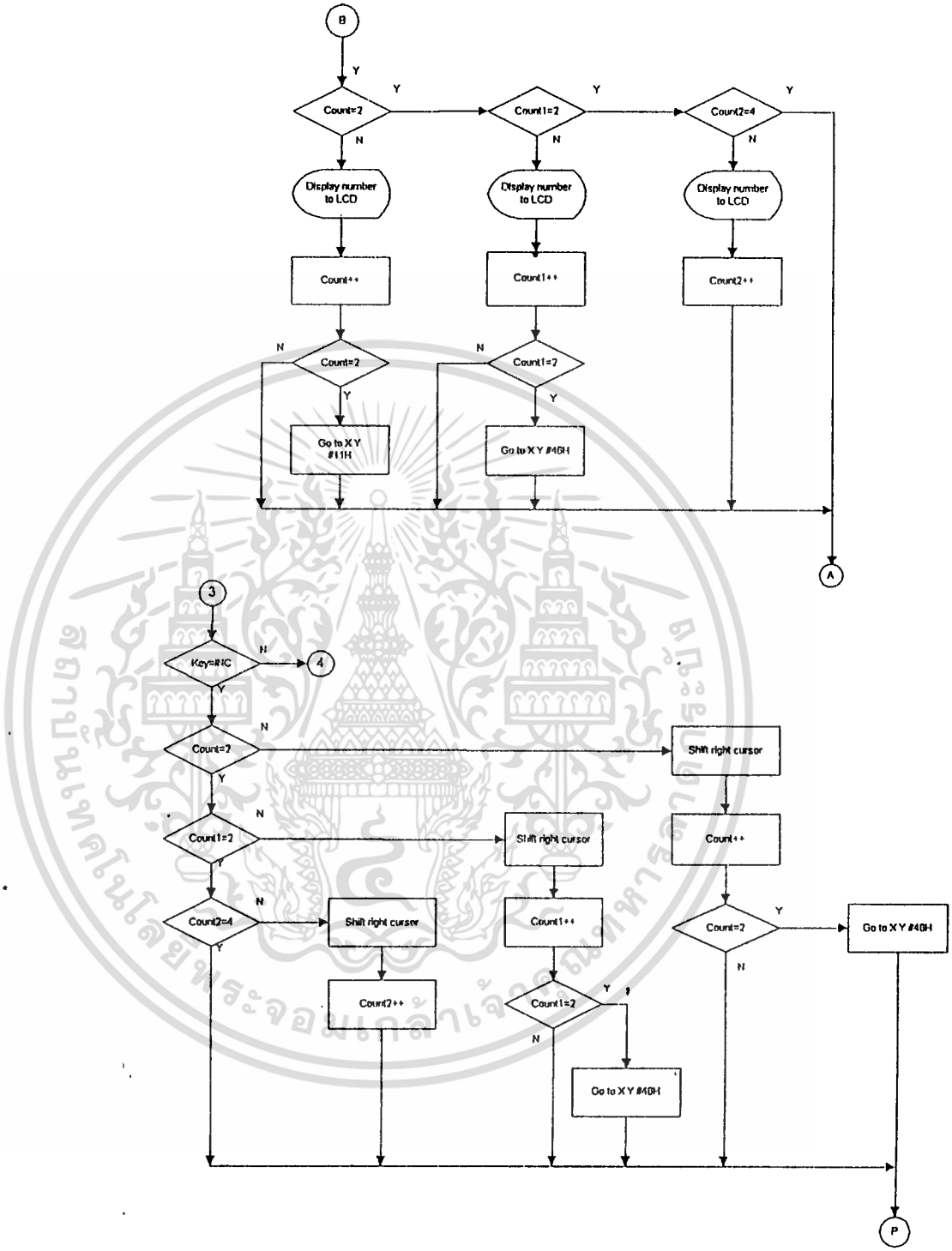
พื้นที่ Memory ในส่วนนี้จะถูกใช้ในการเก็บค่าปริมาณการใช้ไฟฟ้าของ USER แต่ละบ้าน โดยจัดให้บันทึก ข้อมูลได้ 4 หลัก ใช้แทนค่า KW - h บน Meter ไฟฟ้า โดยส่วนของข้อมูลจะเริ่มที่ ADDRESS 0000 และจะเพิ่มขึ้นครั้งละ 4 byte ไปจนถึงส่วนของ USER code area ตามจำนวนของ USER ที่ป้อน

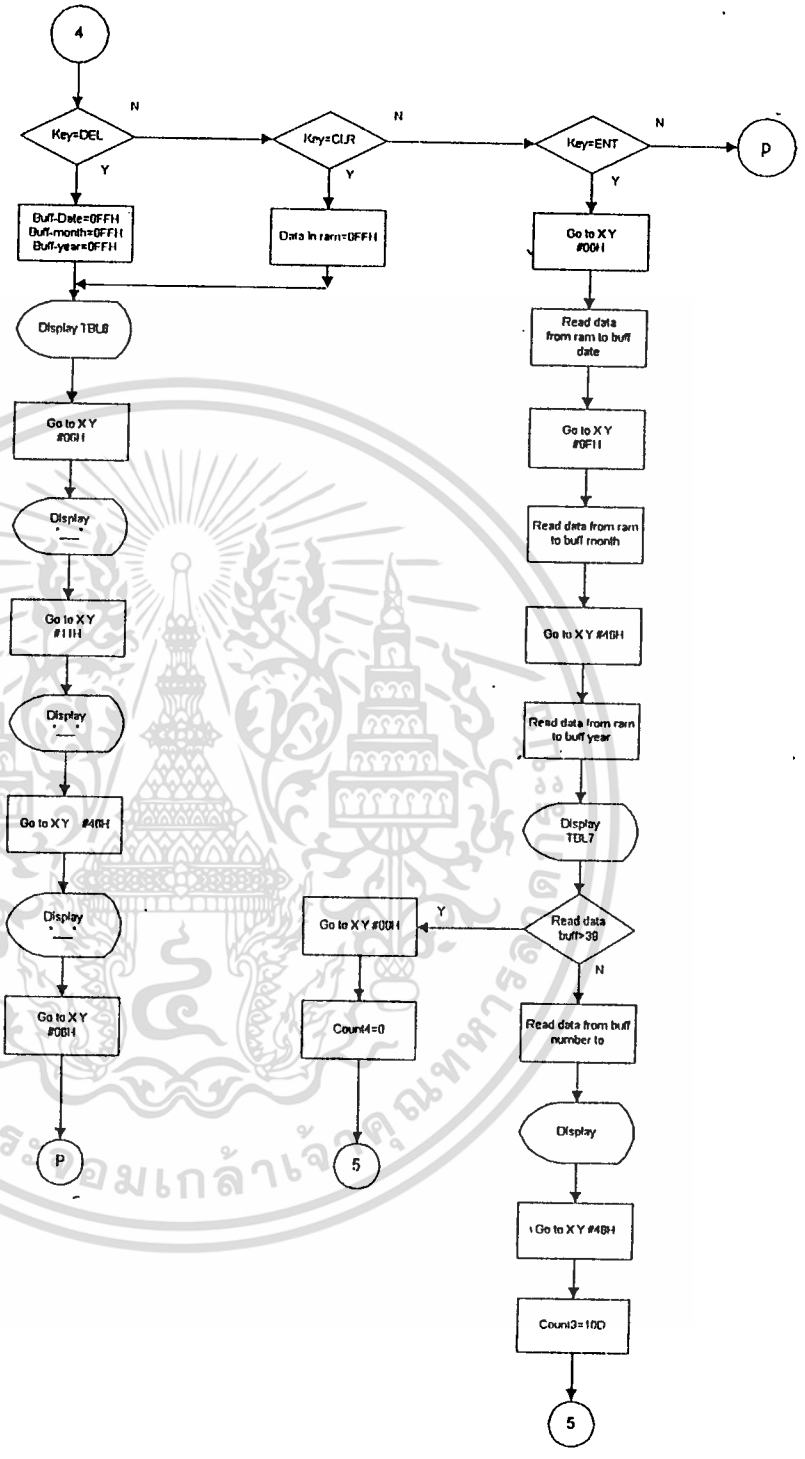
3) USER CODE ARGA

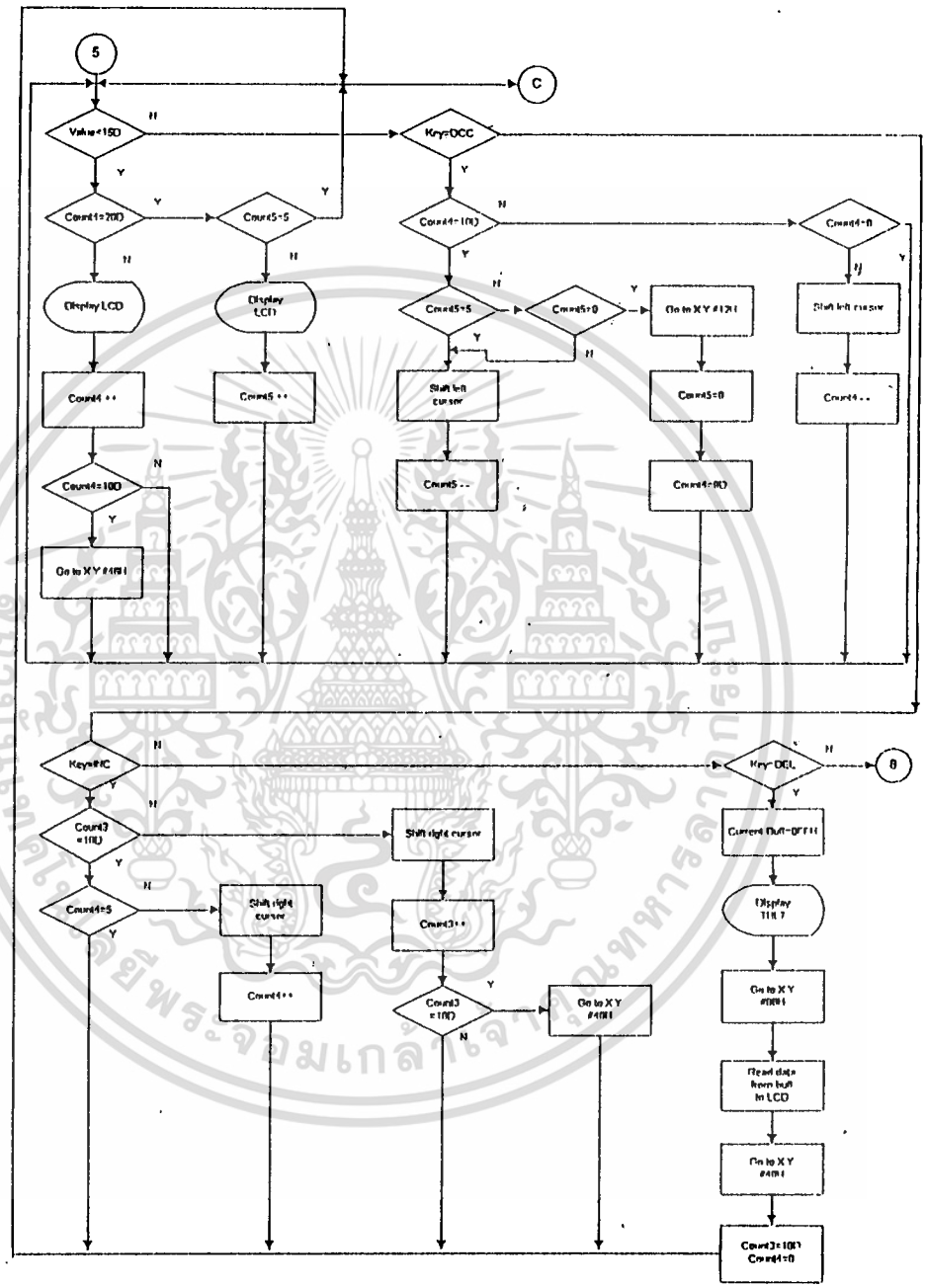
เป็นชิ้นที่ใช้จัดเก็บข้อมูลในส่วนของ CODE ของ Meter ซึ่งจะใช้เป็น Code ตัวแทนไฟในแต่ละบ้านโดยจะใช้ข้อมูลขนาด 4 bate แทน USER 1 บ้าน มี address ตั้งต้นที่ 8192 (10) ไปจนกระทั่งถึง d / m / y area

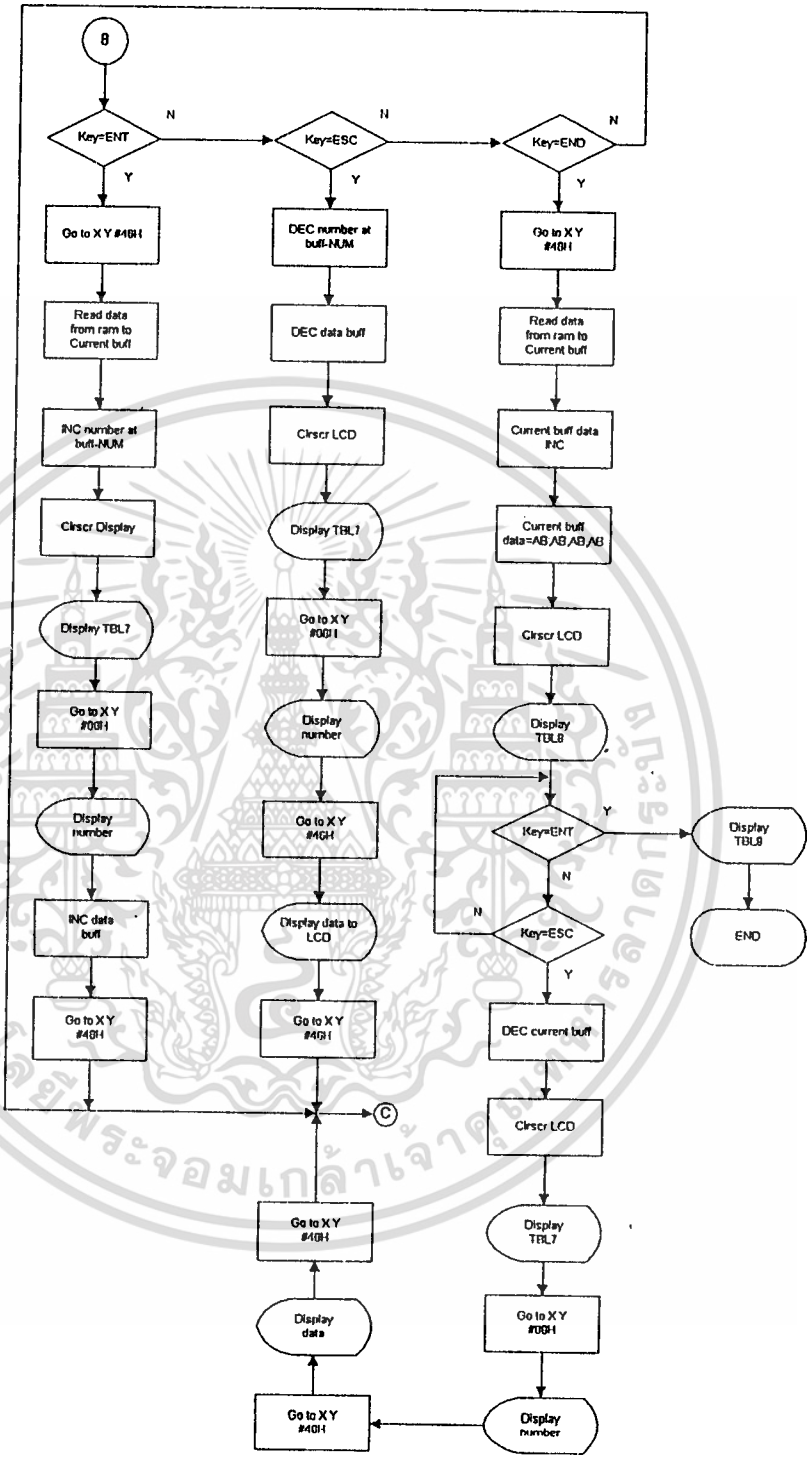
4) d/m/y data area

ใช้กับข้อมูล วัน/เดือน/ปี ของข้อมูลที่ใช้เข้ามาในแต่ละครั้ง มีขนาดวัน 2 byte เดือน 2 byte year 4 byte ทั้งหมด 8 byte ต่อ 1 bate address แรกเริ่มต้นที่ 17398_{10} ไปจนสิ้น address $32767_{(10)}$









```
; NOTE BOOK PROGRAM
; FILE NOTEB.ASM

        CPU "8051.TBL"
        HOF "INT8"

        ORG   0000H

  B:        EQU   0F0H
  A:        EQU   0E0H
  P1:       EQU   090H
  P3.2:     EQU   0B2H
  P3.3:     EQU   0B3H
  P3.4:     EQU   0B4H
  P3.5:     EQU   0B5H
  P3:       EQU   0B0H
  DPH:      EQU   083H
  DPL:      EQU   082H
  SP:       EQU   081H
 COMMAND:   EQU   0800H
 BUSY:      EQU   8001H
 WRITEDAT:  EQU   8002H
 READDAT:   EQU   8003H
 BUFF_DATE: EQU   7FEBH
 BUFF_MONT: EQU   7FEEH
 BUFF_YEAR: EQU   7FF1H
 BUFF_NUMB: EQU   7FF6H
 KEY_1:     EQU   081H
 KEY_5:     EQU   082H
 KEY_9:     EQU   084H
 KEY_DEL:   EQU   088H
 KEY_2:     EQU   041H
 KEY_6:     EQU   042H
 KEY_0:     EQU   044H
 KEY_CLR:   EQU   048H
 KEY_3:     EQU   021H
```

```
KEY_7:    EQU  022H
KEY_INC:  EQU  024H
KEY_ENT:  EQU  028H
KEY_4:    EQU  011H
KEY_8:    EQU  012H
KEY_DEC:  EQU  014H
KEY_ESC:  EQU  018H
DEL_:     EQU  050H
CLR_:     EQU  051H
INC_:     EQU  052H
DEC_:     EQU  053H
ESC_:     EQU  054H
ENT_:     EQU  055H
COUNT1:   EQU  07FE9H
COUNT2:   EQU  07FEAH
UPPER:    EQU
LOWER:    EQU

;-----------------------------------------------
START:    MOV  R2,#00H          ; POWER ON DELAY
RES:      MOV  R3,#00H
          DJNZ R3,$
          DJNZ R2,RES
          ;**********************
          MOV  SP,#020H         ; MOVE STACK TO ADDRESS 20H
          MOV  P1,#0FFH         ; SET PORT1 FOR INPUT
INITLCD:  MOV  DPTR,#COMMAND
          MOV  A,#38H           ; 8 BIT , 2 LINE , 5X7 DOT
          MOVX @DPTR,A
          LCALL WAIT
          MOV  A,#0CH           ; DISPLAY ON, CURSOR OFF
          MOVX @DPTR,A
          LCALL WAIT
          MOV  A,#06H           ; INCREAMENT
          MOVX @DPTR,A
          LCALL WAIT
          MOV  A,#01H           ; CLEAR DISPLAY & HOME
```

```
        MOVX  @DPTR,A
        LCALL WAIT
        ;************************
        MOV   DPTR,#TBL1          ; DISPLAY TABLE 1
        LCALL DISP_PAGE           ; DISPLAY 1 SCREEN
        MOV   R2,#30D
        LCALL DELAY_0.1S          ; DELAY 3SEC.
        LCALL CLR_LCD
        MOV   DPTR,#TBL2          ; DISPLAY TABLE 2
        LCALL DISP_PAGE           ; DISPLAY 1 SCREEN
        MOV   R2,#30D            ; DELAY 3 SEC.
        LCALL DELAY_0.1S
        LCALL CLR_LCD
        MOV   DPTR,#TBL3          ; DISPLAY TABLE 3 MENU
        LCALL DISP_PAGE
        ;************************
SCAN:   SETB  P3.2
        MOV   R0,#80H
        LCALL SCAN_KEY            ; SCAN KEY COLUMN 1
        LCALL SCAN_KEY1           ; CHECK KEY
        XCH   A,R0
        LCALL UP_KEY              ; WAIT UP KEY [1]INPUT,[2]SEND
        CLR   P3.2
        XCH   A,R0
        CJNE  A,#KEY_1,ELSE_KEY2  ; JUMP IF NOT KEY_1 PRESS
        LJMP  IN_DATA
        ;************************
ELSE_KEY2:  SETB  P3.3            ; SCAN KEY COLUMN 2
        MOV   R0,#40H
        LCALL SCAN_KEY
        LCALL SCAN_KEY1           ; CHECK KEY
        XCH   A,R0
        LCALL UP_KEY
        CLR   P3.3
        XCH   A,R0
        CJNE  A,#KEY_2,SCAN       ; JUMP IF NOT KEY_2 PRESS
        LJMP  SEND_DATA
```

```
        ;************************
IN_DATA:  LCALL CLR_LCD
          MOV   DPTR,#TBL4          ; DISPLAY TABLE 4
          LCALL DISP_PAGE
          MOV   R2,#30D             ; DELAY 3 SEC.
          LCALL DELAY_0.1S
          LCALL CLR_LCD
          MOV   DPTR,#TBL12
          LCALL DISP_PAGE
          ;
SCAN11:   SETB  P3.2
          MOV   R0,#80H
          LCALL SCAN_KEY            ; SCAN KEY COLUMN 1
          LCALL SCAN_KEY1           ; CHECK KEY
          XCH   A,R0
          LCALL UP_KEY              ; WAIT UP KEY [1]INPUT,[2]SEND
          CLR   P3.2
          XCH   A,R0
          CJNE  A,#KEY_1,ELSE_KEY2  ; JUMP IF NOT KEY_1 PRESS
          AJMP  NEW_DAT
          ;************************
ELSE_KEY22:  SETB  P3.3             ; SCAN KEY COLUMN 2
          MOV   R0,#40H
          LCALL SCAN_KEY
          LCALL SCAN_KEY1           ; CHECK KEY
          XCH   A,R0
          LCALL UP_KEY
          CLR   P3.3
          XCH   A,R0
          CJNE  A,#KEY_2,SCAN11     ; JUMP IF NOT KEY_2 PRESS
          AJMP  CONTI
          ;
NEW_DAT:  MOV   DPTR,#COUNT1
          MOV   A,#00H
          MOVX  @DPTR,A
          INC   DPTR
          MOVX  @DPTR,A
```

```
        ;************************
CONTI:  LCALL CLR_LCD
        MOV   DPTR,#TBL5          ; DISPLAY TABLE 5
        LCALL DISP_PAGE           ; SELECT CHOICE [CLR] ,[ENT]
        ;************************
SCAN_ENT: SETB P3.4               ; SCAN KEY COLUMN 3
        MOV   R0,#20H
        LCALL SCAN_KEY
        LCALL SCAN_KEY1
        XCH   A,R0
        LCALL UP_KEY
        CLR   P3.4
        XCH   A,R0
        CJNE  A,#KEY_ENT,SCAN_CLR  ; JUMP IF NOT PRESS KEY_ENT
        LJMP  PRS_ENT
        ;
SCAN_CLR: SETB P3.3               ; SCAN KEY COLUMN 2
        MOV   R0,#40H
        LCALL SCAN_KEY
        LCALL SCAN_KEY1
        XCH   A,R0
        LCALL UP_KEY
        CLR   P3.3
        XCH   A,R0
        CJNE  A,#KEY_CLR,SCAN_ENT
        LCALL CLR_DATA
        ;************************
PRS_ENT: MOV  DPTR,#COMMAND       ; SET LCD
        MOV   A,#0FH              ; CURSOR ON , BLINK
        MOVX  @DPTR,A
        LCALL WAIT
        MOV   A,#06H              ; ENTRY MODE SET
        MOVX  @DPTR,A
        LCALL CLR_LCD
        ;************************
        MOV   DPTR,#TBL6          ; DISPLAY TABLE 6
```

```
          LCALL DISP_PAGE
          ;************************
          MOV   DPTR,#BUFF_DATE     ; READ DATA FROM BUFFER DATE
          MOVX  A,@DPTR
          SUBB  A,#039H
          JNC   EDIT_DAT            ; IF DATA IN BUFFER MORE
          ;************************  ; THAN 039H JUMP
          LCALL DISP_DAT
          LCALL WAIT
          MOV   A,#06H
          ;
EDIT_DAT:   LCALL GOTOXY
          MOV   R1,#00H             ; SET COUNT 1 = 0
          MOV   R2,#00H             ; SET COUNT 2 = 0
          MOV   R3,#00H             ; SET COUNT 3 = 0
          ;
EDIT_DAT1:  LCALL SCAN_KEY3         ; SCAN ALL KEY
          MOV   R4,A
          SUBB  A,#040H             ; CHECK IF CHARACTOR <=39 BE
          JNC   DECRET              ; - CONTINOUE
          CJNE  R1,#02H,ENT_DATE
          CJNE  R2,#02H,ENT_MONT
          CJNE  R3,#04H,ENT_YEAR
          AJMP  EDIT_DAT1
          ;
ENT_DATE:   LCALL WAIT
          MOV   A,R4
          LCALL WRITE
          INC   R1
          CJNE  R1,#02H,EDIT_DAT1
          LCALL WAIT
          MOV   A,#0FH
          LCALL GOTOXY
          AJMP  EDIT_DAT1
          ;************************
ENT_MONT:   LCALL WAIT
          MOV   A,R4
```

```
        LCALL WRITE
        INC  R2
        CJNE  R2,#02,EDIT_DAT1
        LCALL WAIT
        MOV  A,#046H
        LCALL GOTOXY
        AJMP  EDIT_DAT1
        ;************************
ENT_YEAR:   LCALL WAIT
        MOV  A,R4
        LCALL WRITE
        INC  R3
        AJMP  EDIT_DAT1
        ;************************
 DECRET:   MOV  A,R4
        CJNE  A,#DEC_,INCRET
        CJNE  R1,#02H,SHF_CUR
        CJNE  R2,#02H,SHF_CUR1
        CJNE  R3,#04H,SHF_CUR2
        ;
LOOP_SHF:   LCALL WAIT
        LCALL SHF_LFT
        DEC  R3
        LJMP  EDIT_DAT1
        ;************************
SHF_CUR2:   CJNE  R3,#00H,LOOP_SHF
        LCALL WAIT
        MOV  A,#010H
        LCALL GOTOXY
        MOV  R3,#00H
        MOV  R2,#01H
        LJMP  EDIT_DAT1
        ;************************
SHF_CUR1:   CJNE  R2,#00H,SHIFT
        LCALL WAIT
        MOV  A,#07H
        LCALL GOTOXY
```

```
          MOV   R2,#00H
          MOV   R1,#01H
          LJMP  EDIT_DAT1
          ;
SHIFT:    LCALL WAIT
          LCALL SHF_LFT
          DEC   R2
          LJMP  EDIT_DAT1
          ;
SHF_CUR:  CJNE  R1,#00H,SHIFT1
          LJMP  EDIT_DAT1
          ;************************
SHIFT1:   LCALL WAIT
          LCALL SHF_LFT
          DEC   R1
          LJMP  EDIT_DAT1
          ;************************
INCRET:   MOV   A,R4
          CJNE  A,#INC_,DELET
          CJNE  R1,#02H,SHF_RCUR
          CJNE  R2,#02H,SHF_RCUR1
          CJNE  R3,#04H,SHF_RCUR2
          LJMP  EDIT_DAT1
          ;************************
SHF_RCUR: LCALL SHF_RHT
          INC   R1
          CJNE  R1,#02H,EDIT_DAT1
          LCALL WAIT
          MOV   A,#0FH
          LCALL GOTOXY
          LJMP  EDIT_DAT1
          ;************************
SHF_RCUR1: LCALL SHF_RHT
          INC   R2
          CJNE  R2,#02H,EDIT_DAT1
          LCALL WAIT
          MOV   A,#046H
```

```
            LCALL GOTOXY
            LJMP  EDIT_DAT1
SHF_RCUR2:  LCALL SHF_RHT
            INC   R3
            LJMP  EDIT_DAT1
            ;*************************
DELET:      MOV   A,R4
            CJNE  A,#DEL_,CLEAR
            ;************************
DELET2:     LCALL CLR_LCD
            LCALL WAIT
            MOV   DPTR,#TBL6
            LCALL DISP_PAGE
            LJMP  EDIT_DAT1
            ;
CLEAR:      MOV   A,R4
            CJNE  A,#CLR_,ENT
            ;
            LCALL CLR_DATA
            LJMP  DELET2
            ;************************
ENTER:      MOV   A,R4
            CJNE  A,#ENT_,ESCAP
            LCALL WAIT
            MOV   A#06H
            LCALL GOTOXY
            MOV   DPTR,#BUFF_DATE     ; KEEP DATA FROM LCD IN BUFFER
            LCALL READ
            MOVX  @DPTR,A
            INC   DPTR
            LCALL READ
            MOVX  @DPTR,A
            MOV   A,#0FH
            LCALL GOTOXY
            MOV   DPTR,#BUFF_MONT     ; KEEP DATA FROM LCD IN BUFFER
            LCALL READ
            MOVX  @DPTR,A
```

```
        INC   DPTR
        LCALL READ
        MOVX  @DPTR,A
        MOV   A,#046H
        LCALL GOTOXY
        MOV   DPTR,#BUFF_YEAR     ; KEEP DATA FROM LCD IN BUFFER
        MOV   R6,#04H
YEAR:   LCALL READ
        MOVX  @DPTR,A
        INC   DPTR
        DJNZ  R6,YEAR
        MOV   DPTR,#TBL7          ; TABLE 7
        LCALL DISP_PAGE           ; DISPLAY TABLE 7
        ;
        MOV   DPTR,#COUNT1
        MOVX  A,@DPTR
        MOV   R1,A
        INC   DPTR
        MOVX  A,@DPTR
        MOV   R2,A
        MOV   R4,#00H             ; COUNT4=0
        MOV   R5,#00H             ; COUNT4=0
        MOV   A,#08H
        LCALL GOTOXY
EDIT_DATA2:   LCALL SCAN_KEY3
        MOV   R6,A
        SUBB  A,#040H
        JNC   DECRET1
        CJNE  R4,#04D,ENT_MEMBER
        CJNE  R4,#04D,ENT_POWER
        AJMP  EDIT_DATA2
        ;**********************
ENT_MEMBER:  MOV   A,R6
        LCALL WRITE
        INC   R4
        CJNE  R4,#04D,DEIT_DATA2
        MOV   A,#046H
```

```
            LCALL GOTOXY
            AJMP  EDIT_DATA2
           ;************************
ENT_POWER:  MOV   A,R6
            LCALL WRITE
            INC   R5
            AJMP  EDIT_DATA2
           ;************************
DECRET1:    MOV   A,R6
            CJNE  A,#DEC_,INCRET1
            CJNE  R4,#04D,SHF_LFET
            CJNE  R5,#04D,BAK_LINE
           ;************************
LOOP_SHF1:  LCALL SHF_LFT                ; *****
            DEC   R5
            AJMP  EDIT_DATA2
           ;
BAK_LINE:   CJNE  R5,#00H,LOOP_SHF1
            MOV   A,#0BH
            LCALL GOTOXY
            MOV   R5,#00H
            MOV   R4,#03D
            AJMP  EDIT_DATA2
           ;
SHF_LEFT:   CJNE  R4,#00H,SHF_LEFT1
           LJMP  EDIT_DATA2
SHF_LEFT1:  LCALL SHF_LFT
            DEC   R4
            LJMP  EDIT_DATA2
          ;************************
INCRET1:    MOV   A,R6
            CJNE  A,#INC_,DEL1
            CJNE  R4,#04D,SHF_RIGHT
            CJNE  R5,#04D,SHF_RIGHT1
           LJMP  EDIT_DATA2
          ;************************
SHF_RIGHT1: LCALL SHF_RHT
```

```
            INC  R5
            LJMP EDIT_DATA2
           ;************************
SHF_RIGHT:  LCALL SHF_RHT
            INC  R4
            CJNE R4,#04D,EDIT_DAT2
            MOV  A,#046H
            LCALL GOTOXY
  DEL1:     MOV  A,R6
            CJNE A,#DEL_,ENT1
            MOV  DPTR,#TBL7
            LCALL DISP_PAGE
           ;************************
  ENT1:     MOV  A,R6
            CJNE A,#ENT_ESC
            MOV  A,#08H
            LCALL GOTOXY
  RMEM:     MOV  DPTR,#COUNT1
            MOVX A,@DPTR
            MOV  R1,A
            INC  DPTR
            MOVX A,@DPTR
            MOV  R2,A
            MOV  DPH,R2          ;COUNTER 1
            MOV  DPL,R1          ;COUNTER 2
            MOV  R6,#04H
 RMEM1:     LCALL READ
            MOVX @DPTR,A
            INC  DPTR
            DJNZ R6,RMEM1
            LCALL ADD_HEX
            ;
 RPOWER:    MOV  A,#46H
            LCALL GOTOXY
            MOV  DPTR,#COUNT1
            MOVX A,@DPTR
            MOV  R1,A
```

```
        INC   DPTR
        MOVX  A,@DPTR
        MOV   R2,A
        MOV   DPH,R2          ;COUNTER 1
        MOV   DPL,R1          ;COUNTER 2
        MOV   R6,#04H
RPOWER1:  LCALL READ
        MOVX  @DPTR,A
        INC   DPTR
        DJNZ  R6,RPOWER1
        LCALL CLR_LCD
        LCALL WAIT
        MOV   DPTR,#TBL7
        LCALL DISP_PAGE
        LCALL COUNT
        LJMP  EDIT_DAT2
        ;
 ESC:    MOV   A,R6
        CJNE  A,#ESC_,CLR_END
        LCALL SUBB_HEX
        LCALL CLR_LCD
        LCALL WAIT
        MOV   DPTR,#TBL7
        LCALL DISP_PAGE
        LJMP  EDIT_DATA2
       ;************************
CLR_END:   MOV   A,R6
        CJNE  A,END_,EDIT_DATA2
        LCALL COUNT
        MOV   A,#03H
        MOVX  @DPTR,A
        INC   DPTR
        MOVX  @DPTR,A
        INC   DPTR
        MOVX  @DPTR,A
        INC   DPTR
        MOVX  @DPTR,A
```

```
        LCALL CLR_LCD
        LCALL WAIT
        MOV   DPTR,#TBL8
        LCALL DISP_PAGE
        ;****************************
ENT3:   MOV   A,R6
        CJNE  A,#ENT_,ESC2
        LCALL CLR_LCD
        LCALL WAIT
        MOV   DPTR,#TBL9
        LCALL DISP_PAGE
        AJMP  END_OFF
ESC2:   MOV   A,R6
        CJNE  A,#ESC_,ENT3
        LCALL CLR_LCD
        LCALL WAIT
        MOV   DPTR,#TBL7
        LCALL DISP_PAGE
        MOV   A,#080H
        LCALL GOTOXY
        LJMP  EDIT_DATA2


;************ REMARK NOT COPLETE ****************
;***************** DATA TABLE *******************
TBL1: DFB "   NOTE BOOK FOR    "
      DFB "   POWER METER      "
TBL2: DFB "  INSTRUMENT KMITL  "
      DFB "  PROJECT  3I 2538  "
TBL3: DFB "  1.INPUT   2.SEND  "
      DFB " PRESS SELECT CHOICE"
TBL4: DFB "    INPUT DATA      "
      DFB "  ELECTRICAL POWER  "
TBL5: DFB "  [CLR] DATA CLEAR  "
```

```
        DFB  "  [ENT] DATA INPUT  "
TBL6:   DFB  " DATE     MONT      "
        DFB  " YEAR               "
TBL7:   DFB  " NUMBER             "
        DFB  " POWER              "
TBL8:   DFB  "   END INPUT DATA   "
        DFB  "   [ENT]    [ESC]   "
TBL9:   DFB  "       OFF          "
TBL10:  DFB  "     PLASE WAIT     "
TBL11:  DFB  "      COMPLETE      "
TBL12:  DFB  "    [1] NEW DATA    "
        DFB  "    [2] CONTINOUS   "

    ;************** END OF DATA TABLE ****************
    ;***  REMARK May be miss match of space between -
    ;***  - data on data table and on lcd ************




    ; ****************** sub program *****************
    ; ************************************************
    ; ************ SCAN_KEY *************************
SCAN_KEY:   MOV   A,P1              ; INPUT KEY PRESS
            ANL   A,#0FH
            CJNE  A,#00H,KEY_PRESS    ; IF KEY PRESS JUMP TO
            MOV   A,#00H              ; - KEY_PRESS
            RET
    ;************ KEY_PRESS **************************
KEY_PRESS:  MOV   R2,#010D
            LCALL DELAY_MS            ; DELAY 10 mS
            MOV   A,P1              ; KEY HAVE PRESS AGAIN
            ANL   A,#0FH
            CJNE  A,#00H,END_SCAN     ; IF NOT KEY PRESS BE CONTINOUS
            MOV   A,#00H
END_SCAN:   RET
    ;************ SCAN_KEY1 **************************
```

```
SCAN_KEY1:   CJNE  A,#01H,LINE2
             ORL   A,R0
             RET
             ;
LINE2:       CJNE  A,#02H,LINE3        ; A = OUT PUT
             ORL   A,R0
             RET
             ;
LINE3:       CJNE  A,#04H,LINE4
             ORL   A,R0
             RET
             ;
LINE4:       CJNE  A,#08H,END_SCAN1
             ORL   A,R0
END_SCAN1:   RET
             ;
UP_KEY:      MOV   A,P1
             ANL   A,#0FH
             CJNE  A,#00H,UP_KEY
             RET
; ************ DISPLAY PAGE ************************
DISP_PAGE:   MOV   A,#0
             LCALL GOTOXY
             MOV   R0,#20D             ; ONE LINE 20 CHARACTOR
DISP1:       MOV   A,#00H
             MOVC  A,@A+DPTR
             LCALL WRITE
             INC   DPTR
             DJNZ  R0,DISP1
             MOV   A,#040H             ; NEW LINE
             LCALL GOTOXY
             MOV   R0,#20D             ; ONE LINE 20 CHAR
DISP2:       MOV   A,#00H
             MOVC  A,@A+DPTR
             LCALL WRITE
             INC   DPTR
             DJNZ  R0,DISP2
```

```
        RET
    ;************ DELAY_MS *************************
DELAY_MS:   MOV  R3,#230D          ; DELAY 1 mS
    MS:   NOP
          NOP                    ; R2 INPUT
          DJNZ R3,MS
          DJNZ R2,DELAY_MS
          RET
    ;************ DELAY_0.1SEC *********************
DELAY_0.1S:   MOV  R3,179D         ; DELAY 0.1 S
    S2:   MOV  R4,#00H             ; R2 IN PUT
          DJNZ $
          NOP
          NOP
          DJNZ R3,S2
          DJNZ R2,DELAY_0.1S
          RET
    ;******************** DELAY_SEC ***************
DELAY_SEC:   MOV  R2,#10D
          LCALL DELAY_0.1S
          RET
    ;****************** WRITE ********************
 WRITE:   PUSH DPL
          PUSH DPH
          MOV  DPYR,#WRITEDAT
          MOVX @DPTR,A
          LCALL WAIT               ; WAIT LCD MODULE READY
          POP  DPH
          POP  DPL
          RET
          ;
  WAIT:   PUSH DPL
          PUSH DPH
          MOV  DPTR,#BUSY
  RDY1:   MOVX A,@DPTR
          JB   ACC.7,RDY1          ;BUSY FLAG
          POP  DPH
```

```
        POP   DPL
        RET
        ;
GOTOXY:   MOV   DPTR,#COMMAND
        SETB  ACC.7          ; SET DDRAM INSTRUCTION
        MOVX  @DPTR,A
        LCALL WAIT
        RET
        ;
 READ:   PUSH  DPL
        PUSH  DPH
        MOV   DPTR,#READDAT
        MOVX  A,@DPTR
        LCALL WAIT
        POP   DPH
        POP   DPL
        RET
 ;****************** SCAN_KEY3 ******************
SCAN_KEY3:   SETB  P3.2
        MOV   RO,#080H
        LCALL SCAN_KEY
        LCALL SCAN_KEY1
        XCH   A,R0
        LCALL UP_KEY
        CLR   P3.2
        XCH   A,R0
        ANL   A,#0FFH
        JZ    COLUMN2
        CJNE  A,#KEY_1,ELSE1
        MOV   A,"1"          ; CONFIG LETTER "1"
        RET
        ;
ELSE1:   CJNE  A,#KEY_5,ELSE2
        MOV   A,"5"          ; CONFIG LETTER "5"
        RET
        ;
ELSE2:   CJNE  A,#KEY_9,ELSE3
```

```
        MOV   A,"9"
        RET
        ;
ELSE3:  CJNE  A,#KEY_DEL,COLUM2
        MOV   A,#DEL_           ; VALUE IN PARAMETER 'DEL'
        RET
        ;                       ; - NOT CONFIG.
COLUM2: SETB  P3.3
        MOV   R0,#040H
        LCALL SCAN_KEY
        LCALL SCAN_KEY1
        XCH   A,R0
        LCALL UP_KEY
        CLR   P3.3
        XCH   A,R0
        ANL   A,#0FFH
        JZ    COLUM3
        CJNE  A,KEY_2,ELSE4
        MOV   A,"2"
        RET
        ;
ELSE4:  CJNE  A,#KEY_6,ELSE5
        MOV   A,"6"
        RET
        ;
ELSE5:  CJNE  A,#KEY_0,ELSE6
        MOV   A,"0"
        RET
        ;
ELSE6:  CJNE  A,#KEY_CLR,COLUM3
        MOV   A,#CLR_           ; VALUE OF PARAMETER 'CRL_'
        RET                     ; - HAS NOT CONFIG
        ;
COLUM3: SETB  P3.4
        MOV   R0,#020H
        LCALL SCAN_KEY
        LCALL SCAN_KEY1
```

```
        XCH   A,R0
        LCALL UP_KEY
        CLR   P3.4
        XCH   A,R0
        ANL   A,#0FFH
        JZ    COLUM4
        CJNE  A,#KEY_3,ELSE7
        MOV   A,"3"
        RET
        ;
ELSE7:  CJNE  A,#KEY_7,ELSE8
        MOV   A,"7"
        RET
        ;
ELSE8:  CJNE  A,#KEY_INC,ELSE9
        MOV   A,#INC_          ; VALUE  OF 'INC_' NOT CONFIG
        RET
        ;
ELSE9:  CJNE  A,#KEY_ENT,COLUM4
        MOV   A,#ENT_          ; VALUE  OF 'ENT' NOT CONFIG
        RET
        ;
COLUM4: SETB  P3.5
        MOV   R0,#010H
        LCALL SCAN_KEY
        LCALL SCAN_KEY1
        XCH   A,R0
        LCALL UP_KEY
        CLR   P3.5
        XCH   A,R0
        ANL   A,#0FFH
        JZ    SCAN_KEY3
        ;
        CJNE  A,#KEY_4,ELSE10
        MOV   A,"4"
        RET
        ;
```

```
ELSE10:   CJNE  A,#KEY_8,ELSE11
          MOV   A,"8"
          RET
          ;
ELSE11:   CJNE  A,#KEY_DEC,ELSE12
          MOV   A,#DEC_
          RET
          ;
ELSE12:   CJNE  A,#KEY_ESC,SCAN_KEY3
          MOV   A,#ESC_
          RET
          ;
  :************SUB RUTINE FOR DISPLAY DATE/MONTH/YEAR ************
DISP_DAT:   MOV   A,06H
          LCALL GOTOXY
          MOV   DPTR,#BUFF_DATE
          MOVX  A,@DPTR
          INC   DPTR
          LCALL WRITE
          MOVX  A,@DPTR
          LCALL WRITE
 MONT:    MOV   A,#0FH
          LCALL GOTOXY
          MOV   DPTR,#BUFF_MONT
          MOVX  A,@DPTR
          INC   DPTR
          LCALL WRITE
          MOVX  A,@DPTR
          LCALL WRITE
 YEAR:    MOV   A,#046H
          LCALL GOTOXY
          MOV   DPTR,#BUFF_YEAR
          MOV   R0,#030H
          MOV   R0,#04H
YEAR1:    MOVX  A,@DPTR          ; MOVE DATA TO BUFFER YEAR
          MOV   @R0,A
          INC   R0
```

```
        INC   DPTR
        DJNZ  R1,YEAR1
        MOV   DPTR,#WRITEDAT
        MOV   R0,#030H
        MOV   R1,#040H
DISP1:  MOV   A,@R0             ; DISPLAY YEAR TO LCD
        LCALL WAIT
        MOVX  @DPTR,A
        INC   R0
        DJNZ  R1,DISP1
        MOV   A,#06H
        LCALL GOTOXY
        RET
        ;******** SUBB_HEX ********
SUBB_HEX:  MOV  R3,#2
        MOV   R0,#34H
        MOV   @R0,#4
        INC   R0
        MOV   @R0,#0
        MOV   DPTR,#COUNT1
        DEC   R0
        CLR   C
SUBC:   MOVX  A,@DPTR
        SUBB  A,@R0
        MOVX  @DPTR,A
        INC   DPTR
        INC   R0
        DJNZ  R3,SUBC
        RET
        ;
ADD_HEX:  MOV  R3,#2
        MOV   R0,#30H
        MOV   @R0,#LOWER
        INC   R0
        MOV   @R0,#UPPER
        MOV   DPTR,#COUNT1
        DEC   R0
```

```
        CLR  C
ADDC:   MOVX A,@DPTR
        ADC  A,@R0
        MOVX @DPTR,A
        INC  DPTR
        INC  R0
        DJNZ R3,ADDC
        RET
        ;
       ;********** COUNT ************
COUNT:  MOV  DPTR,#COUNT1
        MOVX R1,@DPTR
        INC  R1
        CJNE R1,#00H,END3
        MOV  R1,#00H
        MOV  DPTR,#COUNT2
        MOVX R2,@DPTR
        INC  R2
        CJNE R2,#0FFH,END3
        MOV  R2,#00H
        RET
        ;
 END3:  LCALL MULTI          ; COUNT x 4
SUBB_1: MOV  R3,#2
        MOV  R0,#34H
        MOV  @R0,#1
        INC  R0
        MOV  @R0,#0
        MOV  DPTR,#COUNT1
        DEC  R0
        CLR  C
SUBC1:  MOVX A,@DPTR
        SUBB A,@R0
        MOVX @DPTR,A
        INC  DPTR
        INC  R0
        DJNZ R3,SUBC1
```

```
        RET
        ;
MULTI:  MOV  A,R1
        MOV  B,#04H
        MUL  AB
        MOV  R1,A
        MOV  A,R2
        MOV  R2,B
        MOV  B,#04H
        MUL  AB
        ADD  A,R2
        MOV  R2,A
        RET
        ;
CLR_DATA: MOV  DPTR,#0
        MOV  R2,#7FH
        MOV  R3,#0E0H
CLR_DATA1: MOV  A,0FFH
        MOVX @DPTR,A
        PUSH DPL
        CLR  C
        MOV  A,DPL
        SUBB A,R3
        MOV  DPL,A
        MOV  A,DPH
        SUBB A,R2
        ORL  A,DPL
        POP  DPL
        INC  DPTR
        JNZ  CLR_DATA1
        RET
        ;
SHF_LFT:  MOV  DPTR,#COMMAND
        MOV  A,#10H
        MOVX @DPTR,A
        RET
        ;
```

```
SHF_RHT:    MOV   DPTR,#COMMAND
            MOV   A,#14H
            MOVX  @DPTR,A
            RET
            ;
CLR_LCD:    MOV   A,#0
            LCALL GOTOXY
            MOV   R0,#20D
CLR_LCD1:   MOV   A," "
            LCALL WRITE
            DJNZ  R0,CLR_LCD1
            MOV   A,#40H
            LCALL GOTOXY
            MOV   R0,#20D
CLR_LCD2:   MOV   A," "
            LCALL WRITE
            DJNZ  R0,CLR_LCD2
            RET

            END
```

หนังสืออ้างอิง

1. ไมโครคอนโทรลเลอร์ , พิพัฒน์ เลาหสงคราม
2. การใช้งานไมโครคอนโทรลเลอร์ตระกูล 8051 , สุนทร วิทูสุรพจน์
3. อิเลคทรอนิคส์อุตสาหกรรม , ยืน ภู่วรวรรณ
4. Intdl , Microcontorller Hanbook , Intel Coporation