การออกแบบระบบเครื่องเล่นสำหรับคอมพิวเตอร์

DESIGN OF COMPUTER ENTERTAINMENT SYSTEM

โดย

นาย ชัยณรงค์ จันทร์อินทร์

นาย เมธี วิริยะวีรวรรณ

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาหลักสูตรปริญญาอุตสาหกรรมศาสตร์บัณฑิต

สาขาวิชาเทคโนโลยีโทรคมนาคม

ภาควิชาเทคนิคอุตสาหกรรม คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2544

หัวข้อปริญญานิพนธ์	การออกแบบระบบเครื่องเล่นสำหรับคอมพิวเตอร์
ชื่อนักศึกษา	นายชัยณรงค์ จันทร์อินทร์ รหัสประจำตัว 43015808
	นายเมธี วิริยะวีรวรรณ รหัสประจำตัว 43015833
อาจารย์ที่ปรึกษา	ผศ.ดร.อรรถสิทธิ์ หล่าสกุล
ระดับการศึกษา	ปริญญาอุตสาหกรรมศาสตร์บัณฑิต
	สาขาวิชาเทคโนโลยีโทรคมนาคม
ภาควิชา	เทคนิคอุตสาหกรรม
ปีการศึกษา	2544

คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง อนุมัติให้นับปริญญานิพนธ์ฉบับนี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรอุตสาหกรรมศาสตร์บัณฑิต

.....หัวหน้าภาควิชาเทคนิคอุตสาหกรรม

(ผศ.อุทัย ศรีธีระวิโรจน์)

คณะกรรมการตรวจสอบปริญญานิพนธ์

.....(อาจารย์ที่ปรึกษา)

(ผศ.ดร.อรรถสิทธิ์ หล่าสกุล)

.....(กรรมการ)

(.....)

.....(กรรมการ)

(.....)

.....(กรรมการ)

(.....)

.....(กรรมการ)

(.....)

.....(กรรมการ)

(.....)

Thesis Title	DESIGN OF COMPUTER ENTERTAINMENT SYSTEM
Student	Mr. Chainarong Janin No.43015808
	Mr. Mathee Wiriyaweerawan No.43015833
Thesis Advisor	Asst.Prof.Dr.Attasit Lhasakul
Level of Study	Bachelor of Industrial Technology
	Telecommunition
Department	Industrial Technology
Academic Year	2001

Accept by the Faculty of Engineering, King Mongkut's Institute of Technology Ladkrabang in partial fulfillment of the requirements for the bachelor's degree

.....Chairman

(Asst.Prof.U-thai Sritheeravirojana)

Project Report Committee

.....Advisor

(Asst.Prof.Dr.Attasit Lhasakul)

.....Member

(.....)

.....Member

(.....)

.....Member

(.....)

.....Member

(.....)

.....Member

(.....)

หัวข้อปริญญานิพนธ์ การออกแบบระบบเครื่องเล่นสำหรับคอมพิวเตอร์

ชื่อนักศึกษา นายชัยณรงค์ จันทร์อินทร์ รหัสประจำตัว 43015808
นายเมธี วิริยะวีรวรรณ รหัสประจำตัว 43015833

อาจารย์ที่ปรึกษา ผศ.ดร.อรรถสิทธิ์ หล่าสกุล

ระดับการศึกษา ปริญญาอุตสาหกรรมศาสตร์บัณฑิต
สาขาวิชาเทคโนโลยีโทรคมนาคม

ภาควิชา เทคนิคอุตสาหกรรม

ปีการศึกษา 2544

บทคัดย่อ

ในงานด้านอุตสาหกรรมความบันเทิงที่ใช้ระบบคอมพิวเตอร์และอิเล็กทรอนิกส์เข้ามาใช้งานนั้นได้มีการพัฒนาเป็นอย่างกว้างขวาง ในต่างประเทศได้มีบริษัทมากมายได้ใช้งบประมาณมหาศาลเพื่อพัฒนาชิ้นงานเพื่อออกสู่ตลาดในการแข่งขันกับบริษัทอื่นซึ่งมีมากมาย จึงนับได้ว่าเป็นอุตสาหกรรมที่หน้าสนใจเป็นอย่างมาก ในงานปริญญานิพนธ์ฉบับนี้จะได้นำเสนอถึงการนำเอาเทคนิคต่าง ๆ ทั้งด้าน อิเล็กทรอนิกส์, คอมพิวเตอร์ และอื่น ๆ มาประยุกต์ใช้งานในการสร้างเครื่องเล่นหุ่นยนต์ ซึ่งสามารถใช้งานได้ง่ายร่วมกับระบบคอมพิวเตอร์ที่มีใช้กันทั่วไป ในการสร้างนั้นได้ออกแบบให้เครื่องเล่นนี้มีความแปลกใหม่ไม่เหมือนเครื่องเล่นทั่วๆไป และพยายามสร้างเครื่องเล่นสนุกสนานน่าเล่นมากที่สุดที่น่าจะทำได้

Thesis Title	DESIGN OF COMPUTER ENTERTAINMENT SYSTEM
Student	Mr. Chainarong Janin No.43015808
	Mr. Mathee Wiriyaweerawan No.43015833
Thesis Advisor	Asst.Prof.Dr. Attasit Lhasakul
Level of Study	Bachelor of Industrial Technology
	Telecommunition
Department	Industrial Technology
Academic Year	2001

ABSTRACT

Now a day an application of computer has been connected to many field of industrial. The entertainment industrial also took the advance of them to produces many kinds of toy and equipments. In this project, proposed the simple robotics controller system that can be used together with IBM PC computer. Many topics that important to improvement the system will be investigated. Then the final high quality project should be achieved.

กิตติกรรมประกาศ

โครงงานปริญญานิพนธ์นี้ ได้สำเร็จลุล่วงได้ด้วยดี เพราะได้รับการช่วยเหลือให้คำแนะนำโครงงาน ทั้งการแก้ไขปัญหา แนวทางการทำงาน ตลอดจนกำลังใจจาก อาจารย์ที่ปรึกษาโครงงาน ผศ.ดร.อรรถสิทธิ์ หล่าสกุล สำหรับคำแนะนำและเครื่องมือต่างๆที่ใช้ในโครงงาน ซึ่งเป็นส่วนหนึ่งของโครงงาน นักศึกษาผู้จัดทำโครงงานนี้ มีความรู้สึกยินดีและขอบพระคุณสำหรับความรู้ความเอาใจใส่ที่ได้รับจากอาจารย์ที่ปรึกษา และคณาจารย์ทุกท่านประจำภาควิชาเทคนิคอุตสาหกรรม คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบังขอกราบพระคุณเป็นอย่างสูง

ขอบคุณพี่แปะ รุ่นพี่วิศวกรรมคอมพิวเตอร์ สำหรับคำแนะนำ และคำปรึกษา เกี่ยวกับ MCS-51 และข้อมูลต่างๆในการทำโครงงาน ขอบคุณ พี่วัช สำหรับคำแนะนำ และขอบคุณสำหรับเว็บไซต์ที่ซึ่งเป็นแหล่งข้อมูลของการทำโครงงานนี้ โดยไม่คิดมูลค่าใดๆ

สุดท้ายนี้ ที่สำคัญที่สุดขอกราบขอบพระคุณ คุณพ่อ คุณแม่ ที่เคารพ สำหรับแรงกายแรงใจ ที่มอบให้นศ. ขออุทิศคุณประโยชน์จากความรู้ทั้งหมดในโครงงานนี้ให้แด่ท่าน ที่ซึ่งเป็นกำลังใจให้จนกระทั่งโครงงานของปริญญานิพนธ์นี้สำเร็จลงได้ด้วยดี

นายชัยณรงค์ จันทร์อินทร์

นายเมธี วิริยะวีรวรรณ

สารบัญ

สารบัญรูป

สารบัญตาราง

บทที่ 1
บทนำ

1.1 วัตถุประสงค์

ในปัจจุบันนี้เทคโนโลยีทางด้านอิเล็กทรอนิกส์และคอมพิวเตอร์ซึ่งมีความเจริญขึ้นก้าวหน้าอย่างรวดเร็วมาก จึงเกิดแนวความคิดที่จะนำเทคโนโลยีในปัจจุบัน นำมาประยุกต์ใช้ในการสร้างเครื่องมือ และเครื่องเล่นคอมพิวเตอร์อิเล็กทรอนิกส์ ซึ่งเป็นแนวความคิดใหม่ของเครื่องเล่นในปัจจุบันนี้ และเพื่อให้เกิดแนวทางใหม่ๆของเครื่องเล่นและเป็นประโยชน์ต่อวงการอุตสาหกรรมของเครื่องเล่น

1.2 แนวความคิดและที่มาของโครงงาน

เครื่องเล่นเกมส์มีการพัฒนามานานแล้วตั้งแต่รุ่น Hatari , Nintendo ซึ่งในปัจจุบันนี้มีเครื่องเล่นเกมส์ต่างๆเช่นเครื่องเล่น Playstation, Playstation2, Dreamcast , และ ที่จะมาแรงของบริษัทไมโครซอฟ คือ X-Box นอกเหนือนั้นยังมีเกมส์ที่เล่นในคอมพิวเตอร์อีก เครื่องเล่นเกมส์เหล่านี้จะประมวลผลออกมาเป็นภาพส่งมายัง TV หรือ จอคอมพิวเตอร์ได้เท่านั้น

ซึ่งงานโปรเจ็กนี้เป็นการดึงรูปแบบในเกมส์เกี่ยวกับไฟติ้ง จำลองตัวหุ่นนำมาสร้างเป็นรูปตัวจริงๆเพื่อนำมาทำเป็นตัวเครื่องเล่นที่บังคับหุ่นยนต์โบอทไฟเตอร์ ซึ่งเครื่องเล่นนี้ไม่ยึดติดกับเครื่องเล่นเกมส์ที่มีมุมมองแต่ในภาพแล้วจะเป็นมุมมองการเล่นที่มี *มิติใหม่* ให้ความรู้สึกเสมือนจริง เป็นแนวความคิดที่ยังไม่เคยมีใครทำมาก่อน เครื่องเล่นตัวนี้ใช้เทคโนลียีของอิเล็กทรอนิกส์และคอมพิวเตอร์มาประยุกต์เข้าด้วยกัน ใช้ต่อกับคอมพิวเตอร์ คิดว่าเครื่องเล่นที่มี *มิติใหม่* นี้จะทำให้เกิดเป็นประโยชน์ในวงการอุตสาหกรรมของเครื่องเล่น

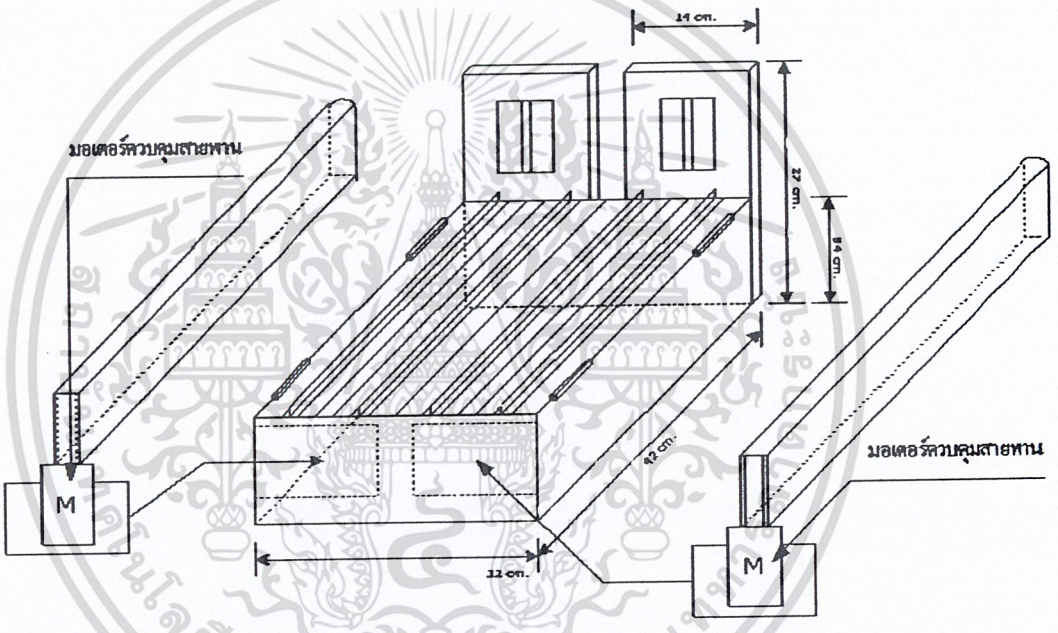
1.3 ขอบเขตและผลที่คาดว่าจะได้รับ

- ให้รูปแบบของเครื่องเล่นเกมส์มี มิติ รูปแบบใหม่ๆเกิดขึ้น
- ได้รับจุดความสนใจในด้านของรูปแบบเครื่องเล่นโฉมใหม่ ไม่ใช่เห็นได้แต่ในจอภาพ แต่ได้นำส่วนออกมาในรูปแบบตัวหุ่นยนต์จริงๆ
- ให้ได้รับความสนุกสนานและความบันเทิงจากเครื่องเล่นชิ้นนี้
- เป็นหุ่นยนต์ต้นแบบที่สามารถนำไปปรับปรุงหรือประยุกต์เป็นเครื่องเล่นที่มีการพัฒนาการที่ใส่เทคโนโลยีเพิ่มขึ้น เช่น การเคลื่อนที่ของหุ่นยนต์แบบไร้สายได้ทุกทิศทาง และมีเทคนิคการต่อสู้กัน และลูกเล่นของหุ่นยนต์ได้มากขึ้น
- หวังว่าจะเป็นรูปแบบแนวทางเลือกใหม่ทางด้านเทคโนโลยีอุตสาหกรรมเครื่องเล่นของคนไทยประดิษฐ์ขึ้นเอง

บทที่ 2
โครงสร้างและวงจรใช้งาน

2.1 การออกแบบฐานรองรับหุ่นโรบอต

เนื่องจากทำหุ่นไฟท์เตอร์นั้นต้องกำหนดขนาดและระยะห่างของหุ่นในการโจมตีและการเคลื่อนที่ในทิศทางที่จำกัดของหุ่น จึงได้ออกแบบให้ฐานรองรับหุ่นกว้างและยาวพอเหมาะกับการเคลื่อนที่ของหุ่น

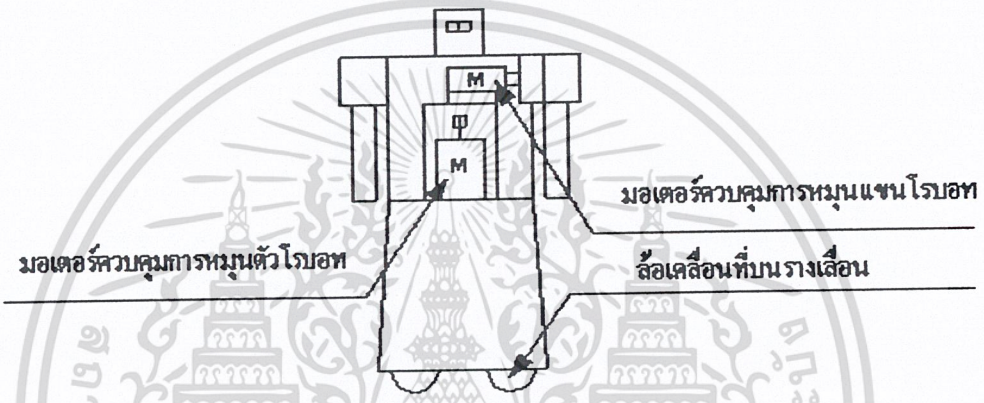


รูปที่ 2-1 ภาพฐานรองรับของตัวหุ่นยนต์และมอเตอร์ที่ต่อกับสายพานใช้ในการเคลื่อนที่ของหุ่นยนต์

ตัวฐานนั้นได้ใช้ไม้ในการสร้างเพื่อง่ายต่อการเปลี่ยนแปลงรูปเพราะไม่ได้มีต้นแบบที่แน่นอนและราคาประหยัด พร้อมทั้งออกแบบให้ไม่มีการชนกันของหุ่น ซึ่งการออกแบบนี้ได้มีการทดลองและวัดระยะจากขนาดของตัวหุ่นจริง ทำให้ได้ขนาดที่ต้องการ และที่ฐานยังทำการติดตั้ง DC มอเตอร์เพื่อใช้ขับสายพานที่ใช้ในการทำให้หุ่นเคลื่อนที่ในทิศทางได้ 2 ทิศทาง คือซ้ายและขวา การติดตั้งรางเลื่อนที่ฐานและลูกปืนที่ตัวหุ่นไว้สำหรับวางตัวหุ่นเพื่อให้หุ่นเคลื่อนที่ได้อย่างคล่องตัว

2.2 การออกแบบตัวหุ่น

รูปแบบหุ่นจากที่ได้ไปหามีรูปแบบและขนาดต่างๆกัน ซึ่งที่ได้นำมาลองทำเป็นหุ่น ซึ่งหาจนได้โครงสร้างหุ่นมีขนาดได้ตรงตามต้องการ ในรูปแบบการโจมตีนั้นผลการกระทบกระเทือน ตัวหุ่นอาจทำให้อุปกรณ์เกิดการเสียหายได้ จึงออกแบบส่วนข้อมือของหุ่นให้มีความยืดหยุ่นได้ โดยนำสปริงมาติดไว้ที่ข้อมือเพื่อลดผลการกระทบโดยตรงได้

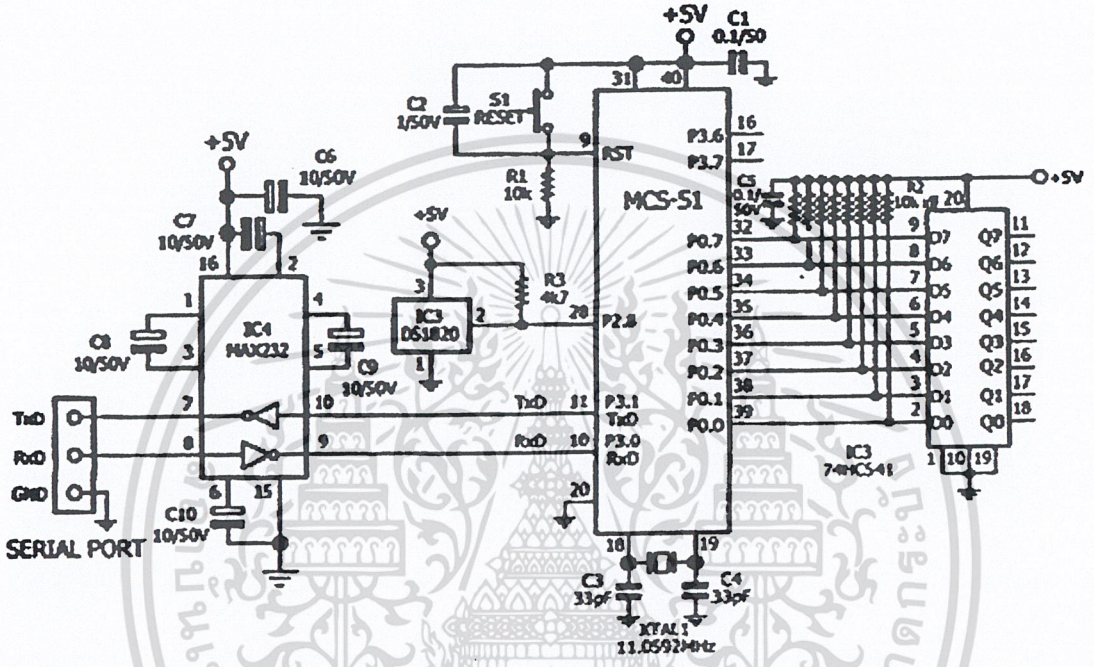


รูปที่ 2-2 ภาพโครงสร้างของตัวหุ่นยนต์ และมอเตอร์ที่ติดภายใน

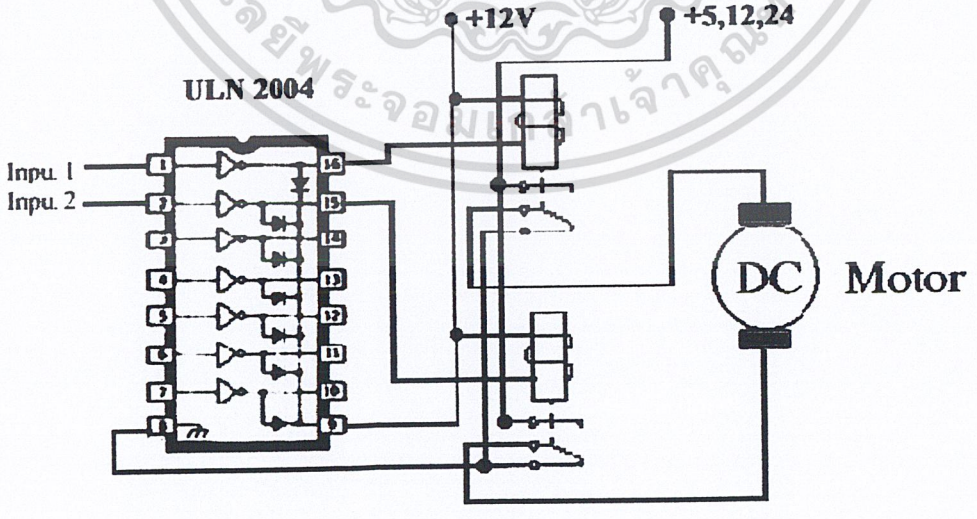
ที่ตัวหุ่นได้ทำการติดตั้ง DC มอเตอร์อยู่สองตัวเพื่อใช้ในการเคลื่อนไหว โดยตัวหนึ่งติดไว้ที่ลำตัวเพื่อทำหน้าที่ในการหมุนตัวและอีกตัวหนึ่งติดไว้ที่แขนขวาของหุ่นเพื่อใช้ในการโจมตี ส่วนล่างของตัวหุ่นได้ติดล้อเลื่อนไว้สำหรับวางบนรางเลื่อนและลำตัวได้มีการติดสวิชท์เพื่อรับสัญญาณเมื่อมีการถูกโจมตี โดยการประมวลผลในการรับรู้ว่าถูกโจมตีจะใช้ MCS-51

2.3 วงจร MCS-51 ต่อใช้งานกับ Serial port

รูปแบบการต่อวงจรของพอร์ตอนุกรมจากเครื่องคอมพิวเตอร์ กับ MCS-51 โดยผ่านไอซีแม็ก 232 เป็นตัวคอนเวอร์เตอร์ พอร์ตของ MCS-51 จะมีไอซี 74HC541 ทำหน้าที่เป็นบัฟเฟอร์ต่อเพื่อนำไปใช้งาน



รูปที่ 2-3 การสื่อสารข้อมูลผ่านพอร์ตอนุกรมของไมโครคอนโทรลเลอร์ MCS-51



รูปที่ 2-4 ภาค Driver motor โดยใช้ ULN 2004 เป็นตัวขับดีเลย์เพื่อควบคุมมอเตอร์

บทที่ 3
ทฤษฎีและหลักการ

3.1 ไมโครคอนโทรลเลอร์ตระกูล MCS-51

ไมโครคอนโทรลเลอร์ในตระกูล MCS-51 เป็นไมโครคอนโทรลเลอร์ขนาด 8 บิทที่มีอุปกรณ์สนับสนุนประกอบอยู่ภายใน หลายอย่างได้แก่ หน่วยความจำสำหรับเก็บข้อมูล หน่วยความจำสำหรับเก็บโปรแกรม ตัวตั้งเวลา/ตัวนับเวลา อุปกรณ์รับส่งข้อมูลแบบอนุกรม เนื่องจากโครงสร้างของไมโครคอนโทรลเลอร์มีอุปกรณ์สนับสนุนประกอบอยู่ภายในนี้เอง ทำให้การใช้งานง่ายขึ้นและมีประสิทธิภาพมากขึ้นโดยไม่ต้องมีการเชื่อมต่อร่วมกับอุปกรณ์อื่นเพิ่มเติมมาก เหมือนไมโครโปรเซสเซอร์ทั่วไป นอกจากนี้หากเราต้องการใช้งานไมโครคอนโทรลเลอร์ร่วมกับอุปกรณ์อื่นเพิ่มเติม เช่น ไอซี 8255 หรือ หน่วยความจำภายนอก เรายังสามารถนำมาเชื่อมต่อเพิ่มเติมเข้ากับไมโครคอนโทรลเลอร์ได้อีกด้วย

Device Name	EPROM	ROM Byte	RAM Byte	16 Bit Timer/Counter	Interupt
8031	-	-	128X8	2	5
8031AH	-	-	128X8	2	5
8031BH	-	-	128X8	2	5
8032AH	-	-	256X8	3	6
8051	-	4KX8	128X8	2	5
8051AH	-	4KX8	128X8	2	5
8051BH	-	4KX8	128X8	2	5
8052AH	-	8KX8	256X8	3	6
8751H	4KX8	-	128X8	2	5
8752H	8KX8	-	256X8	3	6

ตารางที่ 3-1 ไมโครคอนโทรลเลอร์ตระกูล MCS-51 มีให้เลือกได้หลายเบอร์ดังแสดงในตาราง

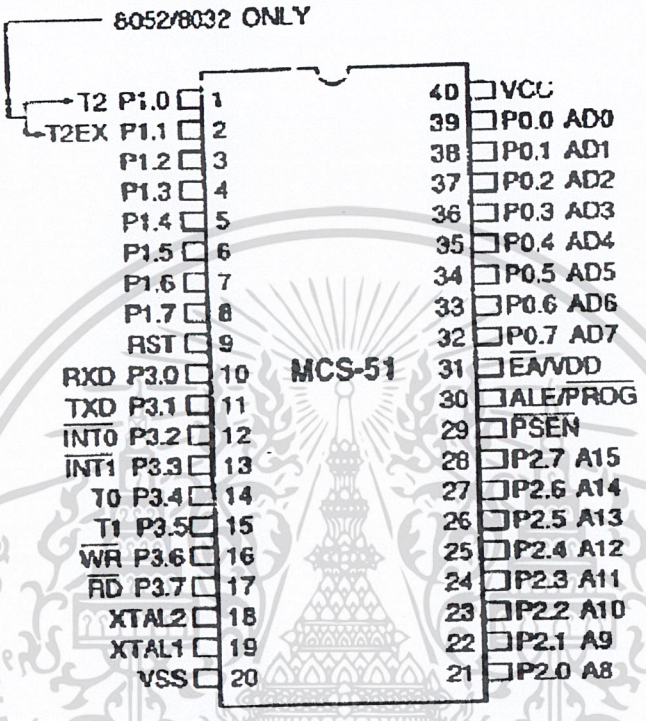
3.1.1 คุณสมบัติทั่วไปของไมโครคอนโทรลเลอร์ MCS-51

คุณสมบัติทั่วไปที่สำคัญของไมโครคอนโทรลเลอร์ตระกูล MCS-51 มีดังนี้

- เป็นไมโครคอนโทรลเลอร์ขนาด 8 บิต
- มีวงจรออสซิลเลเตอร์และวงจรผลิตสัญญาณนาฬิกาภายในไอซี
- มีขาสัญญาณอินพุตเอาต์พุตจำนวน 32 บิต
- สามารถเชื่อมต่อหน่วยความจำข้อมูลภายนอก (external data memory) โดยอ้างตำแหน่งแอดเดรสได้ถึง 64 K
- สามารถเชื่อมต่อหน่วยความจำโปรแกรมภายนอก (external program memory) โดยอ้างตำแหน่งแอดเดรสได้ถึง 64 K
- มีหน่วยความจำโปรแกรมภายในตัว (on – chip pro-gram memory) ขนาด 4 K โดยเฉพาะเบอร์ 8052 จะมีหน่วยความจำในส่วนนี้ถึง 8 K สำหรับเบอร์ 8031 และ M8032 จะไม่มีหน่วยความจำในส่วนนี้
- มีหน่วยความจำข้อมูลภายในตัว (on-chip data memory) ขนาด 128 ไบต์ โดยเฉพาะเบอร์ 8032 และ 8052 จะมีหน่วยความจำในส่วนนี้ถึง 256 ไบต์
- หน่วยความจำข้อมูลภายในบางส่วนสามารถเข้าถึงข้อมูลระดับบิตได้ด้วย ทำให้การควบคุมหรือการตรวจสอบสถานะบิตทำได้ง่าย ส่งผลให้การเขียนโปรแกรมทำได้ง่ายมากขึ้น
- มีไทเมอร์/เคาน์เตอร์ (timer/counters) ขนาด 16 บิต จำนวน 2 ตัว โดยเฉพาะเบอร์ 8032 หรือ 8052 จะมีไทเมอร์/เคาน์เตอร์จำนวน 3 ตัว
- การอินเตอร์รัปต์สามารถทำได้จาก 5 แหล่งกำเนิดโดยเฉพาะเบอร์ 8032 และ 8052 จะทำการอินเตอร์รัปต์ได้จาก 6 แหล่งกำเนิด โดยการอินเตอร์รัปต์ยังสามารถจัดระดับความสำคัญได้เป็น 2 ระดับ
- มีพอร์ตสื่อสารอนุกรมภายในตัวเอง ซึ่งทำงานเป็นแบบฟูลดูเพล็กซ์ (full duplex)
- มีคำสั่งในการคำนวณทางคณิตศาสตร์และทางตรรกศาสตร์
- คำสั่งโดยส่วนใหญ่ใช้เวลาทำงานเพียง 1 ไมโครวินาที เมื่อใช้คริสตอลความถี่ 12 เมกะเฮิรตซ์
- ต้องการแหล่งจ่ายไฟ 5 โวลต์ เพียงชุดเดียว

3.1.2 โครงสร้างภายนอกของ MCS-51

ไมโครคอนโทรลเลอร์ตระกูล MCS-51 ทุกเบอร์จะมีตำแหน่งขาพื้นฐานที่เหมือนกัน ดังแสดงในรูป *สำหรับหน้าที่การใช้งานของแต่ละขามีดังนี้*



รูปที่ 3-1 แสดงการจัดตำแหน่งขาต่าง ๆของไมโครคอนโทรลเลอร์ตระกูล MCS-51

- ขา V_{cc} เป็นขาป้อนแรงดันไฟเลี้ยง + 5 โวลต์
- ขา V_{ss} เป็นขากราวด์
- ขาพอร์ต 0 (Port 0) มี 8 ขา ได้แก่ขา $P_{00} - P_{0.7}$ เป็นขาพอร์ตอินพุตเอาต์พุตแบบ 2 ทิศทางสำหรับใช้งานทั่วไป โดยถ้าใช้งานเป็นอินพุตพอร์ตต้องทำการเขียนค่า 1 ไปยังแต่ละบิตของพอร์ต เพื่อกำหนดให้ขาพอร์ตเหล่านั้นอยู่ในสถานะปล่อยลอย ซึ่งในสถานะนี้เองที่สามารถนำมาใช้พอร์ตอินพุตอิมพีแดนซ์สูงได้ นอกจากพอร์ตนี้จะใช้งานเป็นพอร์ตอินพุตเอาต์พุตแล้วมันยังถูกใช้งานในการติดต่อกับหน่วยความจำภายนอกด้วย โดยทำหน้าที่ในการกำหนดตำแหน่งแอดเดรสไบต์ต่ำ ($A_0 - A_7$) ซึ่งจะใช้งานเป็นแบบมัลติเพล็กซ์กับการรับส่งข้อมูลขนาด 8 บิต ($D_0 - D_7$)
- ขาพอร์ต 1 (Port 1) มี 8 ขา ได้แก่ขา $P_{1.0} - P_{1.7}$ เป็นขาพอร์ตอินพุตเอาต์พุตแบบ 2 ทิศทาง สำหรับใช้งานทั่วไป โดยถ้าใช้งานเป็นอินพุตพอร์ตต้องทำการเขียนค่า 1 ไปยัง

แต่ละบิตของพอร์ต เพื่อกำหนดให้เป็นพอร์ตอินพุต นอกจากนี้สำหรับเบอร์ 8032 และ 8052 ขาพอร์ต $P_{1.0}$ และ $P_{1.1}$ จะถูกนำมาใช้งานเป็นขา T2 และ T2EX ตามลำดับด้วย

- ขาพอร์ต 2 (Port 2) มี 8 ขาได้แก่ขา $P_{2.0}$ - $P_{2.7}$ เป็นขาพอร์ตอินพุตเอาต์พุตแบบ 2 ทิศทางสำหรับใช้งานทั่วไป โดยถ้าใช้งานเป็นอินพุตพอร์ตต้องทำการเขียนค่า 1 ไปยังแต่ละบิตของพอร์ต เพื่อกำหนดให้เป็นพอร์ตอินพุต นอกจากพอร์ตนี้จะใช้งานเป็นพอร์ตอินพุตเอาต์พุตแล้วมันยังถูกใช้งานในการติดต่อกับหน่วยความจำภายนอกด้วย โดยทำหน้าที่ในการกำหนดตำแหน่งแอดเดรสไบต์สูง ($A_8 - A_{15}$)

- ขาพอร์ต 3 (Port 3) มี 8 ขาได้แก่ขา $P_{3.0} - P_{3.7}$ เป็นขาพอร์ตอินพุตเอาต์พุตแบบ 2 ทิศทางสำหรับใช้งานทั่วไปโดยถ้าใช้งานเป็นอินพุตพอร์ตต้องทำการเขียนค่า 1 ไปยังแต่ละบิตของพอร์ต เพื่อกำหนดให้เป็นพอร์ตอินพุต นอกจากพอร์ตนี้จะใช้งานเป็นพอร์ตอินพุตเอาต์พุตแล้วมันยังถูกใช้งานในหน้าที่พิเศษต่าง ๆ ดังแสดงในตารางที่ 2.2

- ขารีเซต (RST) ใช้สำหรับการรีเซตการทำงานของไมโครคอนโทรลเลอร์ โดยการรีเซตต้องคงสถานะเป็น 1 อย่างน้อยนาน 2 แมชีนไซเกิล ในขณะที่ออสซิลเลเตอร์ยังทำงานอยู่

- ขา ALE/$\overline{PROG}$ เป็นขาสัญญาณเพื่อทำหน้าที่ควบคุมการแลตช์ (latch) ค่าตำแหน่งแอดเดรสไบต์ต่ำ (Address Latch Enable) เมื่อต้องการติดต่อกับหน่วยความจำภายนอก นอกจากนี้ขานี้ยังทำหน้าที่เป็นอินพุตรับพัลส์ในการโปรแกรม (program pulse input) ในส่วนของหน่วยความจำ EPROM สำหรับไมโครคอนโทรลเลอร์ในตระกูล MCS-51 ที่มีหน่วยความจำโปรแกรมภายในเป็น EPROM

ขาพอร์ต	หน้าที่พิเศษ
P 3.0	RXD (serial input port)
P 3.1	TXD (serial output port)
P 3.2	INTO (external interrupt 0)
P 3.3	INT1 (external interrupt 1)
P 3.4	TO (Timer 0 external input)
P 3.5	T1 (Timer 1 external input)
P 3.6	WR (external data memory write strobe)
P 3.7	RD (external data menory read strobe)

ตารางที่ 3.2 แสดงหน้าที่พิเศษของแต่ละขาของพอร์ต P3

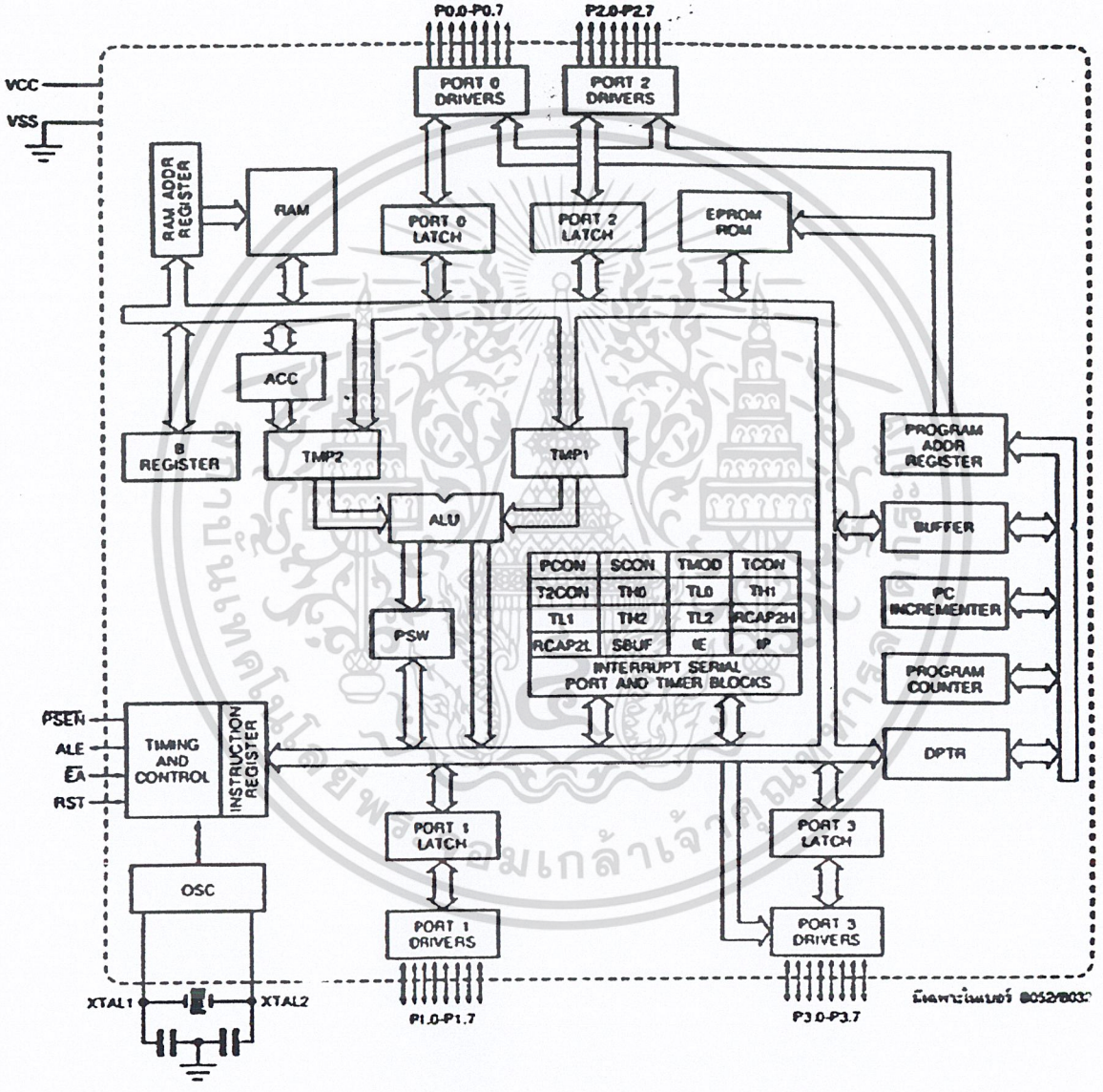
- ขา $\overline{\text{PSEN}}$ (Program Store Enable) ทำหน้าที่เป็นสัญญาณสโตรบเพื่ออ่านคำสั่งจากหน่วยความจำโปรแกรมภายนอก เมื่อไมโครคอนโทรลเลอร์ประมวลผลคำสั่งจากหน่วยความจำภายนอก ขานี้จะส่งสัญญาณสโตรบจำนวน 2 ครั้งในแต่ละแมชีนไซเกิล แต่ในขณะที่ติดต่อกับหน่วยความจำข้อมูลภายนอกจะไม่มีการส่งสัญญาณสโตรบแต่อย่างใด

- ขา $\overline{\text{EA}}$/VPP (External Access enable/VPP) เป็นขาสำหรับการเลือกใช้หน่วยความจำโปรแกรมจากภายในหรือจากภายนอก โดยถ้ามีสถานะเป็น 0 จะหมายถึงให้ไมโครคอนโทรลเลอร์รับคำสั่งจากหน่วยความจำภายนอกที่ตำแหน่งแอดเดรส 0-OFFFH (0-1FFFH ถ้าเป็นเบอร์ 8052) อย่างไรก็ตามถ้าบิตป้องกัน (security bit) ในหน่วยความจำ EPROM ถูกโปรแกรมไว้ ไมโครคอนโทรลเลอร์จะไม่รับคำสั่งจากหน่วยความจำภายนอกเลย นอกจากนี้ขานี้ยังทำหน้าที่รับแรงดันไฟสำหรับการโปรแกรม (V_{pp}) ขนาด 21 โวลต์ เพื่อใช้ในระหว่างการโปรแกรม EPROM

- ขา $XTAL_1$ และขา $XTAL_2$ เป็นขาอินพุตและเอาต์พุตของวงจรอินเวอร์ติ้งออสซิลเลเตอร์แอมปลิไฟเออร์ (inverting oscillator amplifier) สำหรับใช้ต่อร่วมกับคริสตอลภายนอก

3.1.3 โครงสร้างภายในของ MCS-51

โครงสร้างภายในของไมโครคอนโทรลเลอร์ตระกูล MCS-51 แสดงดังในรูปโดยส่วนที่มีเครื่องหมายดอกจัน (*) จะมีเฉพาะในเบอร์ 8032 และ 8052 เท่านั้น



รูปที่ 3-2 แสดงโครงสร้างภายในของไมโครคอนโทรลเลอร์ตระกูล MCS-51

3.1.4 การจัดหน่วยความจำ

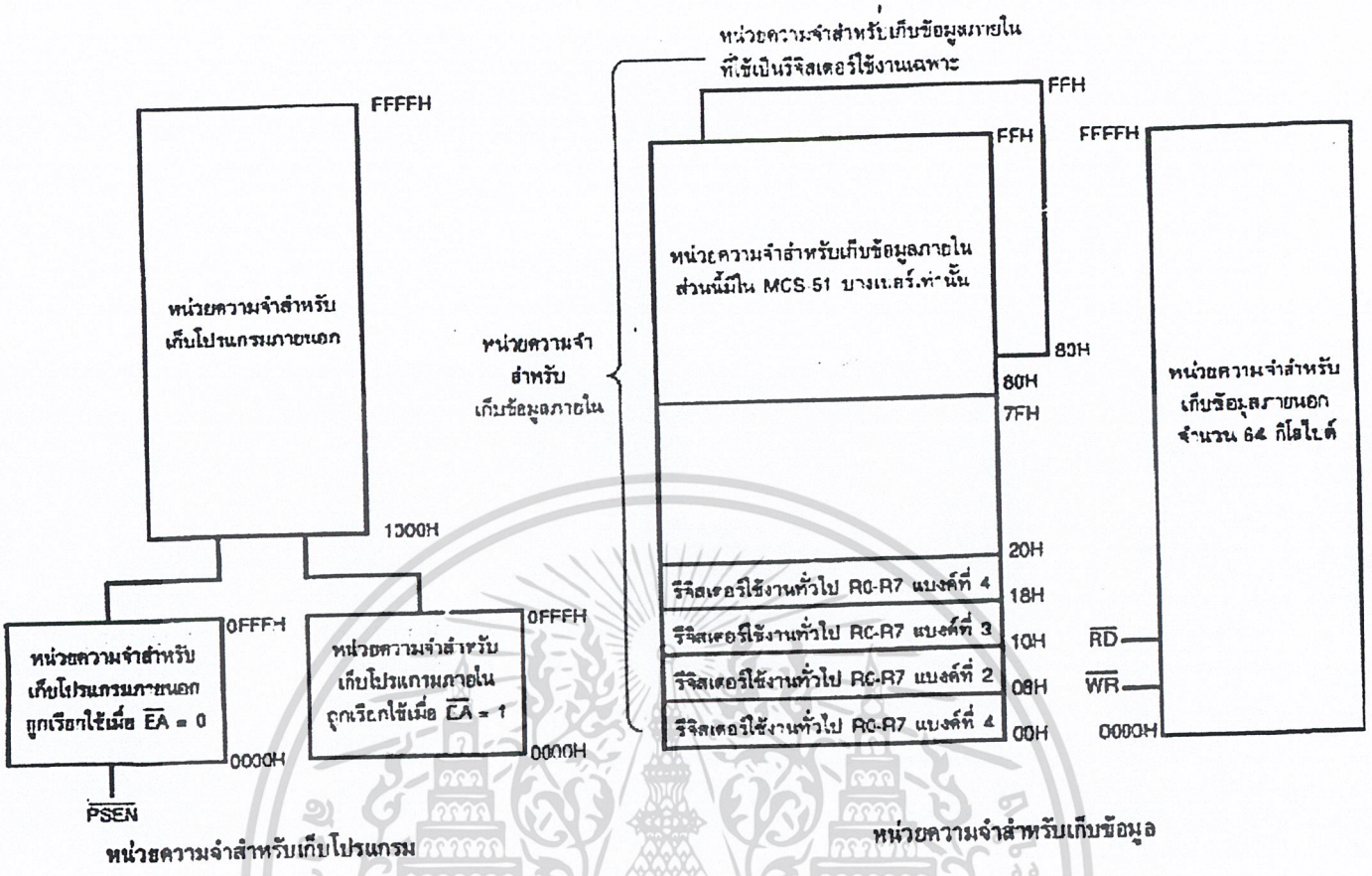
ในไมโครคอนโทรลเลอร์ตระกูล MCS-51 แบ่งชนิดหรือหน้าที่ของหน่วยความจำออกเป็น 2 ส่วนคือหน่วยความจำโปรแกรม (program memory) และหน่วยความจำข้อมูล (data memory)

หน่วยความจำโปรแกรมจะใช้สำหรับเก็บโปรแกรมควบคุมการทำงานของไมโครคอนโทรลเลอร์ซึ่งบางเบอร์จะมีหน่วยความจำในส่วนนี้อยู่ภายในตัว โดยอาจจะมีขนาดไม่เท่ากันหรือเป็นหน่วยความจำต่างชนิดกัน เช่น บางเบอร์เป็น ROM และบางเบอร์อาจเป็น EPROM และบางเบอร์อาจไม่มีหน่วยความจำในส่วนนี้เลย โปรแกรมการทำงานจะถูกเก็บไว้ยังหน่วยความจำโปรแกรมภายนอกทั้งหมด

สำหรับหน่วยความจำข้อมูลจะใช้สำหรับเก็บข้อมูลหรือค่าตัวแปรต่าง ๆ จากการทำงานของโปรแกรม ซึ่งใน MCS-51 ทุกเบอร์จะมีหน่วยความจำในส่วนนี้อยู่จำนวนหนึ่ง แต่อาจมีขนาดมากน้อยต่างกันไปในแต่ละเบอร์สำหรับการจัดโครงสร้างของหน่วยความจำทั้งในส่วนของหน่วยความจำโปรแกรมและหน่วยความจำข้อมูลแสดงไว้ดังในรูปที่ 3-2

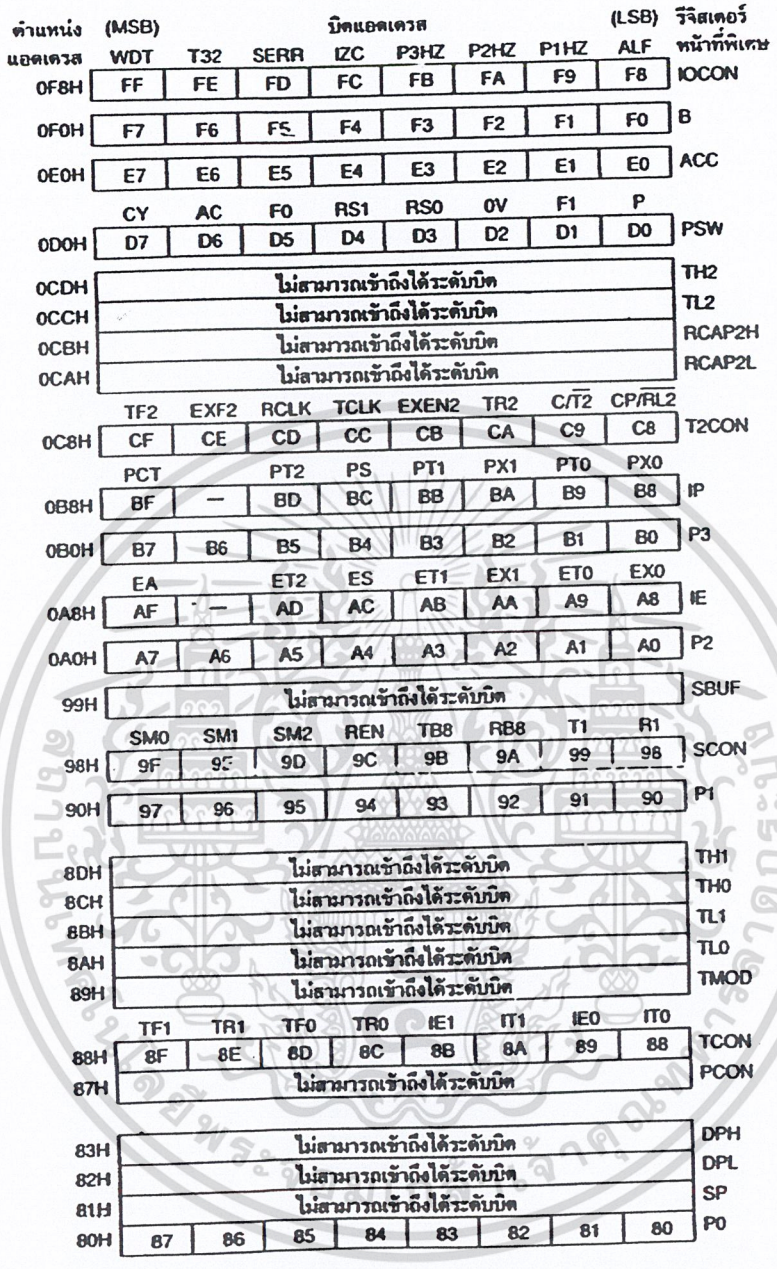
หน่วยความจำโปรแกรม

หน่วยความจำโปรแกรมสามารถแบ่งออกได้เป็น 2 ส่วนคือ หน่วยความจำโปรแกรมภายในและหน่วยความจำโปรแกรมภายนอก หน่วยความจำโปรแกรมภายในจะถูกเลือกใช้งานถ้าขาสัญญาณ EA มีค่าเป็น 1 โดยจะถูกใช้งานในช่วงแอดเดรส 0-OFFFH (หรือช่วงแอดเดรส 0-1FFFH ในเบอร์ 8052) นอกเหนือจากช่วงแอดเดรสนี้จะใช้หน่วยความจำโปรแกรมภายนอกทั้งหมด ในกรณีตรงกันข้ามถ้าขาสัญญาณ EA มีค่าเป็น 0 ในช่วงแอดเดรส 0-0FFFH (หรือช่วงแอดเดรส 0-1FFFH ในเบอร์ 8052) จะถูกใช้จากหน่วย



รูป 3-3 แสดงการจัดโครงสร้างของหน่วยความจำทั้งในส่วนของหน่วยความจำโปรแกรม และหน่วยความจำข้อมูล

ความจำภายนอก หรือกล่าวได้ว่าถ้าขาสัญญาณ $\overline{EA}$ มีค่าเป็น 0 จะเป็นการเลือกใช้หน่วยความจำโปรแกรมภายนอกทั้งหมดตลอดช่วงแอดเดรส



รูปที่ 3-4 แสดงการจัดหน่วยความจำและตำแหน่งของรีจิสเตอร์หน้าที่พิเศษต่าง ๆ

หน่วยความจำข้อมูล

หน่วยความจำข้อมูลสามารถแบ่งออกได้เป็น 2 ส่วน คือ หน่วยความจำข้อมูลภายในและหน่วยความจำข้อมูลภายนอก สำหรับหน่วยความจำข้อมูลภายในยังแบ่งออกได้เป็น 2 ส่วนย่อย คือ ส่วนที่ใช้เก็บข้อมูลทั่วไปและส่วนที่ใช้เป็นรีจิสเตอร์หน้าที่พิเศษหรือ SFR (Special Function Register) โดยส่วนที่ใช้เก็บข้อมูลทั่วไปจะถูกใช้สำหรับเก็บข้อมูลหรือค่าตัวแปรต่าง ๆ

จากการทำงานของโปรแกรม ส่วนรีจิสเตอร์หน้าที่พิเศษจะถูกใช้งานเป็นรีจิสเตอร์ควบคุมการทำงานและบอกสถานะการทำงานของไมโครคอนโทรลเลอร์

ไมโครคอนโทรลเลอร์ตระกูล MCS-51 ทุกเบอร์จะมีหน่วยความจำข้อมูลภายในขนาด 128 ไบต์เป็นอย่างน้อย และบางเบอร์อาจมีถึงขนาด 256 ไบต์

รีจิสเตอร์หน้าที่พิเศษ (SFR)

รีจิสเตอร์หน้าที่พิเศษมีบทบาทอย่างมากในการควบคุมการทำงานของไมโครคอนโทรลเลอร์และทำให้การเขียนโปรแกรมสามารถทำได้สะดวกมากขึ้น รีจิสเตอร์หน้าที่พิเศษทำหน้าที่สำคัญคือควบคุมการทำงานในส่วนต่าง ๆ ภายในไมโครคอนโทรลเลอร์และทำหน้าที่แสดงสถานะการทำงาน ซึ่งในรีจิสเตอร์หน้าที่พิเศษบางตัวยังสามารถเข้าถึงได้ในระดับบิต (bit addressable) ด้วย ดังแสดงรูปการจัดหน่วยความจำและตำแหน่งของรีจิสเตอร์หน้าที่พิเศษต่าง ๆ

รีจิสเตอร์ใช้งานทั่วไป

รีจิสเตอร์ใช้งานทั่วไปมีไว้สำหรับให้ผู้เขียนโปรแกรมสามารถนำข้อมูลไปพักไว้ชั่วคราวหรือใช้งานทั่วไปได้ตามต้องการ ซึ่งรีจิสเตอร์ใช้งานทั่วไปนี้มีอยู่ด้วยกัน 8 ตัว คือรีจิสเตอร์ R0 – R7 โดยรีจิสเตอร์ทั้ง 8 ตัวถูกจัดให้อยู่รวมกันและมีให้เลือกใช้ถึง 4 แบงค์ (bank) นั่นคือมีรีจิสเตอร์ใช้งานทั่วไปถึง 32 ตัวให้ใช้งาน เพียงแต่การเลือกใช้รีจิสเตอร์ R0 – R7 ในแบงค์ใดแบงค์หนึ่งจะถูกกำหนดจากบิต RS0,RS1 ในรีจิสเตอร์หน้าที่พิเศษ PSW ดังนั้นการเลือกใช้จึงเลือกได้เพียงแบงค์เดียวในขณะใดขณะหนึ่ง อย่างไรก็ตามค่าข้อมูลที่เก็บไว้ในรีจิสเตอร์แบงค์ใดก็ตามที่มีชื่อเดียวกันแต่อยู่คนละแบงค์จะไม่มีผลซึ่งกันและกันเลย ทำให้ผู้เขียนโปรแกรมใช้งานรีจิสเตอร์ทั่วไปนี้ได้ทั้ง 32 ตัว อย่างเต็มที่และไม่ยุ่งยากในการเขียนโปรแกรม

```
; FILE XAMPLE01.A51
;
            ORG 4100H          ; start object code at 4100H
START       MOV DPTR, #text    ; DPTR points to text
            LCALL STXT         ; MONITOR Routine print text
            RET                ; return to MONITOR
;
text        DB          'The FIRST Program...', 13, 10, 0
;-------------------------------------------------------------
; MONITOR INTERFACE
;
ccSTXT      EQU     2          ; MONITOR command to send text
COMMAND EQU     030H           ; MONITOR command memory
                               ; location
MON         EQU     0200H      ; MONITOR entry address
;
STXT        MOV COMMAND, #ccSTXT ; MONITOR set command
            LJMP    MON        ; MONITOR call it!
;-------------------------------------------------------------
            END
```

รูปที่ 3-5 ตัวอย่างโปรแกรมภาษาแอสเซมบลีหรือซอร์สโค้ดไฟล์ที่เขียนขึ้นโดยใช้เวิร์ดโปรเซสเซอร์ทั่วไป

3.1.5 การเขียนโปรแกรมภาษาแอสเซมบลี

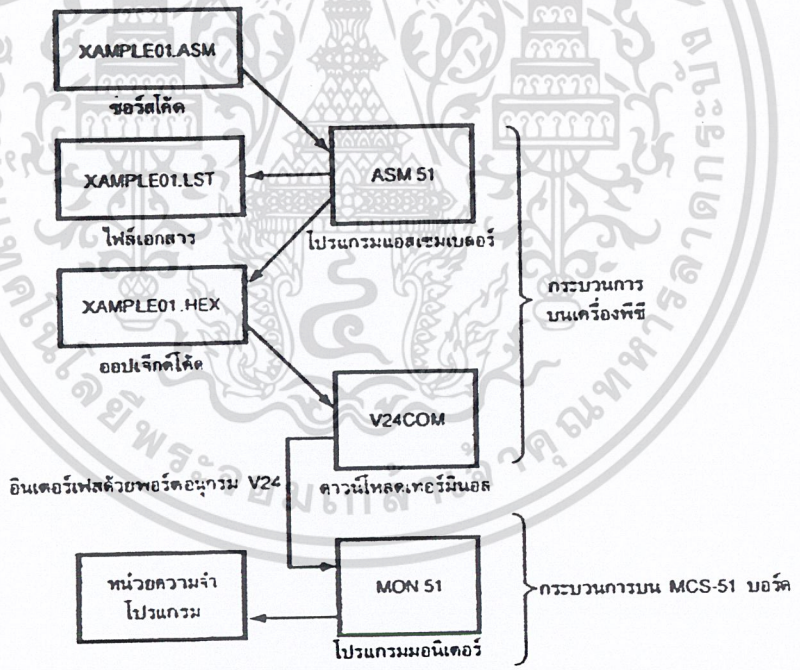
รูปที่ 2.5 เป็นตัวอย่างโปรแกรมภาษาแอสเซมบลีที่เขียนขึ้นภายใต้โปรแกรมเวิร์ดโปรเซสเซอร์ ซึ่งมีชื่อว่า XAMPLE0.1A51 (บรรจุอยู่ในดิสเก็ตต์แล้ว) โปรแกรมนี้เรียกว่าซอร์สโค้ดโปรแกรม (source code program) ในแต่ละบรรทัดจะบรรจุซึ่งสาระสำคัญต่าง ๆ ในรูปของคำสั่ง คำสั่งที่ถูกเขียนขึ้นเรียกว่านิวโมนิก (mnemonic) นอกจากในแต่ละบรรทัดประกอบด้วยคำสั่งไมโครคอนโทรลเลอร์แล้วยังมีส่วนของคำอธิบาย (comment) ด้วย คำอธิบายเหล่านี้มีประโยชน์อย่างยิ่งเพื่อให้ผู้อื่นทำความเข้าใจได้กับแต่ละคำสั่งที่เขียนขึ้นหรือแม้แต่ตัวผู้เขียนโปรแกรมเอง เมื่อต้องการพัฒนาโปรแกรมเพิ่มเติมจะได้ไม่สับสน

โปรแกรมแอสเซมบลียังไม่สามารถนำมาใช้งานกับไมโครคอนโทรลเลอร์ได้ทันที แต่ต้องนำมาทำการแปลงจากซอร์สโค้ดไฟล์เป็นภาษาเครื่อง (machine code) ทั้งคำสั่งต่าง ๆ ข้อมูลและการอ้างแอดเดรสทั้งหมดถูกแปลงไปเป็นโปรแกรมภาษาเครื่องที่ไมโครคอนโทรลเลอร์สามารถเข้าใจได้ หรือเรียกว่า ออบเจ็กต์โค้ด (object code) โดยใช้โปรแกรมแอสเซมเบลอร์ในที่นี้คือ EASM51.EXE มาใช้งาน หรืออาจกล่าวได้อีกอย่างหนึ่งคือโปรแกรมแอสเซมเบลอร์ทำการสร้าง

ไฟล์ใหม่ขึ้นมาอีกไฟล์หนึ่ง โดยการนำเอาออบเจ็กต์โค้ดมาแทนที่คำสั่งหรือนิวโมนิกที่เขียนขึ้น โดยไม่สนใจคำอธิบายต่าง ๆ ที่เขียนไว้หรือตัดส่วนนี้ออกไปไม่นำมาใช้งานเลย

เอาต์พุตไฟล์ที่ได้จากการแอสเซมเบลอร์จะมีนามสกุลเป็น .HEX ตัวอย่างเช่น XAMPLE01.HEX ไฟล์ที่ถูกแปลงเป็นภาษาเครื่องแล้วจะอยู่ในรูปของเลขฐานสิบหกจำนวน 2 หลัก เรียงตามลำดับคำสั่งต่าง ๆ ที่เขียนขึ้น ไฟล์นี้เองที่สามารถประมวลผลได้ทันที เมื่อมันถูกส่งผ่านไปเก็บไว้ที่หน่วยความจำบน MCS-51 บอร์ด หรือกล่าวได้ว่าเครื่องพีซีทำหน้าที่สร้างออบเจ็กต์โค้ดขึ้นมา และทำการส่งผ่านหรือดาวน์โหลดไปยัง MCS-51 บอร์ด โดยการติดต่อผ่านโปรแกรม V24COM เพื่อส่งข้อมูลผ่านพอร์ตสื่อสารอนุกรม

นอกจาก EASM51 จะทำการสร้างออปเจ็กต์โค้ดขึ้นมาไฟล์หนึ่งแล้วมันยังสร้างไฟล์เอกสาร (list file) ขึ้นมาชุดหนึ่งมีนามสกุล .LST ตัวอย่างเช่น XAMPLE01.LST ไฟล์



รูปที่ 3-6 แสดงกระบวนการต่าง ๆ ที่เกิดขึ้นจากซอร์สโค้ดที่เขียนบนเครื่องพีซีจนเป็นออปเจ็กต์โค้ดซึ่งถูกโหลดมาเก็บไว้ในหน่วยความจำโปรแกรมบน MCS-51 บอร์ด

เอกสารนี้สร้างขึ้นเพื่อรวบรวมและแสดงออปเจ็กต์โค้ดที่สร้างโดยโปรแกรมแอสเซมเบลอร์จากซอร์สโค้ดโปรแกรมและข้อมูลอื่น ๆ ที่สำคัญ ดังนั้นไฟล์เอกสารนี้เป็นประโยชน์อย่างยิ่ง เมื่อ

ต้องการศึกษาการทำงานของโปรแกรมและการทำงานภายในไมโครคอนโทรลเลอร์ ตลอดจนตรวจสอบโปรแกรมที่เขียนขึ้นเพื่อพัฒนาในครั้งต่อไป ถึงแม้ว่าไฟล์เอกสารนี้จะไม่สามารถนมาใช้งานกับไมโครคอนโทรลเลอร์แต่มันก็มีประโยชน์ดังได้กล่าวมาแล้ว รูปที่ 3-6 แสดงให้เห็นถึงกระบวนการในการแปลงโปรแกรมภาษาแอสเซมบลี จนกระทั่งเป็นโปรแกรมที่สามารถประมวลผลได้ทันทีกับ MCS-51 บอร์ด สำหรับตัวอย่างให้ผู้อ่านได้ทดลองจริงให้อ่านคำอธิบายในไฟล์ XAMPLE01.DOC

การใช้งานรีจิสเตอร์

โดยปกติไมโครคอนโทรลเลอร์ตระกูล MCS-51 จะทำการประมวลผลข้อมูลครั้งละ 1 ไบต์ ซึ่งกระทำกับรีจิสเตอร์ภายในโดยที่รีจิสเตอร์แต่ละตัวเก็บข้อมูลได้ขนาด 1 ไบต์เช่นกัน เช่น รีจิสเตอร์ A ซึ่งเป็นแอกคิวมูเลเตอร์ (accumulator) ทำหน้าที่เป็นรีจิสเตอร์กลางสำหรับการคำนวณทางคณิตศาสตร์หรือทางลอจิกของตัวกระทำ 2 ตัว ตัวอย่างเช่น ถ้าต้องการบวกค่า 10 กับข้อมูลตัวหนึ่ง ให้ทำการโหลดข้อมูลไปเก็บไว้ในรีจิสเตอร์ A ก่อน จากนั้นให้ใช้คำสั่งนำค่า 10 ไปบวกกับ A ผลที่ได้จากการบวกข้อมูลและค่า 10 จะถูกเก็บไว้ในรีจิสเตอร์ A นอกจากรีจิสเตอร์ A ทำการบวกด้วยการกำหนดค่าโดยตรงแล้ว มันยังทำการคำนวณร่วมกับรีจิสเตอร์ขนาด 8 บิตตัวอื่น ๆ ได้อีกด้วย

ทั้งไมโครโปรเซสเซอร์และไมโครคอนโทรลเลอร์จะมีรีจิสเตอร์สำหรับใช้งานในคำสั่งพิเศษโดยผู้เขียนโปรแกรมอาจกำหนดขึ้นได้เอง โดยที่กำหนดให้อยู่ในตำแหน่งแอดเดรสพิเศษ ในที่นี้มีค่ามากกว่า 07FH ขึ้นไป ตัวอย่างเช่นแอกคิวมูเลเตอร์ถูกกำหนดให้ใช้หน่วยความจำภายในที่ 0E0H รีจิสเตอร์เหล่านี้เรียกว่า รีจิสเตอร์หน้าที่พิเศษ (special function registers หรือ SFRs) จำนวนของรีจิสเตอร์พิเศษอาจจะมีไม่เท่ากันในไมโครคอนโทรลเลอร์แต่ละเบอร์ในตระกูล MCS-51 ขึ้นอยู่กับคำสั่งที่ทำการตั้งค่าไว้ เพราะรีจิสเตอร์พิเศษเหล่านี้ถูกรวมอยู่หรือใช้พื้นที่ในส่วนของหน่วยความจำภายในของไมโครคอนโทรลเลอร์ ซึ่งหน่วยความจำภายในหรือแรมภายในจะมีขนาดไม่เท่ากันในแต่ละเบอร์

นอกจากรีจิสเตอร์พิเศษหรือ SFR แล้วยังมีรีจิสเตอร์ สำหรับใช้งานทั่วไปอีก 8 ตัว คือรีจิสเตอร์ R_0 ถึง R_7 รีจิสเตอร์ทั้ง 8 ตัว ถูกบรรจุอยู่ภายในแรมภายในไมโครคอนโทรลเลอร์ในรูปของแบงค์ (bank) และใช้สำหรับเก็บข้อมูลชั่วคราวระหว่างการประมวลผลในที่นี้จะใช้รีจิสเตอร์เฉพาะแบงค์ศูนย์เท่านั้น และหลังจากรีเซตระบบทุกครั้งรีจิสเตอร์ที่แบงค์ศูนย์จะถูกเลือกโดยอัตโนมัติ

การจัดสรรหน่วยความจำบน MCS-51 บอร์ด

ไมโครคอนโทรลเลอร์แต่ละเบอร์ในตระกูล MCS-51 มีขนาดของหน่วยความจำไม่เท่ากัน ทำให้การจัดสรรพื้นที่ในหน่วยความจำสำหรับเก็บโปรแกรมและข้อมูลแตกต่างกัน

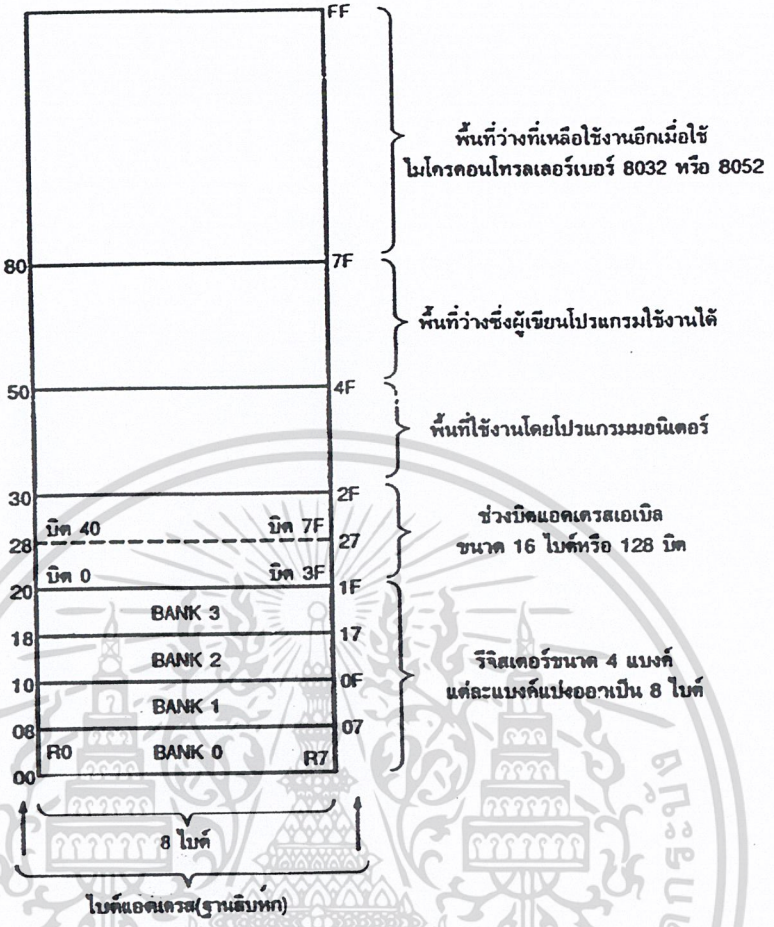
หน่วยความจำสำหรับเก็บโปรแกรม

หน่วยความจำสำหรับเก็บโปรแกรมสามารถขยายได้สูงถึง 64 กิโลไบต์ มีหน้าที่เก็บคำสั่งต่าง ๆ สำหรับไมโครคอนโทรลเลอร์ มันสามารถใช้เก็บตารางข้อมูลและค่าคงที่ได้ในการใช้งาน ในที่นี้จะใช้งานไมโครคอนโทรลเลอร์โดยใช้หน่วยความจำโปรแกรมภายนอกเท่านั้น ที่ขา 31 หรือค่า $\overline{EA}$ (external access enable) จึงถูกต่อลงกราวด์ไว้เพื่อกำหนดให้ไม่ใช้งานหน่วยความจำโปรแกรมภายในที่มีอยู่แล้ว และเมื่อไมโครคอนโทรลเลอร์ต้องการติดต่อกับหน่วยความจำโปรแกรมภายนอกมันจะส่งสัญญาณลอจิก Low ที่ขา 29 หรือขา $\overline{PSEN}$ ออกมา

หน่วยความจำโปรแกรมไม่จำเป็นเสมอไปว่าต้องเป็นรอมหรืออีพรอม เช่นเดียวกับตำแหน่งแอดเดรสที่ว่างแต่ละแอดเดรส อาจอยู่ในรูปของหน่วยความจำหรือเป็นตำแหน่งของพอร์ตอินพุตเอาต์พุตก็ได้ หน่วยความจำโปรแกรมในที่นี้ถูกแบ่งออกเป็น 2 ช่วงดังนี้ คือช่วงแอดเดรสต่ำ 00000H ถึง 04000H เป็นส่วนของอีพรอม IC5 และช่วงแอดเดรสจาก 04000H ถึง 08000H เป็นหน่วยความจำแรม IC6 ของระบบ คำสั่งต่าง ๆ จะถูกป้อนให้ไปเก็บไว้และทำการประมวลผลจากที่แรมนี้

หน่วยความจำสำหรับเก็บข้อมูล

ไมโครคอนโทรลเลอร์ตระกูล MCS-51 สามารถทำการอ่านและเขียนข้อมูลจากหน่วยความจำข้อมูลที่มีขนาดสูงสุดได้ 64 กิโลไบต์ หน่วยความจำในส่วนนี้ทำหน้าที่เก็บข้อมูลใช้งานจำนวนมากเป็นส่วนใหญ่ ซึ่งในบางครั้งอาจเรียกได้ว่าแรมบน MCS-51 บอร์ด หรือ IC6 เป็นผู้ทำหน้าที่นี้ หน่วยความจำข้อมูลบนบอร์ด MCS-51 กำหนดให้มีตำแหน่งใช้งานได้ตั้งแต่ 00000H ถึง 08000H ซึ่งตามที่กล่าวมาแล้วหน่วยความจำสำหรับเก็บโปรแกรมโดยใช้แรมถูกกำหนดให้เริ่มต้นที่ตำแหน่ง 04000H เป็นต้นไป นั่นคือโปรแกรมทดลองหรือโปรแกรมที่ดาวน์โหลดจากพิวเตอร์จะต้องเริ่มประมวลผลที่ตำแหน่ง 04000H ขึ้นไปเสมอ



รูปที่ 3-7 รูปแสดงการเก็บรีจิสเตอร์

การประยุกต์ใช้งานไมโครคอนโทรลเลอร์ส่วนใหญ่แล้วสามารถทำงานได้โดยมไต้องใช้หน่วยความจำข้อมูลมากเท่าใดนัก ทำให้เมื่อต้องการใช้หน่วยความจำก็อาศัยแรมภายในที่มีอยู่แล้วมากกว่าที่จะใช้งานแรมที่อยู่ภายนอกขนาดของแรมภายในมีขนาด 128 ไบต์ สำหับไมโครคอนโทรลเลอร์เบอร์ 8031 และ 8051 และมีขนาด 256 ไบต์ สำหรับเบอร์ 8032 และ 8052 แต่ในที่นี้จะใช้แรมภายในนี้สูงสุดไม่เกิน 128 ไบต์ ดังนั้นจึงไม่มีปัญหาไม่ว่าผู้อ่านจะใช้ไมโครคอนโทรลเลอร์เบอร์ใดมาทำการศึกษา

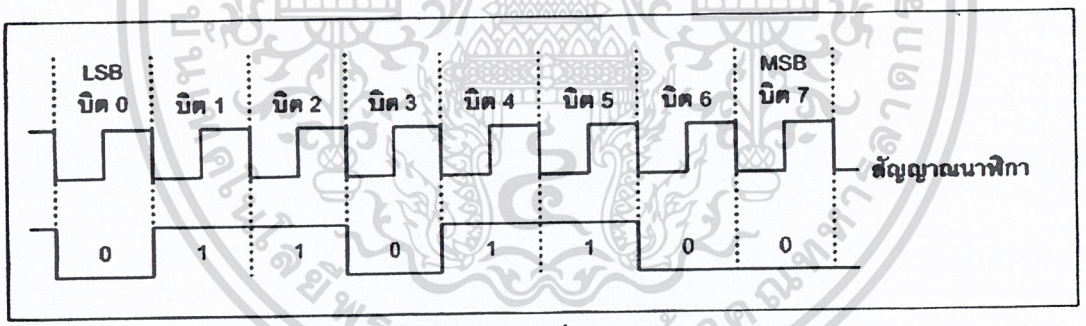
ในส่วนของแรมภายในประกอบด้วยรีจิสเตอร์ของไมโครคอนโทรลเลอร์ หน่วยความจำสแต็คสำหรับใช้งานและจัดการระบบภายในซับรูทีนก็อยู่ในส่วนของแรมภายในด้วย ดังนั้นขนาดของหน่วยความจำภายในที่ผู้เขียนโปรแกรมใช้งานได้จริงจึงน้อยกว่า 128 ไบต์ ในช่วงแอดเดรสระหว่าง 20H ถึง 2FH เรียกว่าบิตแอดเดรสเอเบิล (bit addressable range) ในส่วนนี้ใช้งานในการจัดแจงหรือโยกย้ายถ่ายเทบิตข้อมูลของคำสั่งไปยังแอดเดรส เปลี่ยนหรือเรียกใช้บิตใดบิตหนึ่ง ส่วนประกอบสุดท้ายที่ใช้แรมภายในก็คือ โปรแกรมมอนิเตอร์ซึ่งบรรจุอยู่ในอีพรอม ซึ่งการ

ประมวลผลในส่วนนี้ต้องใช้งานแรมภายในบางส่วนด้วยเช่นกัน รูปที่ 3-7 แสดงการแบ่งช่วงแอดเดรสของแรมภายใน

ถึงจุดนี้ผู้อ่านคงรู้จักกับไมโครคอนโทรลเลอร์ในตระกูล MCS-51 กันแล้ว และทราบวิธีการเขียนโปรแกรมเบื้องต้น หลังจากบทนี้ผู้อ่านคงพร้อมที่จะเจาะลึกเพื่อเรียนรู้การใช้งานได้มากขึ้นและพร้อมที่จะเขียนโปรแกรมทดลองจากตัวอย่างโปรแกรมต่าง ๆ ในบทต่อ ๆ ไปได้แล้ว

3.2 พอร์ตอนุกรม

การสื่อสารอบบอนุกรมนั้นจะแบ่งได้เป็น 2 แบบคือการสื่อสารอนุกรมแบบซิงโครนัสและการสื่อสารอนุกรมแบบอะซิงโครนัสจะมีสัญญาณนาฬิการ่วมอยู่กับการรับและส่งสัญญาณด้วย ตัวอย่างการส่งข้อมูลแบบซิงโครนัสก็คือคีย์บอร์ดของคอมพิวเตอร์ ซึ่งสายเส้นหนึ่งจะเป็นสายของสัญญาณนาฬิกา ส่วนอีกสายจะเป็นสายของข้อมูล ดังนั้นการติดต่อกันแบบซิงโครนัสนี้จะต้องใช้



รูปที่ 3-8 แบบอย่างง่ายที่สุดของข้อมูลอนุกรม

สายในการเชื่อมต่ออย่างน้อยที่สุด 3 เส้นคือ สัญญาณนาฬิกา, ข้อมูลและกราวด์ รูปที่ 3-8 แสดงให้เห็นถึงไทมิ่งไดอะแกรมของการส่งข้อมูลแบบซิงโครนัส

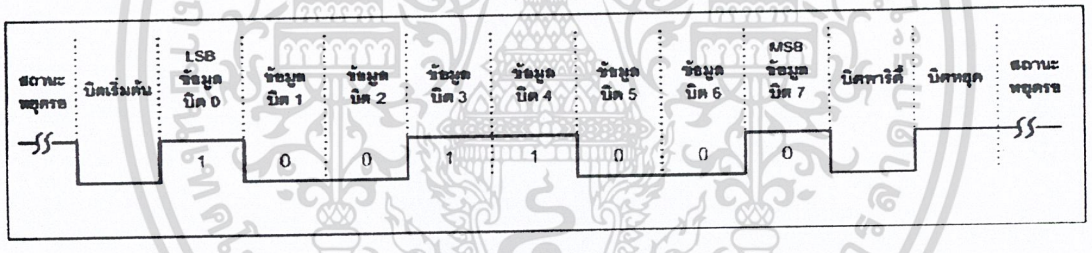
3.2.1 การสื่อสารข้อมูลแบบอะซิงโครนัส

การสื่อสารข้อมูลแบบอะซิงโครนัสคือการรับและส่งข้อมูลไปในสายโดยไม่จำเป็นต้องมีสัญญาณนาฬิการ่วมด้วยเหมือนกับการรับส่งข้อมูลแบบซิงโครนัส แต่จะใช้การกำหนดค่าสัญญาณนาฬิกาทั้งภาครับและภาคส่งนี้ว่า อัตราการถ่ายทอดข้อมูล หรือ บอดเรต (baudrate) มีหน่วยเป็น บิตต่อวินาที (bit per secone :bps)

รูปแบบของข้อมูลที่ใช้ในการรับส่งแบบอะซิงโครนัสประกอบด้วย 4 ส่วนด้วยกันคือ

1. บิตเริ่มต้น (Start Bit) ซึ่งจะมีขนาด 1 บิต
2. บิตข้อมูลแบบอนุกรมจะมีขนาด 5,6,7 หรือ 8 บิต
3. บิตตรวจสอบพาริตี้ (Parity Bit) จะมีขนาด 1 บิตหรือไม่มี
4. บิตปิดท้าย (Stop Bit) จะมีขาด 1,1.5 หรือ 2 บิต

รูป แสดงแบบของข้อมูลอนุกรมแบบอะซิงโครนัส ซึ่งเมื่อไม่มีข้อมูลที่จะส่งขา DATA จะมีสถานะลอจิก "1" ซึ่งจะเรียกสถานะนี้ว่าสถานะหยุดรอ (waiting stage) การเริ่มต้นส่งข้อมูลจากนั้นข้อมูลถูกส่งออกไป โดยเริ่มจากบิตที่มีนัยสำคัญต่ำสุด (LSB) ก่อน ซึ่งข้อมูลในไบต์ที่จะส่งอาจจะมีจำนวนบิต 5,6,7 หรือ 8 บิตก็ได้ จากนั้นจะตามด้วยบิตติพาริตี้ ซึ่งใช้เพื่อตรวจสอบความผิดพลาดที่เกิดขึ้นจากการส่งข้อมูล บิตสุดท้ายที่จะส่งคือบิตปิดท้าย ซึ่งจะให้ขา data มีสถานะลอจิก 1 อีกครั้งด้วยระยะเวลาอย่างน้อย 1 บิต, 1.5 บิต หรือ 2 บิต เพื่อเป็นการแสดงว่าสิ้นสุดข้อมูลแล้ว



รูปที่ 3-9 รูปแบบอย่างง่ายที่สุดของข้อมูลอนุกรมแบบอะซิงโครนัส

อุปกรณ์พิเศษที่ได้รับการออกแบบมาสำหรับการรับส่งข้อมูลแบบอะซิงโครนัสเรียกว่า Universal Asynchronous Receiver/Transmitter หรือ UART อัตราความเร็วในการรับส่งข้อมูลของการรับส่งข้อมูลแบบอะซิงโครนัสคือ ค่าบอดเรต ซึ่งก็คือค่าจำนวนบิตต่อวินาทีที่ใช้ในการรับและส่งขอ้มูลของการรับสองข้อมูลแบบอะซิงโครนัสคือ ค่าบอดเรต ซึ่งก็คือค่าจำนวนบิตต่อวินาทีที่ใช้ในการรับและส่งข้อมูล บอดเรตมาตรฐานที่ใช้สำหรับพอต์ตอนุกรม RS-232 ได้แก่ 110, 150 , 300, 600, 1200, 2400, 4800, 9600 และ 19200 บิตต่อวินาที และมีค่าเพิ่มมากขึ้นตามเทคโนโลยีของคอมพิวเตอร์ ซึ่งการรับส่งแบบอนุกรมโดยไม่ผ่านโมเด็มอาจจะสามารถกำหนดค่าบอดเรตได้สูงถึง 11520 บิตต่อวินาที เนื่องจากบอดเรตคือจำนวนบิตของข้อมูลที่สามารถถ่ายทอดได้ภายใน 1 วินาที ยกตัวอย่าง ข้อมูลอนุกรมถูกส่งในลักษณะ 8 บิต ไม่มีการตรวจสอบพาริตี้ มีบิตเริ่มต้น 1 บิต และบิตปิดท้าย 1 บิต ความยาวของข้อมูลที่รับส่งนี้เท่ากับ 10 บิต ถ้าใช้บอด

เรตในการส่งข้อมูลเท่ากับ 9600 บิตต่อวินาที ก้สจะสามารถรับส่งข้อมูลได้ด้วยความเร็ว 960 ไบต์ต่อวินาที และถ้ามีการใช้พาริตี้ความเร็วในการรับส่งข้อมูลจะเหลือเป็น 872 ไบต์ต่อวินาที

การตรวจสอบพาริตี้สามารถกำหนดให้เป็นแบบคี่ (odd) ,แบบคู่ (even) หรือไม่มีการตรวจสอบพาริตี้ก็ได้ การตรวจสอบพาริตี้เป็นการตรวจสอบจำนวนรวมของบิตที่เป็นลอจิก "1" ภายในข้อมูลที่ส่งไป 1 ไบต์ว่ามีจำนวนรวมเป็นเลขคู่หรือเลขคี่โดยต้องรวมบติพาริตี้เข้าไปด้วย ตัวอย่างข้อมูลที่จะทำการส่งมีขนาด 8 บิตและมีค่าเท่ากับ 99 ฐานสิบหก หรือ 10011001 ฐานสอง จะเห็นว่าข้อมูลในไบต์นี้มีจำนวนลอจิก "1" จำนวน 4 ตัวซึ่งเป็นเลขคู่ ดังนั้นถ้ากำหนดค่าพาริตี้เป็นคู่ค่าในพาริตี้ จะต้องมีลอจิกเป็น "0" แต่ถ้าพาริตี้เป็นคี่ ค่าที่บิตพาริตี้จะต้องเป็น "1" เพื่อให้ข้อมูล 1 ไบต์รวมทั้งบิตพาริตี้มีจำนวนบิตที่เป็นลอจิก "1" มีจำนวนรวมกันเป็นเลขคี่ ในตารางแสดงตัวอย่างของบิตพาริตี้ในการรับส่งข้อมูลอนุกรม

บิตพาริตี้ถูกสร้างขึ้นจากภาคส่งข้อมูลของ UART ซึ่งทางภาครับจะต้องทำการกำหนดคุณสมบัติการตรวจสอบพาริตี้ให้ตรงกันว่าจะตรวจสอบพาริตี้คี่หรือพาริตี้คู่ จากนั้นภาครับของ UART จะทำการตรวจสอบค่าพาริตี้ที่เกิดขึ้นว่าเป็นคู่หรือเป็นคี่ โดยการนับจำนวนลอจิก "1" ทั้งหมดรวมทั้งบิตพาริตี้ด้วย ถ้ากำหนดพาริตี้ไว้เป็นคู่แต่อ่านค่าตัวเลขในการนับออกมาได้ตัวเลขเป็นคี่ทางภาครับจะแสดงข้อผิดพลาดออกมาให้ผู้ใช้ทราบ นับเป็นการตรวจสอบความผิดพลาดที่เกิดขึ้นในการถ่ายทอดข้อมูลที่ง่ายที่สุดแต่จะเชื่อถือได้เมื่อมีบิตข้อมูลที่ทำการส่งผิดพลาดเพียงบิตเดียวเท่านั้น ถ้าข้อมูลที่ทำการส่งมีบิตที่ผิดพลาดมากกว่า 1 บิต การตรวจสอบด้วยวิธีนี้จะไม่ได้ผล สำหรับการตั้งพาริตี้บิตเป็น NONE นั้นทั้งภาครับและภาคส่ง จะไม่มีการตรวจสอบพาริตี้

ข้อมูล	บิตพาริตี้คู่	บิตพาริตี้คี่
00000000	0	1
00000001	1	0
00000010	1	0
00000011	0	1
00000100	1	0
11111110	0	1
11111111	1	0

ตารางที่ 3-3 แสดงบิตพาริตี้ของข้อมูล

คอมพิวเตอร์ในรุ่น AT เกือบทั้งหมดจะใช้ UART เบอร์ 16450 และ 16550 ส่วนคอมพิวเตอร์ในรุ่น XT ใช้ UART เบอร์ 8250 UART ชิปเหล่านี้มีระดับแรงดันเป็นแบบทีทีแอล (0 และ +5V) แต่เพื่อให้มีแรงดันเป็นไปตามมาตรฐาน RS-232 และเพื่อให้การรับส่งข้อมูลสามารถทำได้ที่ระยะทางไกลมากขึ้น ระดับแรงดันทีทีแอลจะถูกแปลงเป็นระดับแรงดันที่สูงขึ้น โดยลอจิก "0" มีระดับแรงดัน +3V ถึง +12V ในขณะที่ลอจิก "1" มีระดับแรงดัน -3V จนถึง -12V

3.2.2 มาตรฐานพอร์ตอนุกรมแบบ RS-232

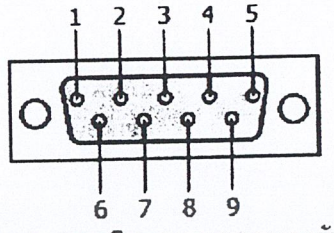
มาตรฐานการเชื่อมต่อแบบอนุกรม RS-232 เป็นมาตรฐานอุตสาหกรรมที่ออกแบบมาเพื่อใช้ในการส่งข้อมูลอนุกรมแบบอะซิงโครนัส 2 ทิศทาง โดยมาตรฐาน RS-232 ในอดีตนั้นถูกออกแบบมาเพื่อการส่งผ่านสายโทรศัพท์ไปยังคอมพิวเตอร์อีกชุดซึ่งอยู่ห่างไกลกัน โดยคณะกรรมการที่เรียกกันว่า EIA RS-232 มาตรฐานนี้ในช่วงแรกจะใช้คอนเน็กเตอร์เป็นแบบ DB-25 โดยกำหนดความยาวสูงสุดของสายสัญญาณไว้ที่ 50 ฟุต มีระดับสัญญาณตั้งแต่ -3 ถึง -12V แสดงว่ามีข้อมูล (Mark) และ +3 ถึง +12V แสดงว่าเป็นช่องว่าง (Space)

มาตรฐาน RS-232 ได้กำหนดรูปแบบของอุปกรณ์เชื่อมต่อข้อมูล (Data Terminal Equipment : DTE) กับวงจรข้อมูลปลายทาง (Data Circuit Terminating : DCE) ไว้ว่า อุปกรณ์ DTE จะต้องเป็นอุปกรณ์ที่มีการประมวลผลในตัวเช่น ไมโครคอนโทรลเลอร์หรือไมโครคอมพิวเตอร์ ซึ่งมีความสามารถในการสร้างบิตข้อมูลแบบอนุกรมได้ ส่วนอุปกรณ์ DCE จะทำหน้าที่เป็นเพียงตัวรับข้อมูลที่ส่งมาจาก DTE เท่านั้น โดยการรับส่งข้อมูลระหว่างอุปกรณ์ทั้งสองจะกระทำผ่านมาตรฐาน RS-232

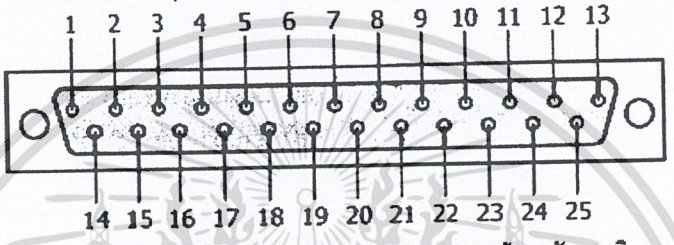
ข้อแตกต่างของอุปกรณ์ DTE และอุปกรณ์ DCE อย่างหนึ่งที่เห็นได้ชัดคือ คอนเน็กเตอร์ของ DTE จะเป็นตัวผู้ ส่วนคอนเน็กเตอร์ของ DCE จะเป็นตัวเมีย ซึ่งพอร์ตอนุกรมของคอมพิวเตอร์ที่ใช้กันอยู่ทั่วไปจะเป็นแบบ DTE ส่วนคอนเน็กเตอร์ที่อยู่โมเด็มจะเป็นแบบ DCE

สำหรับการใช้งานบนคอมพิวเตอร์ พอร์ตอนุกรม RS-232 มักถูกใช้เชื่อมต่อกับโมเด็มหรือเมาส์ โดยสามารถรับส่งข้อมูลได้ที่ความยาวของสายสัญญาณสูงสุดถึง 20 เมตร

3.2.3 คอนเน็กเตอร์สำหรับพอร์ต RS-232 และการเชื่อมต่อ



(ก) คอนเน็กเตอร์ชนิดกรม 9 ขาหรือแบบ DB-9 (มองจากด้านหลังคอมพิวเตอร์)



(ข) คอนเน็กเตอร์ชนิดกรม 25 ขาหรือแบบ DB-25 (มองจากด้านหลังคอมพิวเตอร์)

คอนเน็กเตอร์ DB-9	คอนเน็กเตอร์ DB-25	ชื่อของสายสัญญาณ	ชนิดของสายสัญญาณ
1	8	Data Carrier Detect : DCD	อินพุต
2	3	Received Data : RxD	อินพุต
3	2	Transmitted Data : TxD	เอาต์พุต
4	20	Data Terminal Ready : DTR	เอาต์พุต
5	7	Signal Ground : GND	-
6	6	Data Set Ready : DSR	อินพุต
7	4	Request To Send : RTS	เอาต์พุต
8	5	Clear To Send : CTS	อินพุต
9	22	Ring Indicator : RI	อินพุต

รูปที่ 3-10 การจัดขาของคอนเน็กเตอร์พอร์ตอนุกรมตามมาตรฐาน RS-232 ทั้งแบบ DB-9 และ DB-25

มาตรฐานการเชื่อมต่อแบบ RS-232 จะใช้คอนเน็กเตอร์แบบ DB-25 ตัวผู้หรือ DB-9 ตัวผู้ ซึ่งคอนเน็กเตอร์แบบ DB-25 จะมีขาต่อใช้งานเพียง 9 เส้นเช่นเดียวกับคอนเน็กเตอร์แบบ DB-9 เนื่องจากขาอื่นๆที่เคยใช้งานในอดีต ปัจจุบันมีการใช้งานไม่มากนัก จึงถูกยกเลิกไป โดยแสดงรูปร่างและตำแหน่งขาในรูปที่ 3-10

สำหรับการเชื่อมต่อคอนพิวเตอร์กับอุปกรณ์ภายนอกแสดงดังในรูป ลูกศรในรูปแสดงถึงทิศทางของข้อมูล ในรูป (ก) เป็นการเชื่อมต่อแบบ Null modem หรือการเชื่อมต่อโดยตรงโดยไม่ต่องผ่านโมเด็ม โดยมีการตรวจสอบหรือแฮนด์เช็กเต็มรูปแบบ ส่วนในรูป (ข) เป็นการเชื่อมต่อแบบ Null modem ในลักษณะที่ใช้สายสัญญาณเพียง 3 เส้นโดยเส้นหนึ่งสำหรับส่งข้อมูล อีกเส้นสำหรับรับข้อมูล และเส้นสุดท้ายเป็นกราวด์ สำหรับรายละเอียดหน้าที่การทำงานในแต่ละขาของพอร์ตอนุกรม RS-232 มีดังนี้

(ก) การต่ออุปกรณ์ภายนอกเข้ากับคอมพิวเตอร์แบบ Null modem

(ข) การต่ออุปกรณ์ภายนอกเข้ากับคอมพิวเตอร์แบบ RS-232 โดยใช้สายสัญญาณเพียง 3 เส้น

รูปที่ 3-11 การต่ออุปกรณ์เข้ากับคอมพิวเตอร์

Data Carrier Detect : DCD หรืออาจเรียกว่า Carrier Detect :CD ขานี้จะแอกตีฟเมื่อมีการส่งสัญญาณพาห์จากอุปกรณ์สื่อสารข้อมูลเช่น โมเด็ม สำหรับการใช้งานปกติ ขานี้จะไม่ได้ถูกใช้งานมากนัก

Receive Data : RD **หรือ** RxD ขานี้ใช้เพื่อรับสัญญาณอนุกรมเข้ามายังคอมพิวเตอร์ โดยนำข้อมูลที่อ่านได้เก็บไว้ในรีจิสเตอร์ บัฟเฟอร์

Transmitted Data : TD **หรือ** TxD ขานี้ใช้เพื่อส่งข้อมูลออกจากคอมพิวเตอร์ โดยนำข้อมูลที่เก็บอยู่ในบัฟเฟอร์สำหรับส่งข้อมูลส่งออกไป

Data Terminal Ready : DTR เป็นขาสัญญาณที่ส่งออกจากคอมพิวเตอร์เพื่อให้อุปกรณ์ปลายทางรับรู้ว่า ต้องการติดต่อด้วย และขา DTR เชื่อมต่อกับขา DSR ของอุปกรณ์ปลายทาง และขา DTR ของอุปกรณ์ปลายทางจะต้องเชื่อมต่อกับขา DSR ของคอมพิวเตอร์ ถ้าใช้การเชื่อมต่อเป็นแบบ Null Modem ซึ่งใช้สายในการเชื่อมต่อเพียง 3 เส้น จะต้องต่อขา DTR และ DSR ของตัวมันเองเข้าด้วยกันและต้องต่อกับขา DCD ด้วยในกรณีที่โปรแกรมสื่อสารที่ใช้มีการตรวจจับสัญญาณพาห์

Signal Ground : GND ขากราวด์ของระบบ

Data Set Ready : DSR ขานี้จะใช้คู่กับขา DTR เพื่อตรวจสอบการเชื่อมต่อกันระหว่างคอมพิวเตอร์กับอุปกรณ์ที่ปลายทาง ซึ่งขา DSR นี้จะเป็นขาสำหรับรับข้อมูลจากภายนอกซึ่งถูกส่งมาจากขา DTR

Request To Send : RTS เป็นขาสำหรับส่งสัญญาณร้องขอให้ทางอุปกรณ์ปลายทางส่งข้อมูลกลับมายังคอมพิวเตอร์ โดยขาที่รับสัญญาณ RTS ก็คือขา CTS ในกรณีที่ใช้การเชื่อมต่อแบบ Null Medem 3 สาย จะต้องเชื่อต่อขา RTS และ CTS ของตัวมันเองเข้าด้วยกัน เพื่อจะให้การรับและส่งข้อมูลสามารถเกิดขึ้นได้ตลอดเวลา

Clear To Send : CTS ขานี้จะคอยรับสัญญาณจากขา RTS เมื่อรับสัญญาณได้ ข้อมูลที่ขา TxD จะถูกส่งออกไป ดังนั้นขานี้จึงถูกใช้เพื่อตรวจสอบอุปกรณ์ต่อพ่วงพร้อมที่จะรับข้อมูลหรือไม่

Ring Indicator : RI ใช้แสดงสถานะสัญญาณเรียกจากสายโทรศัพท์ ปกติในการสื่อสารโดยทั่งไปสายนี้จะไม่ถูกใช้งาน จะใช้งานก็ต่อเมื่อมีการเชื่อมต่อกับโมเด็มและโปรแกรมมีการตรวจสอบสัญญาณนี้เท่านั้น

3.2.4 UART

UART มาจากคำว่า Universal Asynchronous Receiver Transmitter ซึ่งหมายถึงอุปกรณ์ที่ทำหน้าที่รับและส่งข้อมูลแบบอะซิงโครนัสนั่นเอง สำหรับการสื่อสารอนุกรมบนคอมพิวเตอร์แล้ว UART ถือว่าเป็นหัวใจสำคัญของการสื่อสารอนุกรม

หน้าที่หลักของ UART คือทำหน้าทีแปลงข้อมูลที่อยู่ในรูปแบบขนานจากคอมพิวเตอร์ให้อยู่ในรูปแบบอนุกรมแบบอะซิงโครนัส แล้วส่งออกไป และทำหน้าที่แปลงสัญญาณอนุกรมแบบอะซิงโครนัสป้อนมายัง UART ให้เป็นแบบขนานก่อนที่จะส่งเข้าสู่คอมพิวเตอร์ ซึ่งนอกจาก UART จะส่งข้อมูลไปยังคอมพิวเตอร์แล้ว ยังแจ้งข้อมูลอื่นๆ ให้คอมพิวเตอร์รับทราบด้วย เช่น อัตราเร็วในการรับส่งข้อมูล (บอดเรต), ความผิดพลาดที่เกิดขึ้นระหว่างการถ่ายทอดข้อมูล (ผิดพลาดจากพาริตี้, เฟรมข้อมูล, โอเวอร์รัน) เป็นต้น

ภายใน UART จะมีส่วนของวงจรสร้างบอดเรตแบบโปรแกรมได้(programmable buadrate generator) โดยการกำหนดค่าตัวหารให้กับสัญญาณนาฬิกาของ UART โดยตัวหารนี้มีขนาด 16 ฮาล์ฟดูเพล็กซ์ (half duplex) และฟูลดูเพล็กซ์เป็นการส่งแบบทิศทางเดียว ส่วนการส่งแบบฟูลดูเพล็กซ์นั้นสามารถรับและส่งข้อมูลได้ในคราวเดียวกัน

ชนิดของ UART

ในเครื่องคอมพิวเตอร์ทั่วไปมี UART ที่ใช้งานกันอยู่ 2 เบอร์คือ 8250 ซึ่งเป็น UART มาตรฐานที่มีใช้กันมายาวนาน UART เบอร์นี้จะมีบัฟเฟอร์สำหรับรับและส่งข้อมูลตำแหน่งเยวกัน ทำให้การรับแบะส่งข้อมูลถูกจำกัดความเร็วอยู่ที่ 57.6 กิโลบิตต่อวินาทีเท่านั้น แต่ UART เบอร์นี้ก็ถือว่าเป็นต้นแบบของ UART ที่ใช้ในคอมพิวเตอร์ โดยคอมพิวเตอร์ทุกๆรุ่นจะต้องสนับสนุนการทำงานตามรูปแบบของ UART เบอร์นี้

UART อีกเบอร์หนึ่งคือ 16450 มีความสามารถรับส่งข้อมูลได้ที่ความเร็ว 115,200 บิตต่อวินนาที และเพิ่มรีจิสเตอร์สำหรับพักข้อมูลสำหรับ UART นอกจากนั้นยังเพิ่มส่วนของชิฟต์รีจิสเตอร์แบบ FIFO (First In First Out) ขนาด 16 ไบต์เข้าไป ทำให้สามารถสนับสนุนความเร็วในการรับส่งข้อมูลที่ 256 กิโลบิตต่อวินาทีได้ โดยคอมพิวเตอร์ในปัจจุบันใช้ UART เบอร์นี้หรือใหม่กว่า เช่น เบอร์ TL16C750 ซึ่งมีรีจิสเตอร์แบบ FIFO ขนาด 64 ไบต์ ทำงานได้ที่ระดับแรงดัน +5V และ +3V มีโหมดประหยัดพลังงาน สามารถรับส่งข้อมูลได้ที่ความเร็ว 1 เมกะบิตต่อวินาทีเมื่อใช้สัญญาณนาฬิกา 16 MHz

วงจรภายในและรีจิสเตอร์ของพอร์ตอนุกรม RS-232

เครื่องคอมพิวเตอร์โดยทั่วไปสามารถต่อพอร์ตอนุกรม RS-232 สูงสุดได้ 4 พอร์ต ซึ่งจะมีชื่อเรียกเป็น COM1, COM2, COM3 และ COM4 ซึ่งพอร์ตอนุกรมแต่ละตัวต่างก็ใช้งาน UART ภายในคอมพิวเตอร์ในการติดต่อกับอุปกรณ์ภายนอกเช่นเดียวกัน

ในรูปที่ 3.2.4 แสดงไดอะแกรมการทำงานภายในของพอร์ตอนุกรม ซึ่งประกอบไปด้วยรีจิสเตอร์ขนาด 8 บิต 8 ตัวที่ใช้งานร่วมกับ UART แอดเดรสของรีจิสเตอร์ภายในพอร์ตอนุกรมสามารถคำนวณได้จากค่ารีจิสเตอร์พื้นฐานของพอร์ตอนุกรม ยกตัวอย่าง พอร์ตอนุกรม COM1 มีแอดเดรสอยู่ที่ 3F8H ตำแหน่งของรีจิสเตอร์ต่างๆจะเป็นตำแหน่งที่บวกเข้าไปกับค่า 3F8H ตำแหน่งของรีจิสเตอร์ต่างๆจะเป็นตำแหน่งที่บวกเข้าไปกับค่า 3F8H โดยรีจิสเตอร์ที่ใช้งานกับพอร์ตอนุกรมมีดังนี้

00H เป็นรีจิสเตอร์บัฟเฟอร์สำหรับเก็บข้อมูลที่รับเข้ามาหรือเตรียมข้อมูลที่จะส่งออกไป

01H รีจิสเตอร์อีนาเบิลการอินเตอร์รัปต์ ใช้ในการเซตโหมดการอินเตอร์รัปต์ของพอร์ตอนุกรม

02H รีจิสเตอร์แสดงโหมดการอินเตอร์รัปต์ ใช้เพื่อตรวจสอบโหมดของการอินเตอร์รัปต์เมื่อมีการอินเตอร์รัปต์เกิดขึ้น

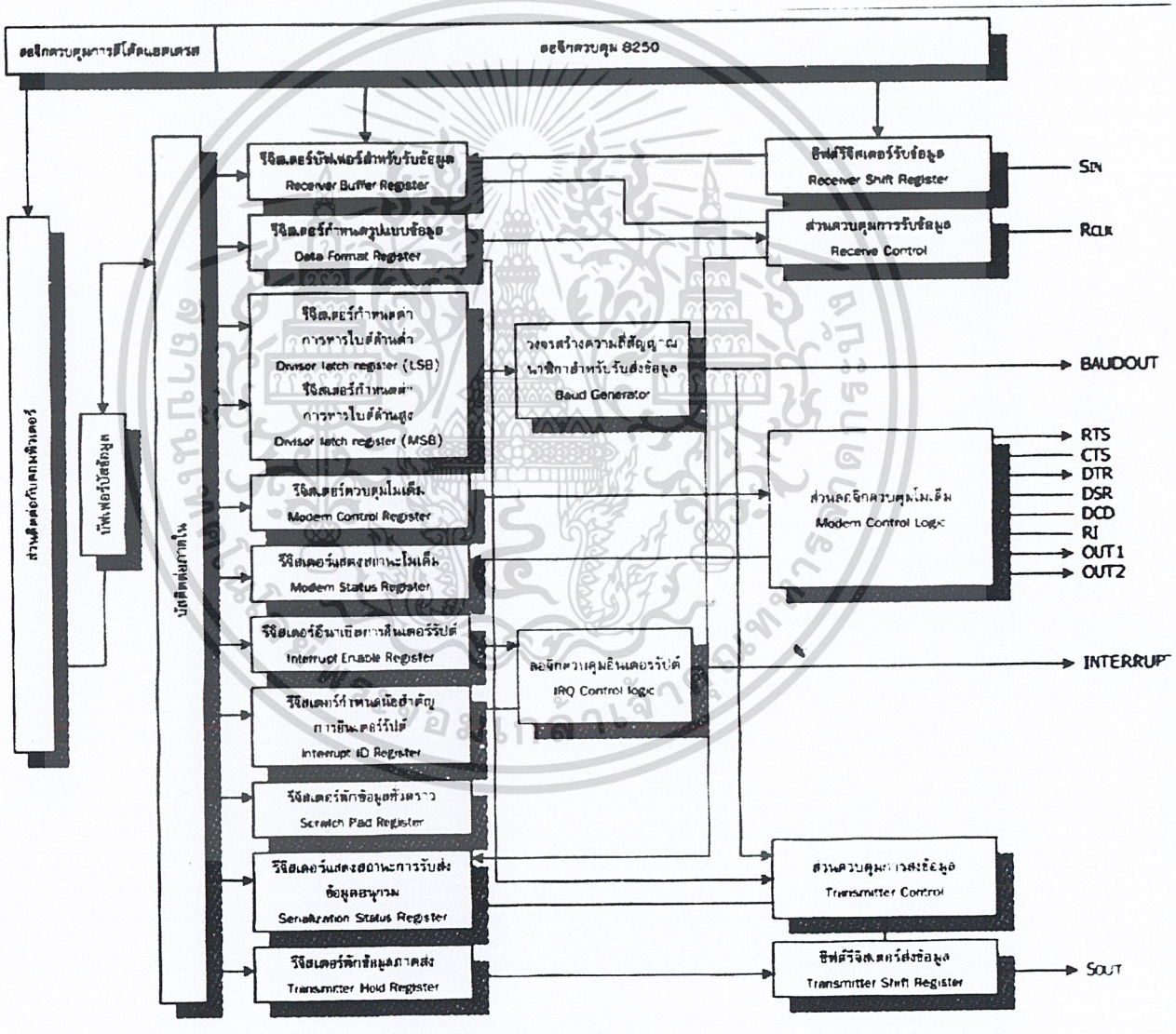
03H รีจิสเตอร์กำหนดรูปแบบของข้อมูล

04H รีจิสเตอร์ควบคุมโมเด็ม ใช้ตรวจสอบบิตสำหรับติดต่อกับโมเด็ม เช่น RTS หรือ DTR

05H รีจิสเตอร์แสดงสถานะการรับและการส่งข้อมูลแบบอนุกรม

06H รีจิสเตอร์แสดงสถานะของโมเด็ม ซึ่งจะแสดงสถานะของขา DCD, RI, DSR และ CTS

07H รีจิสเตอร์สำหรับการเก็บข้อมูลชั่วคราว



รูปที่ 3-12 ไดอะแกรมการทำงานภายในของพอร์ตอนุกรมของเครื่องคอมพิวเตอร์

รีจิสเตอร์ตำแหน่ง 00H : รีจิสเตอร์บัฟเฟอร์

เป็นรีจิสเตอร์เพื่อเก็บข้อมูลที่ต้องการจะส่งจะต้องกำหนดให้บิต DLAB ในรีจิสเตอร์กำหนดรูปแบบข้อมูล (03H) จะต้องมีสถานะเป็น 0 ซึ่งการเขียนข้อมูลมายังแอดเดรสนี้ เป็นการส่งข้อมูลไปยังรีจิสเตอร์ส่งข้อมูลจะถูกส่งออกไปแบบอนุกรม สำหรับการรับข้อมูล เมื่อข้อมูลที่รับเข้ามาเรียบร้อยและแปลงเป็นแบบขนานแล้ว ข้อมูลจะถูกเคลียร์ และเตรียมพร้อมสำหรับการรับข้อมูลไบต์ต่อไป

รีจิสเตอร์ตำแหน่ง 01H : รีจิสเตอร์อีนาเบิลการอินเตอร์รัปต์

เป็นรีจิสเตอร์สำหรับการอีนาเบิลการอินเตอร์รัปต์ ซึ่งเป็นการกำหนดให้ UART สร้างสัญญาณอินเตอร์รัปต์ขึ้นมา ฟังก์ชันการทำงานในแต่ละบิตของรีจิสเตอร์นี้มีดังนี้

บิต 7	บิต 6	บิต 5	บิต 4	บิต 3	บิต 2	บิต 1	บิต 0
0	0	0	0	SINP	ERBK	TBE	RxRD

บิต 4-7 บิตเหล่านี้ไม่ถูกใช้งาน กำหนดให้เท่ากับ “0”

SINP อีนาเบิลการอินเตอร์รัปต์เนื่องจากเกิดการเปลี่ยนสถานะที่ขาอินพุต CTS, DSR, DCD หรือขา RI
“1” อีนาเบิลการอินเตอร์รัปต์
“0” ไม่มีการใช้อินเตอร์รัปต์รูปแบบนี้หรือดิสเอเบิล

ERBK อีนาเบิลการอินเตอร์รัปต์เนื่องจากเกิดความผิดพลาดขึ้นด้วยสาเหตุจากพาริตี้, โอเวอร์รัน, เฟรมข้อมูล หรือการเบรกข้อมูล
“1” อีนาเบิลการอินเตอร์รับปต์
“0” ไม่มีการใช้อินเตอร์รัปต์รูปแบบนี้หรือดิสเอเบิล

TBE อีนาเบิลการอินเตอร์รัปต์เนื่องจากรีจิสเตอร์บัฟเฟอร์ได้รับข้อมูลเรียบร้อย
“1” อีนาเบิลการอินเตอร์รัปต์
“0” ไม่มีการใช้อินเตอร์รัปต์รูปแบบนี้หรือดิสเอเบิล

รีจิสเตอร์ตำแหน่ง 02H : รีจิสเตอร์แสดงโหมดและสถานะการอินเตอร์รัปต์

มีรายละเอียดของแต่ละบิตดังนี้

บิต 7	บิต 6	บิต 5	บิต 4	บิต 3	บิต 2	บิต 1	บิต 0
0	0	0	0	0	ID1	ID0	PND

บิต 3-7 ไม่ได้ใช้งาน อ่านได้เท่ากับ "0"

ID1,ID0 ใช้งานร่วมกันเพื่อแจ้งสาเหตุของการเกิดอินเตอร์รัปต์

"00" เกิดการอินเตอร์รัปต์เนื่องจากการเปลี่ยนแปลงของขาอินพุตขึ้น การอินเตอร์รัปต์แบบนี้มีนัยสำคัญเป็นอันดับ 4

"01" เกิดการอินเตอร์รัปต์เนื่องจากรีจิสเตอร์บัฟเฟอร์ส่งข้อมูลว่างขึ้น การอินเตอร์รัปต์แบบนี้มีนับสำคัญเป็นอันดับ 3

"10" เกิดการอินเตอร์รัปต์เนื่องจากข้อมูลถูกเก็บลงในรีจิสเตอร์บัฟเฟอร์สำหรับข้อมูลเรียบร้อยแล้ว การอินเตอร์รัปต์แบบนี้มีนัยสำคัญเป็นอันดับ 2

"11" เกิดการอินเตอร์รัปต์เนื่องจากความผิดพลาดในการถ่ายทอดข้อมูล หรือเกิดการเบรก (break : เกิดการหยุดถ่ายทอดข้อมูลกระทันหัน) การอินเตอร์รัปต์แบบนี้มีนัยสำคัญเป็นอันดับ 1 หรือมีนัยสำคัญสูงสุด

PND ใช้แสดงสถานะของการเกิดอิเตอร์รัปต์

"1" แสดงว่าไม่มีการอินเตอร์รัปต์

"0" แสดงว่ามีการอินเตอร์รัปต์เกิดขึ้น

เมื่อมีการสร้างสัญญาณอินเตอร์รัปต์ขึ้น จะต้องมีการเคลียร์ค่าก่อนที่จะให้เกิดอินเตอร์รัปต์ครั้งต่อไป โดยสามารถทำได้ดังนี้คือ

ถ้าเกิดอินเตอร์รัปต์เนื่องจากการเปลี่ยนแปลงของขาอินพุตจะต้องอ่านค่าจากรีจิสเตอร์แสดงสถานะของโมเด็ม (รีจิสเตอร์ตำแหน่ง 06H) เพื่อเคลียร์ค่าการอินเตอร์รัปต์

ถ้าเกิดการอินเตอร์รัปต์เนื่องจากบัฟเฟอร์ส่งข้อมูลว่าง จะต้องเขียนข้อมูลไปยังรีจิสเตอร์บัฟเฟอร์ส่งข้อมูล (รีจิสเตอร์ตำแหน่ง 00H) หรืออ่านค่ารีจิสเตอร์แสดงสถานะอินเตอร์รัปต์ (รีจิสเตอร์ตำแหน่ง 02H) เพื่อเคลียร์ค่าการอินเตอร์รัปต์

ถ้าเกิดอินเตอร์รัปต์เนื่องจากการเก็บข้อมูลจากรีจิสเตอร์บัฟเฟอร์สำหรับข้อมูลเรียบร้อย จะต้องเคลียร์ค่าอินเตอร์รัปต์โดยการอ่านข้อมูลจากรีจิสเตอร์บัฟเฟอร์

ถ้าเกิดอินเตอร์รัปต์เนื่องจากความผิดพลาดในการรับส่งข้อมูลหรือเกิดการเบรก จะต้องเคลียร์ค่าอินเตอร์รัปต์โดยการอ่านค่ารีจิสเตอร์แสดงสถานะการรับและส่งข้อมูลแบบอนุกรม

รีจิสเตอร์ตำแหน่ง 03H : รีจิสเตอร์กำหนดรูปแบบของข้อมูล

มีรายละเอียดหน้าที่ของแต่ละบิตดังนี้

บิต 7	บิต 6	บิต 5	บิต 4	บิต 3	บิต 2	บิต 1	บิต 0
DLAB	BRK	PAR2	PAR1	PAR0	STOP	DAB1	DAB0

DLAB ใช้ในการกำหนดหน้าที่การทำงานของรีจิสเตอร์บัฟเฟอร์ (00H)

"1" เป็นการเข้าสู่โหมดการหาร่าบอดเรต

"0" เป็นการเข้าถึงรีจิสเตอร์บัฟเฟอร์ (รีจิสเตอร์ตำแหน่ง 00H) และรีจิสเตอร์สำหรับอีนาเบิลการอินเตอร์รัปต์ (รีจิสเตอร์ตำแหน่ง 01H)

เมื่อบิต DLAB เป็น "1" รีจิสเตอร์บัฟเฟอร์ (00H) และรีจิสเตอร์อีนาเบิลการอินเตอร์รัปต์ (01H) จะใช้สำหรับโหลดค่าการหารความถี่สำหรับกำหนดค่าบอดเรต โดยรีจิสเตอร์ 00H เก็บค่าตัวหารไบต์ต่ำ ส่วนรีจิสเตอร์ 01H ใช้เก็บค่าตัวหารไบต์สูง การหาค่าบอตเรตสามารถเขียนเป็นสมการได้ดังนี้

บอตเรต = 115200/ค่าตัวหาร 16 บิต

ค่าตัวเลข 115200 มาจากความถี่ของคริสตอลในวงจร UART ภายในเครื่องคอมพิวเตอร์ โดยคริสตอลที่มีความถี่ 1.8432 MHz วงจรภายใน UART จะทำการหารค่าความถี่นี้ด้วย 16 ทำให้ได้ค่าความถี่ 115200 Hz ออกมา

ค่าตัวหาร 16 บิต = ข้อมูลในรีจิสเตอร์ 00H+(256 × ข้อมูลในรีจิสเตอร์ 01H)

สมมติว่าต้องการค่าบอตเรตเท่ากับ 9600 ค่าตัวหารที่ใช้จะต้องมีค่าเท่ากับ 12 ซึ่งค่านี้จะต้องถูกโหลดลงในรีจิสเตอร์ 00H และโหลดค่า 0 ลงไปในรีจิสเตอร์ 01H ค่าตัวหารที่ทำให้เกิดค่าบอดเรตสูงสุดที่ 115200 บิตต่อวินาทีคือ ค่า 0001 นั่นคือรีจิสเตอร์ 00H มีค่าเท่ากับ 1 และรีจิสเตอร์ 01H มีค่าเท่ากับ 0

BRK ใช้ควบคุมการหยุดถ่ายทอดข้อมูล

"1" สามารถหยุดหรือเบรกได้

"0" ไม่มีการหยุดหรือเบรกได้

PAR2,PAR1,PAR0 ใช้เพื่อกำหนดบิตพาริตี

"000" ไม่ใช้บิตพาริตี้

"001" กำหนดพาริตี้คี่

"011" กำหนดพาริตี้คู่

"101" มาร์ก (mark)

"111" ช่องว่าง (space)

STOP ใช้กำหนดจำนวนบิตปิดท้าย

"1" มีบิตปิดท้าย 2 บิต

"0" มีบิตปิดท้าย 1 บิต

DAB1,DAB0 ใช้รวมกันในการกำหนดจำนวนบิตของข้อมูลที่ต้องการถ่ายทอด

"00" จำนวนบิตข้อมูลเท่ากับ 5 บิต

"01" จำนวนบิตข้อมูลเท่ากับ 6 บิต

"10" จำนวนบิตข้อมูลเท่ากับ 7 บิต

"11" จำนวนบิตข้อมูลเท่ากับ 8 บิต

รีจิสเตอร์ตำแหน่ง 04H : รีจิสเตอร์ควบคุมโมเด็ม

มีรายละเอียดหน้าที่ของแต่ละบิตดังนี้

บิต 7	บิต 6	บิต 5	บิต 4	บิต 3	บิต 2	บิต 1	บิต 0
0	0	0	LOOP	OUT2	OUT1	RTS	DTR

บิต 5-7 ไม่มีการใช้งาน อ่านค่าได้เท่ากับ 0

LOOP "1" อีนาเบิลการส่งค่ากลับ

"0" ดิสเอเบิล

OUT1,OUT2 "1" อีนาเบิลการใช้งานภายใน

"0" ดิสเอเบิล

RTS ใช้ควบคุมการทำงานของขา RTS (Ready To Send)

"1" อีนาเบิล

"0" ดิสเอเบิล

DTR ใช้ควบคุมการทำงานของขา DTR (Data Terminal Ready)

"1" อีนาเบิล

"0" ดิสเอเบิล

รีจิสเตอร์ตำแหน่ง 05H : รีจิสเตอร์แสดงสถานะการรับและส่งข้อมูลอนุกรมของ UART

ใช้งานร่วมกับรีจิสเตอร์แสดงโหมดและสถานะของการอินเตอร์รัปต์ (รีจิสเตอร์ตำแหน่ง 02H) เพื่อแสดงสาเหตุของการเกิดอินเตอร์รัปต์ มีรายละเอียดหน้าที่ของแต่ละบิตดังนี้

บิต 7	บิต 6	บิต 5	บิต 4	บิต 3	บิต 2	บิต 1	บิต 0
0	TXE	TBE	BREK	FRME	PARE	OVFE	RxRD

TXE (Transmitter Empty) "1" แสดงว่ารีจิสเตอร์บัฟเฟอร์สำหรับส่งข้อมูลว่าง

"0" แสดงว่ายังคงมีข้อมูล 1 ไบต์เก็บอยู่ในรีจิสเตอร์บัฟเฟอร์สำหรับส่งข้อมูล

TBE (Transmitter Buffer Empty) "1" รีจิสเตอร์บัฟเฟอร์สำหรับส่งข้อมูลว่าง

"0" ยังคงมีข้อมูล 1 ไบต์เก็บอยู่ในรีจิสเตอร์บัฟเฟอร์สำหรับส่งข้อมูล

BREK (Break) "1" UART ตรวจพบการเบรก

"0" ไม่มีการเบรก

FRME (Frame Error) "1" UART ตรวจพบความผิดพลาดด้านเฟรมข้อมูล

"0" ไม่พบความผิดพลาดด้านเฟรมข้อมูล

PARE (Parity Error) "1" UART ตรวจพบความผิดพลาดทางพาริตี้

"0" ไม่พบความผิดพลาดทางพาริตี้

OVRE (Overrun Error) "1" UARTตรวจพบความผิดพลาดแบบโอเวอร์รัน

"0" ไม่พบความผิดพลาดแบบโอเวอร์รัน

RxRD (Received Data Ready)"1" มีการรับข้อมูลเข้ามาเก็บไว้ในบัฟเฟอร์

"0" ไม่มีข้อมูล

.รีจิสเตอร์ตำแหน่ง 06H : รีจิสเตอร์แสดงสถานะของโมเด็ม

ใช้เพื่อกำหนดสถานะสัญญาณอินพุต ของพอร์ตอนุกรม RS-232 ซึ่งได้แก่ สัญญาณ DCD, DSR, CTS และ RI สำหรับการเชื่อมต่อใช้งานแบบอเนกประสงค์ ดังมีรายละเอียดหน้าที่ของแต่ละบิตต่อไปนี้

บิต 7	บิต 6	บิต 5	บิต 4	บิต 3	บิต 2	บิต 1	บิต 0
DCD	RI	DSR	CTS	DDCD	DRI	DDSR	DCTS

DCD ใช้แสดงสภาวะของขา DCD

"1" แสดงว่าที่ขา DCD เป็นลอจิก "1"

"0" แสดงว่าที่ขา DCD เป็นลอจิก "0"

RI ใช้แสดงสภาวะของขา RI

"1" แสดงว่าที่ขา RI เป็นลอจิก "1"

"0" แสดงว่าที่ขา RI เป็นลอจิก "0"

DSR ใช้แสดงสภาวะของขา DSR

"1" แสดงว่าที่ขา DSR เป็นลอจิก "1"

"0" แสดงว่าที่ขา DSR เป็นลอจิก "0"

DCTS (Delta Clear To Send) ใช้แจ้งการเปลี่ยนแปลงที่เกิดขึ้นของบิต CTS

"1" แสดงว่าบิต CTS (Clear To Send) เกิดการเปลี่ยนแปลงเมื่อเทียบจากการอ่านค่าครั้งที่แล้ว

"0" แสดงว่าไม่มีการเปลี่ยนแปลงเมื่อเทียบกับการอ่านค่าครั้งที่แล้ว

DDSR(Delta Data Set Ready)ใช้แจ้งการเปลี่ยนแปลงที่เกิดขึ้นของบิต DSR

"1" แสดงว่าบิต DSR (Data Set Ready) เกิดการเปลี่ยนแปลงเมื่อเทียบ จากการอ่านค่าครั้งที่แล้ว

"0" แสดงว่าไม่มีการเปลี่ยนแปลงเมื่อเทียบกับการอ่านค่าครั้งที่แล้ว

DRI (Delta Ring Indicator)ใช้แจ้งการเปลี่ยนแปลงที่เกิดขึ้นของบิต RI

"1" แสดงว่าบิต RI (Ringing Indicator) เกิดการเปลี่ยนแปลงเมื่อเทียบจากการอ่านค่าครั้งที่แล้ว

"0" แสดงว่าไม่มีการเปลี่ยนแปลงเมื่อเทียบกับการอ่านค่าครั้งที่แล้ว

DDCD (Delta Data Carrier Detect)ใช้แจ้งการเปลี่ยนแปลงที่เกิดขึ้นของบิต DDCD

"1" แสดงว่าบิต CTS (Clear To Send) เกิดการเปลี่ยนแปลงเมื่อเทียบจากการอ่านค่าครั้งที่แล้ว

"0" แสดงว่าไม่มีการเปลี่ยนแปลงเมื่อเทียบกับการอ่านค่าครั้งที่แล้ว

DCTS (Delta Clear To Send) ใช้แสดงสภาวะของขา CTS

"1" แสดงว่าที่ขา CTS เป็นลอจิก "1"

"0" แสดงว่าที่ขา CTS เป็นลอจิก "0"

รีจิสเตอร์ตำแหน่ง 07H : รีจิสเตอร์สำหรับเก็บข้อมูลชั่วคราว

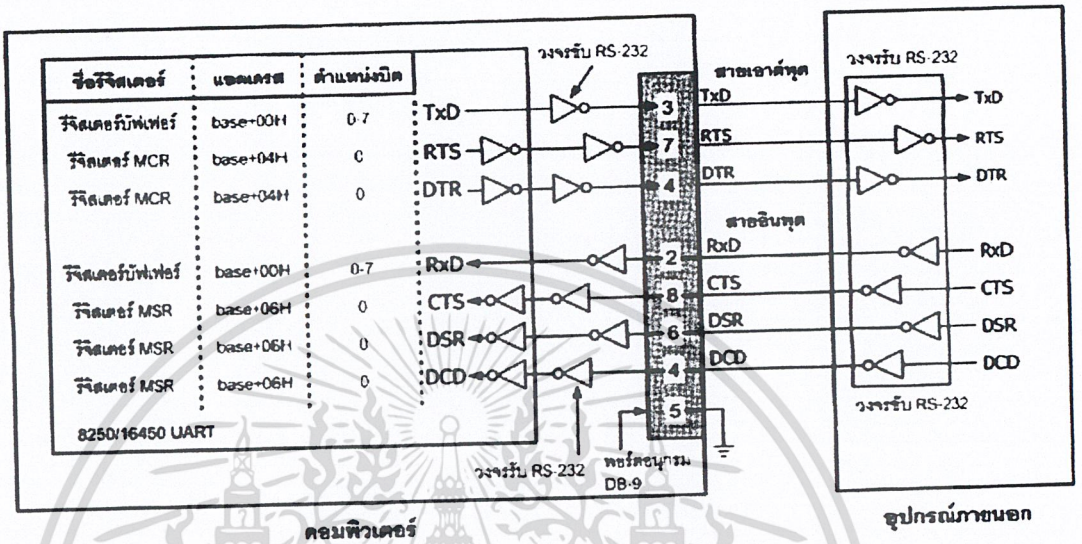
ทำหน้าที่เป็นหน่วยความจำแรมขนาด 1 ไบต์ การอ่านและเขียนข้อมูลที่รีจิสเตอร์ตัวนี้ไม่ส่งผลใดๆต่อการใช้งาน UART

ลักษณะสัญญาณอินพุตและเอาต์พุตของพอร์ต RS-232

สัญญาณเอาต์พุตที่ใช้ควบคุม (RTS และ DTR) และสัญญาณแสดงสถานะอินพุต (CTS, DSR และ DCD) ของพอร์ตอนุกรม RS-232 จะถูกกลับสถานะภายในตัว UART ส่วนสัญญาณข้อมูลทั้งภาคส่งและรับจะไม่ถูกกลับสุถานะ UART จะให้ระดับสัญญาณเอาต์พุตออกมาเป็นแบบทีทีแอลเท่านั้น ดังนั้นเมื่อสัญญาณถูกส่งออกมาจาก UART จึงต้องส่งเข้าสู่วงจรขับเพื่อปรับระดับแรงดันให้ระดับสัญญาณเป็นไปตามมาตรฐาน RS-232 ก่อนส่งออกไปจากคอมพิวเตอร์สำหรับอุปกรณ์ต่อเชื่อมปลายทางก็จะต้องมีวงจรขับในลักษณะนี้เช่นเดียวกัน เพื่อให้ได้ระดับสัญญาณในระดับเดียวกัน แต่วงจรขับที่ใช้ทั้งภายในคอมพิวเตอร์และอุปกรณ์ต่อเชื่อมปลายทางนั้นจะถูกกลับสถานะ ดังแสดงเป็นบล็อกไดอะแกรมรูปที่ 3-13

แอดเดรสของพอร์ตอนุกรม

แอดเดรสของพอร์ตอนุกรมมี 4 ตำแหน่งดังนี้คือ



รูปที่ 3-13 ไดอะแกรมแสดงโครงสร้างทางฮาร์ดแวร์ของพอร์ตอนุกรม

COM1 : 3F8H

COM2 : 2F8H

COM3 : 3E8H

COM4 : 2E8H

เมื่อเริ่มเปิดเครื่องเพื่อใช้งานคอมพิวเตอร์ ไบออสภายในคอมพิวเตอร์จะทำการตรวจสอบแอดเดรสของพอร์ตอนุกรมทั้งหมด ถ้าไบออสตรวจพบแอดเดรสของพอร์ตอนุกรม ไบออสจะนำแอดเดรสที่ตรวจพบไปเก็บไว้ในหน่วยความจำขนาด 2 ไบต์ สำหรับพอร์ตอนุกรม COM1 จะเก็บไว้ที่แอดเดรส 0000:0400H และ 0000:0401H ส่วนตำแหน่งอื่นๆมีรายละเอียดดังนี้

บิต 3	บิต 2	บิต 1	จำนวนพอร์ต
0	0	0	ไม่มีพอร์ตอนุกรม
0	0	1	มีพอร์ตอนุกรม 1 พอร์ต
0	1	0	มีพอร์ตอนุกรม 2 พอร์ต
0	1	1	มีพอร์ตอนุกรม 3 พอร์ต
1	0	0	มีพอร์ตอนุกรม 4 พอร์ต

ตารางที่ 3-4 แสดงข้อมูลในแอดเดรส 0000:0411H ที่ใช้แจ้งจำนวนพอร์ตอนุกรม

COM2 = 0000:0402H – 0000:0403H

COM3 = 0000:0404H – 0000:0405H

COM4 = 0000:0406H – 0000:0407H

นอกจากนี้ที่หน่วยความจำแอดเดรส 0000:0411H ยังใช้สำหรับแสดงจำนวนของพอร์ตอนุกรมที่มีใช้อยู่ในคอมพิวเตอร์อีกด้วยโดยมีรายละเอียดดังแสดงในตาราง

การส่งและอ่านข้อมูลผ่านพอร์ตอนุกรม RS-232

พอร์ตอนุกรมนอกจากจะมีขาสำหรับการรับและส่งข้อมูลปกติแล้วยังมีขาที่ออกแบบไว้สำหรับควบคุมการไหลของข้อมูลอีกหลายตำแหน่งด้วยกัน โดยแยกเป็น 2 ประเภทคือขาที่ทำหน้าที่เป็นเอาต์พุต

ขาสัญญาณเอาต์พุตของพอร์ตอนุกรม

ขาที่ทำหน้าที่เป็นขาเอาต์พุตได้แก่ ขา DTR, RTS และ TxD โดยรีจิสเตอร์ที่ทำหน้าที่ควบคุมขาเหล่านี้คือรีจิสเตอร์ควบคุมโมเด็ม (MCR) โดยมีแอดเดรสอยู่ถัดจากรีจิสเตอร์หลักของพอร์ตอนุกรม 4 ตำแหน่ง รีจิสเตอร์ควบคุมการทำงานของขา DTR จะอยู่ที่ตำแหน่งบิต 0 ส่วนขา RTS จะอยู่ที่ตำแหน่งบิต 1 ส่วนขา TxD เป็นขาปกติในการส่งข้อมูล ดังนั้นจึงมีแอดเดรสอยู่ที่แอดเดรสของรีจิสเตอร์หลัก

ระดับแรงดันที่ใช้งานสำหรับพอร์ตอนุกรม RS-232

มาตรฐานการสื่อสารข้อมูลของพอร์ตอนุกรม ได้ระบุช่วงระดับแรงดันสำหรับการทำงานของพอร์ตอนุกรมไว้ว่า ที่ลอจิก "0" จะมีระดับสัญญาณ +3 ถึง +15 ส่วนลอจิก "1" จะมีระดับสัญญาณ –3 ถึง –15V ระดับสัญญาณนี้ทำให้ไม่สามารถที่จะนำขาเอาต์พุตใดๆต่อเข้ากับลอจิกเกตเพื่อใช้งานได้โดยตรง จะต้องผ่านวงจรเพื่อเปลี่ยนระดับแรงดันเสียก่อน โดยปกติจะใช้ไอซีจำพวก RS-232 transceiver ที่นิยมมากคือ Max232 หรือ ICL232 ไอซีในกลุ่มนี้จะทำหน้าที่แปลงระดับแรงดันของ RS-232 ให้อยู่ในระดับทีทีแอล โดยลอจิก "0" ซึ่งเดิมมีระดับสัญญาณ +3 ถึง –15V จะแปลงเป็น +5V ทั้งนี้เพื่อให้สามารถต่อกับอุปกรณ์ดิจิตอลอื่นที่ใช้ระดับแรงดันทีทีแอลได้

รีจิสเตอร์ที่เกี่ยวข้องกับการทำงานของพอร์ตอนุกรมใน MCS-51

ในการทำงานของพอร์ตอนุกรมในไมโครคอนโทรลเลอร์ MCS-51 มีรีจิสเตอร์ที่เกี่ยวข้องอยู่ 2 ตัว ดังนี้

รีจิสเตอร์บัฟเฟอร์ของพอร์ตอนุกรมหรือ SBUF (Serial data buffer register)

มีแอดเดรสอยู่ที่ 99H ในพื้นที่ของรีจิสเตอร์ฟังก์ชั่นพิเศษหรือ SFR มีขนาด 8 บิต แบ่งเป็น 2 ส่วนคือ รีจิสเตอร์บัฟเฟอร์สำหรับส่งข้อมูล (transmit buffer register) และรับข้อมูล (receive buffer register) เมื่อมีการเขียนข้อมูลมายังรีจิสเตอร์ SBUF ข้อมูลนั้นจะถูกส่งต่อไปยังบัฟเฟอร์สำหรับส่งข้อมูล เพื่อส่งออกจากไมโครคอนโทรลเลอร์ผ่านทางขา TxD หรือขา P3.1 ในกรณีที่มีการอ่านข้อมูลจากรีจิสเตอร์ SBUF ข้อมูลจะถูกส่งผ่านไปยังรีจิสเตอร์บัฟเฟอร์สำหรับรับข้อมูลเพื่อส่งต่อไปยังไมโครคอนโทรลเลอร์ต่อไป สำหรับการรับข้อมูลอนุกรมจากภายนอกนั้นจะผ่านมาทางขา RxD หรือ P3.0 ของไมโครคอนโทรลเลอร์ MCS-51 แบบแฟลช

รีจิสเตอร์ควบคุมการทำงานของพอร์ตอนุกรมหรือ SCON (Serial port Control Register)

เป็นรีจิสเตอร์ขนาด 8 บิต มีแอดเดรสอยู่ที่ 98H ในพื้นที่ของรีจิสเตอร์ SFR สามารถเข้าถึงได้ในระดับบิตมีรายละเอียดการทำงานดังนี้

บิต 7	บิต 6	บิต 5	บิต 4	บิต 3	บิต 2	บิต 1	บิต 0
SM0	SM1	SM2	REN	TB8	RB8	TI	RI

SM0-SM1 (Serial port mode bit 0–1) : ใช้ในการเลือกโหมดการทำงานของพอร์ตอนุกรมภายในไมโครคอนโทรลเลอร์ MCS-51 ดังมีรายละเอียดต่อไปนี้

SM2 : ใช้ในการเอ็นเอเบิลการสื่อสารในแบบมัลติโปรเซสเซอร์ (multiprocessor) ในการทำงานของโหมด 2 และ 3 ของพอร์ตอนุกรมในไมโครคอนโทรลเลอร์ MCS-51 ถ้าบิตนี้เป็น "1" บิต RI จะไม่แอกตีฟถ้าบิตที่ 9 ที่รับเข้ามาเป็น "0" (ข้อมูลบิตที่ 9 เก็บไว้ที่บิต RB8) ในการทำงานโหมด 1 ถ้าบิตนี้เซต บิต RI จะไม่แอกตีฟถ้ายังไม่ได้รับบิตหยุด ส่วนในโหมด 0 บิตนี้ไม่มีการใช้งาน

REN (Enable serial reception) : ใช้ในการเอ็นเอเบิลการรับข้อมูลของพอร์ตอนุกรม ทำการเซตและเคลียร์ด้วยกระบวนการทางซอฟต์แวร์ ถ้าต้องการให้มีการรับข้อมูลต้องเซตบิตนี้ให้เป็น "1"

TB8 : ใช้สำหรับเก็บข้อมูลบิตที่ 9 ที่ต้องการส่งออกไปในการทำงานโหมด 2 และ 3 ของพอร์ตอนุกรมในไมโครคอนโทรลเลอร์ MCS-51 เซตและเคลียร์ด้วยกระบวนการทางซอฟต์แวร์

RB8 : ใช้สำหรับรับข้อมูลบิตที่ 9 ที่เข้ามาในการทำงานโหมด 2 และ 3 ของพอร์ตอนุกรมในไมโครคอนโทรลเลอร์ MCS-51 แต่ถ้าหากพอร์ตอนุกรมทำงานอยู่ในโหมด 1 และบิต SM2 เป็น "0" ข้อมูลที่บิต RB8 คือข้อมูลของบิตหยุด สำหรับในการทำงานโหมด 0 บิตนี้จะไม่ใช้งาน บิต RB8 นี้สามารถเซตและเคลียร์ด้วยกระบวนการทางซอฟต์แวร์

TI (Transmit Interrupt flag) : ใช้ในการแสดงการเกิดอินเตอร์รัปต์เมื่อมีการส่งข้อมูลออกจากพอร์ตอนุกรมของไมโครคอนโทรลเลอร์ MCS-51สามารถเซตได้ด้วยกระบวนการทางฮาร์ดแวร์เมื่อทำการส่งข้อมูลบิตที่ 8 ไปเรียบร้อยแล้วในการทำงานโหมด 0 ส่วนในการทำงานโหมดอื่น บิตนี้จะเซตเมื่อมีการเริ่มต้นส่งบิตหยุดออกไป การเคลียร์บิตนี้ต้องใช้กระบวนการทางซอฟต์แวร์เท่านั้น

RI (Receive Interrupt flag) : ใช้แสดงการเกิดอินเตอร์รัปต์เมื่อมีการรับข้อมูลเข้าสู่พอร์ตอนุกรม สามารถเซตได้ด้วยกระบวนการทางฮาร์ดแวร์ เมื่อทำการรับข้อมูลบิตที่ 8 เรียบร้อยแล้วในการทำงานโหมด 0 ส่วนในการทำงานโหมดอื่น บิตนี้จะเซตเมื่อสามารถรับบิตหยุดของข้อมูลอนุกรมไปได้ครึ่งทางแล้ว ยกเว้นในกรณีที่บิต SM2 มีการเซต บิตนี้จะเซตได้ก็ต่อเมื่อการรับบิตหยุดหรือบิตที่ 9 เกิดขึ้นอย่างสมบูรณ์แล้ว การเคลียร์บิตนี้ต้องใช้กระบวนการทางซอฟต์แวร์เท่านั้น

3.2.5 โหมดการทำงานของพอร์ตอนุกรมใน MCS-51

พอร์ตอนุกรมในไมโครคอนโทรลเลอร์ MCS-51 สามารถเลือกการทำงานได้ถึง 4 โหมดคือ

1.โหมด 0 เป็นการกำหนดให้พอร์ตอนุกรมทำงานในลักษณะชิฟต์รีจิสเตอร์

2.โหมด 1 เป็นการกำหนดให้เป็น UART ขนาด 8 บิต สามารถเลือกอัตราบอดได้

3.โหมด 2 เป็นการกำหนดให้เป็น UART ขนาด 9 บิต โดยมีอัตราบอดคงที่

4.โหมด 3 เป็นการกำหนดให้เป็น UART ขนาด 9 บิต สามารถเลือกอัตราบอดได้

การเลือกโหมดทำได้ด้วยการกำหนดข้อมูลให้แก่บิต SM0 และ SM1 ในรีจิสเตอร์ SCON

การทำงานในโหมด 0 ของวงจรพอร์ตอนุกรม

มีไดอะแกรมการทำงานและไดอะแกรมเวลาแสดงในรูปที่ 2 ข้อมูลอนุกรมจะผ่านเข้าและออกทางขา RxD ส่วนขา TxD ทำหน้าที่เป็นสัญญาณนาฬิกาของการเลื่อนข้อมูล (shift clock) ในโหมดนี้มีจำนวนข้อมูล 8 บิต โดยทำการรับและส่งข้อมูลในบิต LSB ก่อน อัตราในการรับส่งข้อมูลหรืออัตราบอดถูกกำหนดไว้คงที่ที่ 1/12 ของความถี่สัญญาณนาฬิกา

เริ่มต้นการส่งข้อมูลด้วยการเขียนข้อมูลที่ต้องการส่งมายังรีจิสเตอร์ SBUF สัญญาณเขียนข้อมูล แอกตีฟเป็น "1" ที่สเตต 6 เฟส 2 (S6P2) ของแมชีนไซเกิล ส่งมายังวงจรควบคุมการส่ง (TX control) ทำให้วงจรควบคุมเริ่มต้นส่งข้อมูล สัญญาณ SEND จะแอกตีฟเป็น "1" ตลอดการส่งข้อมูล

ข้อมูลจากรีจิสเตอร์ SBUF จะถูกเลื่อนออกที่ขา P3.0 หรือขา RxD ครั้งละบิต ตามจังหวะของสัญญาณนาฬิกาที่ส่งออกมาทางขา P3.1 หรือ TxD โดยสัญญาณนาฬิกาของการเลื่อนข้อมูลจะมีขอบขาลงของสัญญาณที่สเตต 3 เฟส และมีขอบขาขึ้นของสัญญาณที่สเตต 6 เฟส 1 ของแต่ละแมชีนไซเกิลในกระบวนการส่งข้อมูลจนกระทั่งเมื่อส่งข้อมูลครบ 8 บิตแล้ว บิต TI ในรีจิสเตอร์ SCON จะเกิดการเซต เป็นการแจ้งให้ทราบว่าส่งข้อมูลครบแล้ว หากการอินเตอร์รัปต์จากพอร์ตอนุกรมได้รับการเอ็นเอเบิลไว้ ก็จะเกิดการอินเตอร์รัปต์ขึ้นในระบบ เมื่อเสร็จสิ้นกระบวนการรับข้อมูล สัญญาณ SEND จะกลายเป็น "0" จนกว่าจะเริ่มต้นกระบวนการรับข้อมูลใหม่

ในกระบวนการรับข้อมูลเริ่มต้นด้วยการเซต REN ให้เป็น "1" และเคลียร์บิต ในรีจิสเตอร์ SCON ก่อนที่สเตต 6 เฟส 2 ของแมชีนไซเกิลถัดไป วงจรควบคุมการรับ (RX control) จะทำการเขียนข้อมูล 11111110 ไปยังชิฟต์รีจิสเตอร์สำหรับรับข้อมูลและทำการแอกตีฟสัญญาณ RECEIVE ให้เป็น "1" ในสัญญาณนาฬิกาลูกถัดไป

เมื่อสัญญาณ RECEIVE แอกตีฟ ก็จะเกิดการส่งสัญญาณนาฬิกาของการเลื่อนข้อมูล (Shift clock) ขึ้นผ่านทางขา P3.1 หรือ TxD เพื่อทำการกำหนดจังหวะการรับข้อมูลครั้งละบิต

โดยสัญญาณนาฬิกานี้จะเกิดขึ้นในช่วงสเตต 3 เฟส 1 ถึงสเตต 6 เฟส 1 ของแต่ละแมชีนไซเกิล การรับข้อมูลเข้ามาทางขา P3.0 หรือ RxD จะเกิดขึ้นที่สเตต 5 เฟส 2 ในแมชีนไซเกิลเดียวกับสัญญาณนาฬิกาของการเลื่อนข้อมูล จนกระทั่งรับข้อมูลครบทั้ง 8 บิต บิต RI จะได้รับการเซตเพื่อแจ้งการเสร็จสิ้นกระบวนการรับข้อมูล หากการอินเตอร์รัปต์จากพอร์ตอนุกรมได้รับการเอ็นเอเบิลไว้ก็จะเกิดการอินเตอร์รัปต์ขึ้นในระบบ เมื่อเสร็จสิ้นกระบวนการรับข้อมูล สัญญาณ RECEIVE จะกลายเป็น "0" จนกว่าจะเริ่มต้นกระบวนการรับข้อมูลใหม่

การทำงานในโหมดนี้ของพอร์ตอนุกรมในไมโครคอนโทรลเลอร์ MCS-51 จะใช้ในการเชื่อมต่อกับไอซีรีจิสเตอร์ภายนอกเพื่อทำการขยายจำนวนพอร์ตอินพุตหรือเอาต์พุต แต่ไม่เป็นที่นิยมใช้งานมากนัก เนื่องจากในไมโครคอนโทรลเลอร์ MCS-51 เองมีพอร์ตอยู่ค่อนข้างมาก และติดต่อกับพอร์ตเหล่านั้นได้ง่ายและเร็วกว่ามาก

การทำงานในโหมด 1 ของวงจรพอร์ตอนุกรม

มีไดอะแกรมแสดงในรูปที่ 3 ในโหมดนี้ใช้ในการรับส่งข้อมูลรวม 10 บิต โดยส่งข้อมูลออกทางขา P3.1 หรือ TxD และรับข้อมูลเข้าทางขา P3.0 หรือ RxD ข้อมูลทั้ง 10 บิตประกอบด้วย บิตเริ่มต้น (มีค่าเป็น "0") 1 บิต บิตข้อมูล 8 บิต โดยรับหรือส่งข้อมูลในบิต LSB ก่อน และบิตหยุดหรือบิตปิดท้าย (มีค่าเป็น "1") ในการรับข้อมูล บิตหยุดจะถูกเก็บไว้ในบิต RB8 ในรีจิสเตอร์ SCON อัตราบอดในโหมดนี้ได้รับการกำหนดโดยอัตราการเกิดโอเวอร์โฟลวของไทเมอร์ 1 ใน AT89C51 ส่วนในไมโครคอนโทรลเลอร์เบอร์ AT89C52 และในอนุกรม AT89Sxx สามารถเลือกใช้อัตราการเกิดโอเวอร์โฟลวของไทเมอร์ 1 หรือไทเมอร์ 2 ในการกำหนดอัตราบอดได้

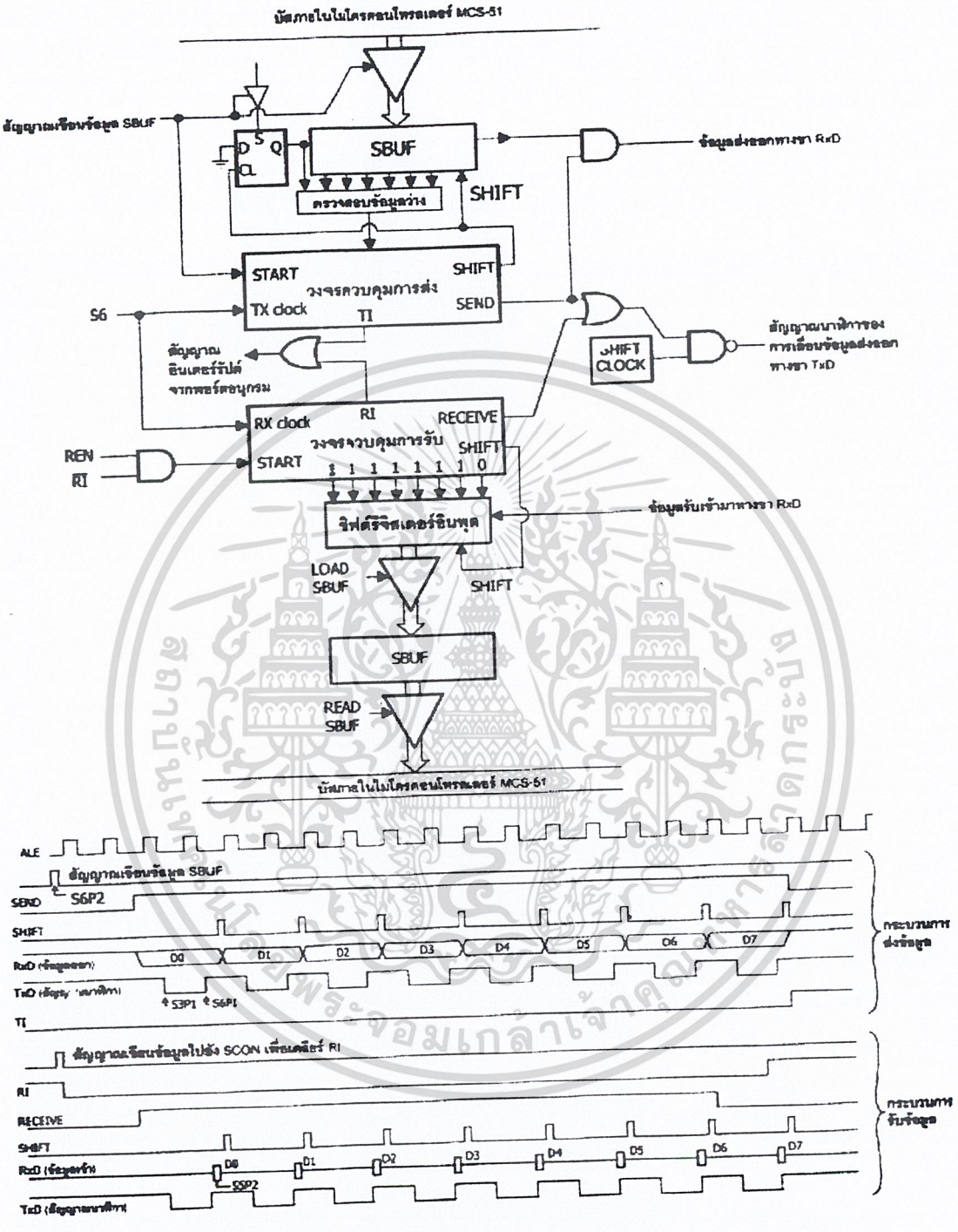
กระบวนการส่งข้อมูลเริ่มต้นด้วยการแอกตีฟสัญญาณเขียนข้อมูลมายังรีจิสเตอร์ SBUF ส่งมายังวงจรควบคุมการส่ง (TX control) จากนั้นวงจรควบคุมจะทำการแอกตีฟสัญญาณ SEND ที่สเตต 1 เฟส 1 ของแมชีนไซเกิลต่อมา โดยสัญญาณ SEND จะเป็น "0" ตลอดการส่งข้อมูล เมื่อสัญญาณ SEND แอกตีฟ จะทำการส่งบิตเริ่มต้นก่อนเป็นบิตแรก โดยมีคาบเวลาของบิตเริ่มต้นเท่ากับ1 แมชีนไซเกิล จากนั้นตามด้วยการส่งบิตข้อมูล 8 บิต เรียงลำดับจากบิต LSB โดยข้อมูลที่ทำการส่งถูกเรียกออกมาจากรีจิสเตอร์บัฟเฟอร์สำหรับการส่งข้อมูล ในทุก ๆ บิตข้อมูลที่ทำการส่งออกไป จะเกิดสัญญาณพัลส์ SHIFT ขึ้น เพื่อให้เรียกข้อมูลในแต่ละบิตจากรีจิสเตอร์บัฟเฟอร์ การกำหนดจังหวะการส่งข้อมูลใช้สัญญาณนาฬิกาการส่ง (TX clock) เป็นตัวกำหนด โดยสัญญาณนาฬิกานี้ได้มาจากการหารสัญญาณ TCLK จากไทเมอร์ 1 ด้วย 16 หลังจากการส่งบิตข้อมูลก็จะทำการส่งบิตหยุดหรือบิตปิดท้าย 1 บิต ดังนั้นการส่งข้อมูลจะใช้สัญญาณนาฬิกาทั้ง

หมด 10 ลูก เมื่อทำการส่งข้อมูลครบเรียบร้อยแล้ว จะทำการเซตบิต TI ในรีจิสเตอร์ SCON หากการอินเตอร์รัปต์จากพอร์ตอนุกรมได้รับการเอ็นเอเบิลไว้ก็จะเกิดการอินเตอร์รัปต์ขึ้นในระบบ หลังจากที่ทำการบริการอินเตอร์รัปต์หรือส่งข้อมูลเรียบร้อยแล้ว ต้องทำการเคลียร์บิต TI ก่อนเป็นอันดับแรก เพื่อให้การรับส่งข้อมูลทางพอร์ตอนุกรมดำเนินต่อไปได้

ด้านการรับข้อมูล จะทำการตรวจจับการเปลี่ยนแปลงระดับลอจิกจาก “1” เป็น “0” ที่ขา RxD โดยใช้อัตราการสุ่มเท่ากับ 1/16 เท่าของอัตราบอด เมื่อตรวจจับพบ ไทเมอร์/เคาน์เตอร์ที่ใช้ในการกำหนดอัตราบอดจะรีเซตและทำการเขียนข้อมูล 1FFH ไปยังชิฟต์รีจิสเตอร์ ข้อมูลจะเริ่มเดินทางเข้าสู่พอร์ตอนุกรมของไมโครคอนโทรลเลอร์ผ่านทางขา RxD ในการตีความว่าบิตที่เข้ามาเป็น “0” หรือ “1” จะใช้ผลการสุ่มข้างมาก โดยบิตของข้อมูลที่เข้ามาได้รับการแบ่งออกเป็น 16 สเตต การสุ่มข้อมูลจะทำการสุ่มสเตตที่ 7,8 และ 9 หาก 2 ใน 3 ของการสุ่มพบว่าข้อมูลเป็นลอจิกใด จะตีความข้อมูลในบิตนั้นเป็นตามเสียงข้างมาก ยกตัวอย่าง สุ่มพบลอจิก “1” 2 ใน 3 ครั้ง จะตีความว่าบิตของข้อมูลที่รับได้นั้นเป็น “1”

ลำดับของการรับข้อมูลมีลักษณะเดียวกับการส่งข้อมูลคือ เริ่มด้วยบิตเริ่มต้นก่อน ตามด้วยบิตข้อมูล และบิตปิดท้าย ในทุก ๆ การรับข้อมูลได้ 1 บิต จะมีพัลส์ SHIFT เกิดขึ้น เพื่อทำการเลื่อนข้อมูลเข้าสู่รีจิสเตอร์บัฟเฟอร์การรับข้อมูล การกำหนดจังหวะการรับข้อมูลใช้สัญญาณนาฬิกาการรับข้อมูล (RX clock) หลังจากสัญญาณนาฬิกาลูกสุดท้าย อันหมายถึงสามารถรับข้อมูลได้ครบแล้ว วงจรควบคุมการรับข้อมูลจะทำการส่งข้อมูลจากรีจิสเตอร์บัฟเฟอร์ไปยังรีจิสเตอร์ SBUF และบิต RB8 ในรีจิสเตอร์ SCON โดยข้อมูลในบิต RB8 ก็คือข้อมูลของบิตหยุดนั่นเอง พร้อมกันนั้นยังทำการเซตบิต RI ในรีจิสเตอร์ SCON ด้วย หากการอินเตอร์รัปต์จากพอร์ตอนุกรมได้รับการเอ็นเอเบิลไว้ก็จะเกิดการอินเตอร์รัปต์ขึ้นในระบบ หลังจากบริการอินเตอร์รัปต์หรือรับข้อมูลเรียบร้อยแล้ว ต้องทำการเคลียร์บิต RI ก่อน เพื่อให้การรับส่งข้อมูลทางพอร์ตอนุกรมดำเนินต่อไปได้

การทำงานในโหมดนี้ได้รับความนิยมสูงสุด เนื่องจากมีกระบวนการที่ไม่ซับซ้อนและสามารถทำการรับส่งข้อมูลกับคอมพิวเตอร์ได้อย่างมีประสิทธิภาพ



รูปที่ 3-14 ไดอะแกรมการทำงานในโหมด 0 ของพอร์ตอนุกรมภายในไมโครคอนโทรลเลอร์MCS-51

การทำงานในโหมด 2 และ 3 ของวงจรพอร์ตอนุกรม

ในทั้งสองโหมดนี้จะใช้รูปแบบข้อมูลรวม 11 บิต ประกอบด้วยบิตเริ่มต้น มีค่าเป็น "0" จำนวน 1 บิต, บิตข้อมูล 8 บิต โดยทำการรับและส่งบิต LSB ก่อน, บิตข้อมูลบิตที่ 9 และบิตปิดท้ายมีค่าเป็น "1" จำนวน 1 บิต ในการส่งข้อมูล ข้อมูลบิตที่ 9 จะได้รับการเก็บไว้ในบิต TB8 ในรีจิสเตอร์ SCON และในการรับข้อมูล ข้อมูลบิตที่ 9 จะนำไปเก็บไว้ในบิต RB8 ในรีจิสเตอร์ SCON สำหรับอัตราบอดในโหมด 2 จะคงที่โดยเลือกได้ 2 ค่าคือ 1/32 หรือ 1/64 ของความถี่สัญญาณนาฬิกา สำหรับในโหมด 3 อัตราบอดสามารถปรับได้เหมือนกับในโหมด 1

ในรูปที่ 4 และ 5 เป็นไดอะแกรมการทำงานและไดอะแกรมของการทำงานในโหมด 2 และ 3 ของพอร์ตอนุกรม การทำงานโดยรวมจะคล้ายกับการทำงานในโหมด 1 ส่วนที่แตกต่างกันคือจำนวนบิตของข้อมูลที่ในโหมด 2 และ 3 จะมีเพิ่มมาอีก 1 บิต โดยส่วนใหญ่จะใช้เป็นบิตตรวจสอบพาริตี้

อัตราบอดของพอร์ตอนุกรมในไมโครคอนโทรลเลอร์ MCS-51

โหมด 0

อัตราบอดของโหมดนี้มีค่าคงที่ โดยสามารถคำนวณได้จากสูตร

อัตราบอดในโหมด 0 = ความถี่ของสัญญาณนาฬิกา /12 หน่วยเป็น บิตต่อวินาที

โหมด 1 และ 3

เนื่องจากทั้งสองโหมดนี้สามารถเลือกแหล่งกำเนิดอัตราบอดได้ 2 แหล่งคือ จากอัตราโอเวอร์โฟลวของไทเมอร์ 1 และ 2 สำหรับอัตราบอดเมื่อใช้การโอเวอร์โฟลวของไทเมอร์ 1 จะต้องใช้ค่าของบิต SMOD ในรีจิสเตอร์ PCON มาพิจารณาประกอบด้วย สามารถคำนวณค่าอัตราบอดได้จาก

อัตราบอด = $(2^{\text{ค่าของบิต SMOD}}/32)$ x อัตราโอเวอร์โฟลวของไทเมอร์ 1

ถ้าหากในไทเมอร์ 1 ไม่ได้เอ็นเอเบิลการอินเตอร์รัปต์ไว้ สามารถคำนวณค่าอัตราบอดได้จาก

อัตราบอด = $(2^{\text{ค่าในรีจิสเตอร์ SMOD}}/32)$ x (ความถี่สัญญาณนาฬิกา / {12x[256 – (TH1)]})

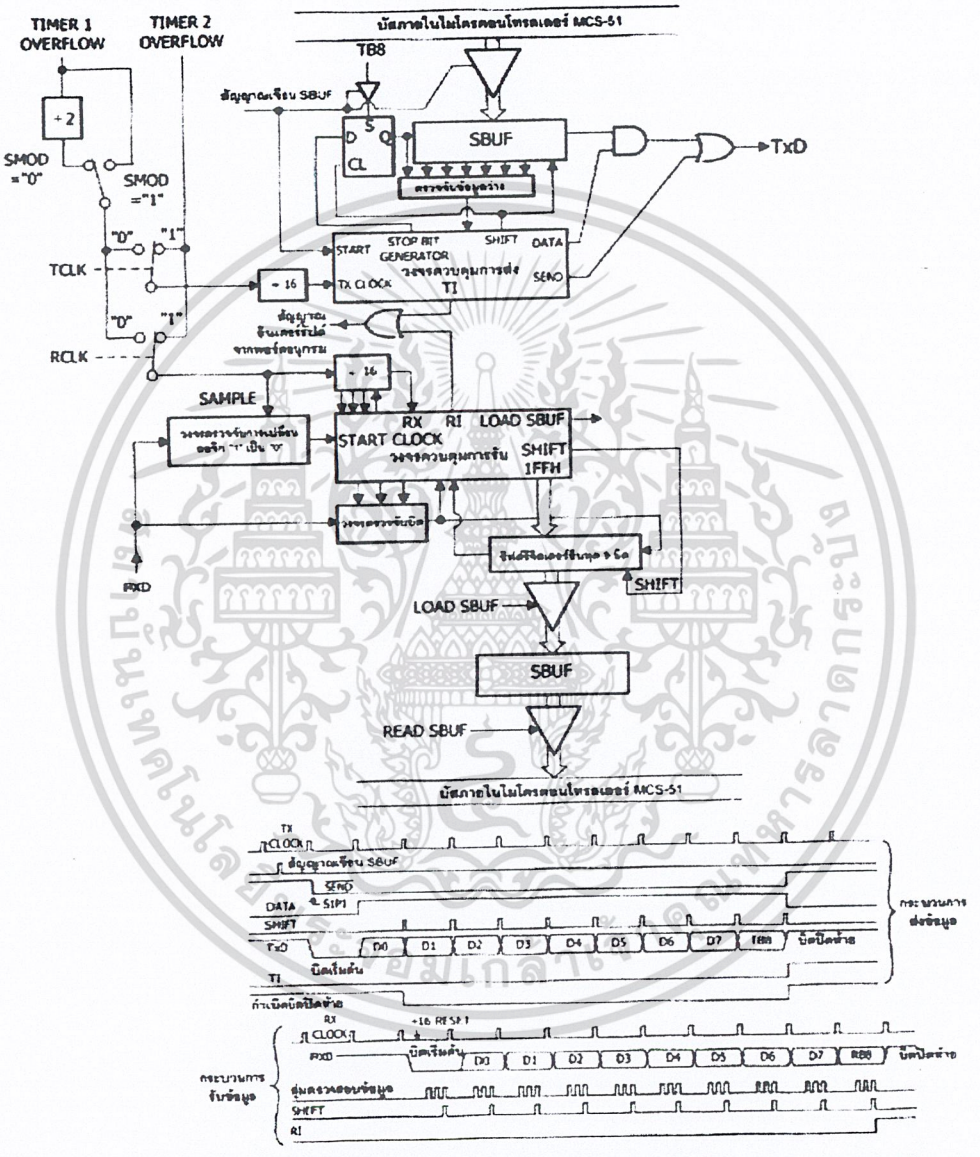
ในตารางที่ 1 แสดงการกำหนดอัตราบอดโดยใช้ไทเมอร์ 2 ทำงานในโหมดกำเนิดอัตราบอด (baud rate generator) สามารถคำนวณหาค่าอัตราบอดได้จาก

อัตราบอด = อัตราโอเวอร์โฟลวของไทเมอร์ 2/16 หน่วยเป็น บิตต่อวินาที

ถ้าหากกำหนดให้ไทเมอร์ 2 ทำงานในโหมดปกติ สามารถคำนวณหาค่าอัตราบอดได้จาก

อัตราบอด = ความถี่ของสัญญาณนาฬิกา / (32 x (65536 – (RCAP2H,RCAP2L)))

โดยที่ (RCAP2H,RCAP2L) เป็นค่าของรีจิสเตอร์ RCAP2H และ RCAP2L มีขนาด 16 บิตไม่คิดเครื่องหมาย



รูปที่ 3-15 ไดอะแกรมการทำงานในโหมด 1 ของพอร์ตอนุกรมภายในไมโครคอนโทรลเลอร์ MCS-51

โหมด 2

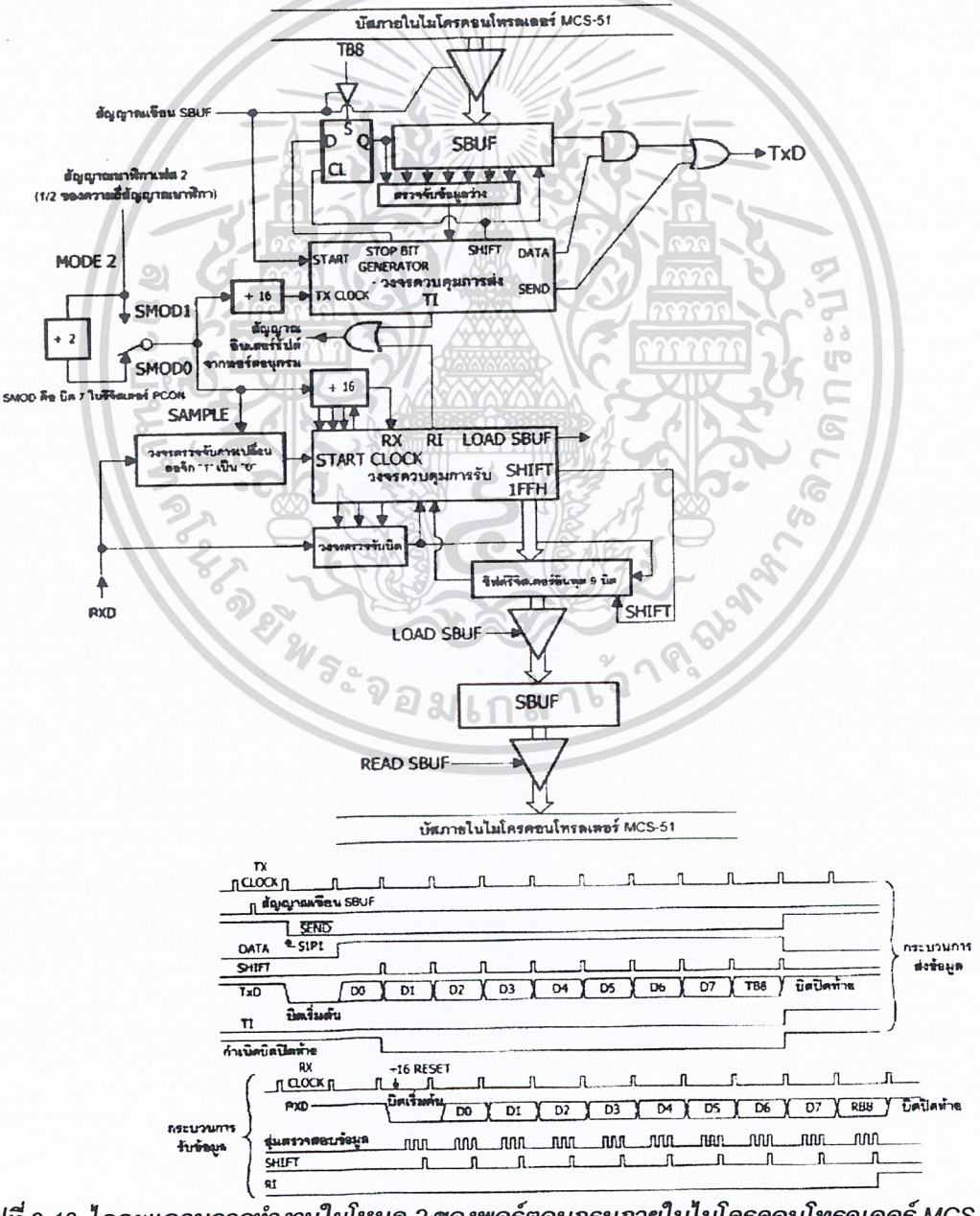
ในโหมดนี้อัตราบอดจะขึ้นอยู่กับค่าของบิต SMOD ในรีจิสเตอร์ PCON ถ้า SMOD เป็น "0" อัตราบอดจะเท่ากับ 1/64 ของความถี่สัญญาณนาฬิกา ในกรณีที่ SMOD เป็น "1" อัตราบอด

จะเท่ากับ 1/32 ของความถี่สัญญาณนาฬิกา สามารถแสดงเป็นสูตรคำนวณทางคณิตศาสตร์ได้ดังนี้

อัตราบอด = ($2^{\text{ค่าของบิต SMOD}}$ /64) x ความถี่สัญญาณนาฬิกา

การกำหนดค่าของไทเมอร์เพื่อเลือกอัตราบอด

ในการใช้งานพอร์ตอนุกรมของไมโครคอนโทรลเลอร์ MCS-51 สิ่งที่ต้องให้ความสนใจมากที่สุดประการหนึ่งคือ อัตราการถ่ายทอดข้อมูล หรือ อัตราบอด ซึ่งการกำหนดอัตราบอดนั้นจะขึ้น



รูปที่ 3-16 ไดอะแกรมการทำงานในโหมด 2 ของพอร์ตอนุกรมภายในไมโครคอนโทรลเลอร์ MCS-51

อยู่กับค่าความถี่ของสัญญาณนาฬิกาเป็นหลัก สำหรับโหมดการทำงานของพอร์ตอนุกรมที่สามารถเลือกอัตราบอดได้อย่างอิสระคือในโหมด 1 และ 3 โดยกำหนดได้จากอัตราการเกิดโอเวอร์โฟลวของไทเมอร์ 1 ถ้าหากไทเมอร์ 1 มีการเกิดโอเวอร์โฟลวในอัตราที่สูงมากเท่าใด อัตราบอดก็จะมีค่าสูงมากขึ้นตาม นั่นหมายความว่า อัตราในการถ่ายทอดข้อมูลจะสูงมาก สามารถถ่ายทอดข้อมูลได้อย่างรวดเร็ว

ในการใช้ไทเมอร์ 1 เพื่อกำหนดอัตราบอดในโหมด 1 และ 3 ของพอร์ตอนุกรมจะต้องกำหนดให้ไทเมอร์ 1 ทำงานในโหมด 2 หรือโหมด 8 บิตแบบตั้งค่าการนับอัตโนมัติ และการกำหนดค่ารีโหลดให้แก่รีจิสเตอร์ TH1 จึงเป็นตัวแปรหลักที่ใช้ในการกำหนดอัตราบอดให้แก่พอร์ตอนุกรมของไมโครคอนโทรลเลอร์ MCS-51

เริ่มต้นด้วยการเคลียร์บิต SMOD ซึ่งเป็นบิต 7 ของรีจิสเตอร์ PCON ให้เป็น “0” ค่าของการรีโหลดให้แก่ TH1 สามารถคำนวณได้จาก

TH1 = 256 – ((ค่าความถี่ของคริสตอล /384) /อัตราบอด)

แต่ถ้าบิต SMOD เกิดการเซต จะเป็นการเอ็นเอเบิลการทวีคูณของอัตราบอด ดังนั้นการกำหนดค่าให้แก่ TH1 จึงต้องคำนวณจาก

TH1 = 256 – ((ค่าความถี่ของคริสตอล /192) /อัตราบอด)

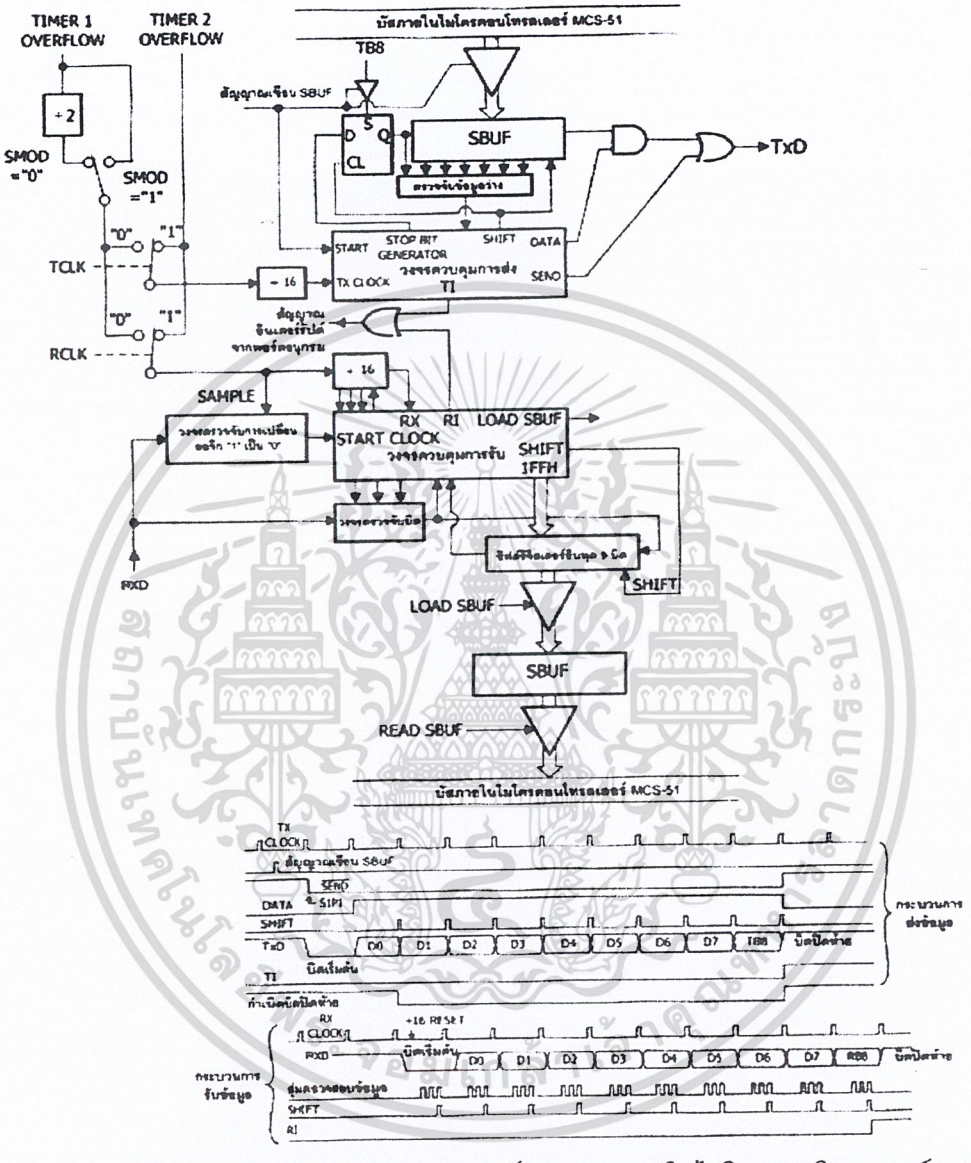
ยกตัวอย่าง ถ้าหากในไมโครคอนโทรลเลอร์ AT89C51 ใช้คริสตอล 11.0592 MHz ต้องการกำหนดอัตราบอดของพอร์ตอนุกรมของไมโครคอนโทรลเลอร์ไว้ที่ 19,200 บิตต่อวินาที ในกรณีที่ไม่เอ็นเอเบิล การทวีคูณของอัตราบอด ค่ารีโหลดของไมโครคอนโทรลเลอร์จะเท่ากับ

TH1 = 256 – ((ค่าความถี่ของคริสตอล /384) /อัตราบอด)

= 256 – ((11059200/384) /19200)

= 256 – (28800/19200)

= 256 – 1.5 = 254.5



รูปที่ 3-17 ไดอะแกรมการทำงานในโหมด 3 ของพอร์ตอนุกรมภายในไมโครคอนโทรลเลอร์ MCS-51

เนื่องจากผลลัพธ์ที่ได้เป็นค่าที่ไม่ใช่จำนวนเต็ม ถ้าหากกำหนดค่าของ TH1 เป็น 254 เมื่อทำการแทนค่าเพื่อคำนวณหาค่าอัตราบอด จะได้อัตราบอดเท่ากับ 14,400 บิตต่อวินาที และถ้าหากกำหนดค่าของ TH1 เป็น 255 อัตราบอดจะมีค่าเท่ากับ 28,800 บิตต่อวินาที ดังนั้นจะเห็นได้ค่าของ TH1 ที่ไม่เป็นจำนวนเต็มจะไม่สามารถทำให้เกิดอัตราบอดตามที่ต้องการได้

ทางแก้ไขคือ ให้ทำการเอ็นเอเบิลการทวีคูณอัตราบอด โดยการเซตบิต SMOD ในรีจิสเตอร์ PCON ให้เป็น "1" จากนั้นแทนค่าลงในสมการหาค่า TH1 เมื่อมีการเซตบิต SMOD ได้ผลดังนี้

TH1 = 256 – ((ค่าความถี่ของคริสตอล /192) /อัตราบอด)

= 256 – ((11059200/192) / 19200)

= 256 – (57600/19200)

= 256 – 3 = 253

นำค่าของ TH1 ที่ได้ทำการแทนค่าคำนวณหาค่าอัตราบอดจะได้เท่ากับ 19,200 บิตต่อวินาที

สามารถสรุปขั้นตอนในการเลือกอัตราบอดโดยการกำหนดค่าของไทเมอร์ 1 ได้ดังนี้

อัตราบอด (บิตต่อวินาที : bps)	ความถี่ สัญญาณนาฬิกา	SMOD	ไทเมอร์ 1		
			C/T	โหมด	ค่ารีโหลด
โหมด 0 : สูงสุด 1 Hz	12 MHz	X	X	X	X
โหมด 2 : สูงสุด 375K	12 MHz	1	X	X	X
โหมด 1,3 : 62.5K	12 MHz	1	0	2	FFH
19.2K (19,200)	11.0592 MHz	1	0	2	FDH
9.6K (9,600)	11.0592 MHz	0	0	2	FDH
4.8K (4,800)	11.0592 MHz	0	0	2	FAH
2.4K (2,400)	11.0592 MHz	0	0	2	F4H
1.2K (1,200)	11.0592 MHz	0	0	2	E8H
137.5	11.0592 MHz	0	0	2	1DH
110	6 MHz	0	0	2	72H
110	12 MHz	0	0	1	FEEBH

ตารางที่ 3-5 การเลือกอัตราบอดของวงจรพอร์ตอนุกรมภายในไมโครคอนโทรลเลอร์ MCS-51

1.กำหนดให้พอร์ตอนุกรมของไมโครคอนโทรลเลอร์ MCS-51 ทำงานในโหมด 1 หรือ 3

2.กำหนดให้ไทเมอร์ 1 ทำงานในโหมด 2 หรือโหมด 8 บิตตั้งค่าอัตโนมัติ

3.กำหนดข้อมูลให้แก่ TH1 เท่ากับ 253 เพื่อให้สามารถกำเนิดอัตรายอดได้ 19,200 บิตต่อวินาที ตามที่ต้องการ

4.ทำการเซตบิต SMOD ซึ่งเป็นบิต 7 ของรีจิสเตอร์ PCON เพื่อเอ็นเอเบิลการทวีคูณอัตราบอด

3.2.7 การเขียนหรือส่งข้อมูลออกจากพอร์ตอนุกรม

ข้อมูลที่ต้องการส่งออกทุกค่าต้องนำไปเก็บไว้รีจิสเตอร์บัฟเฟอร์ของพอร์ตอนุกรม ซึ่งก็คือรีจิสเตอร์ SBUF ดังตัวอย่าง

```
MOV  SBUF , # ' A'
```

จากคำสั่งข้างต้นเป็นการส่งข้อมูลของตัวอักษร A ออกไปยังพอร์ตอนุกรมของไมโครคอนโทรลเลอร์ อย่างไรก็ตามก่อนทำการส่งข้อมูลทุกครั้งต้องแน่ใจว่าบิต TI เคลียร์หรือมีค่าเป็น "0" และเมื่อทำการส่งข้อมูลเรียบร้อยแล้ว ก็จะเกิดการเซตบิต TI เพื่อแจ้งให้ทราบ ดังตัวอย่างโปรแกรมต่อไปนี้

```
CLR    TI            ; เคลียร์บิต TI เพื่อเตรียมการส่งข้อมูลออก
MOV    SBUF , # 'A'  ; ส่งข้อมูลของตัวอักษร A ไปยังพอร์ตอนุกรม
JNB    TI , $        ; รอการเซตของบิต TI เพื่อแจ้งการส่งข้อมูลที่เสร็จสมบูรณ์
```

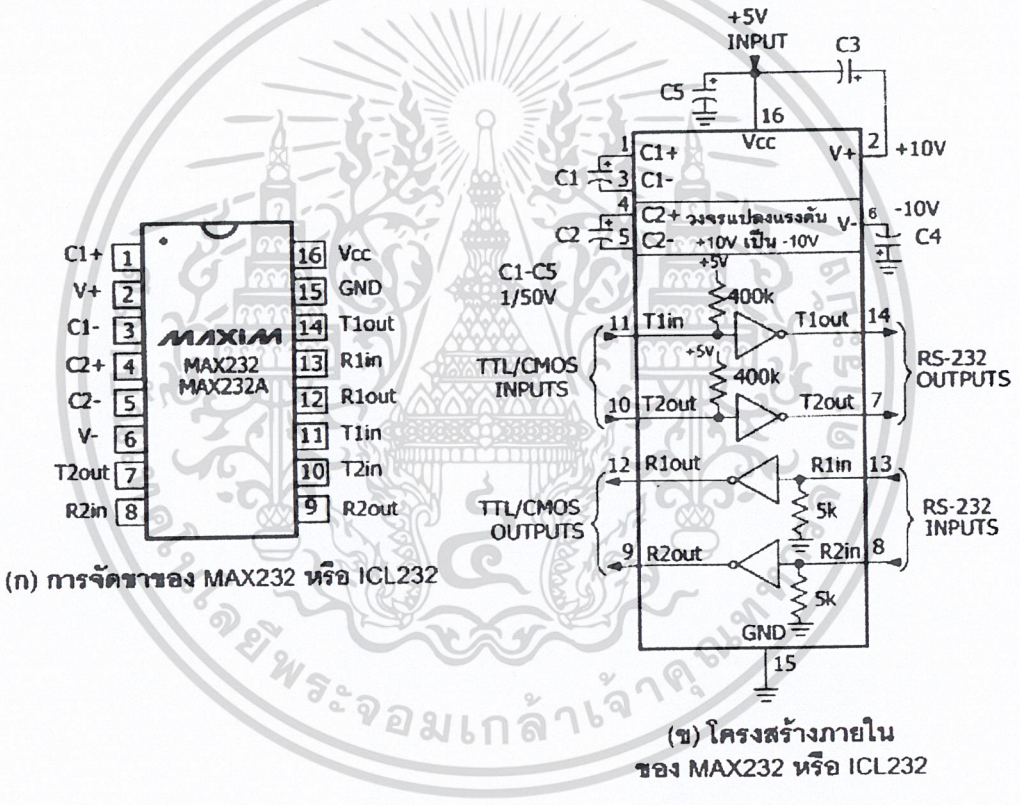
การอ่านหรือรับข้อมูลจากพอร์ตอนุกรม

การรับข้อมูลจากพอร์ตอนุกรมสามารถกระทำได้ง่ายมาก เพียงทำการตรวจสอบว่าบิต RI เกิดการเซตขึ้นหรือไม่ ถ้าพบว่ามีการเซตเกิดขึ้นแล้ว ให้ทำการอ่านค่าจากรีจิสเตอร์ SBUF โดยต้องทำการโอนย้ายข้อมูลผ่านทางแอกคิวมูเลเตอร์หรือรีจิสเตอร์ A ดังตัวอย่าง

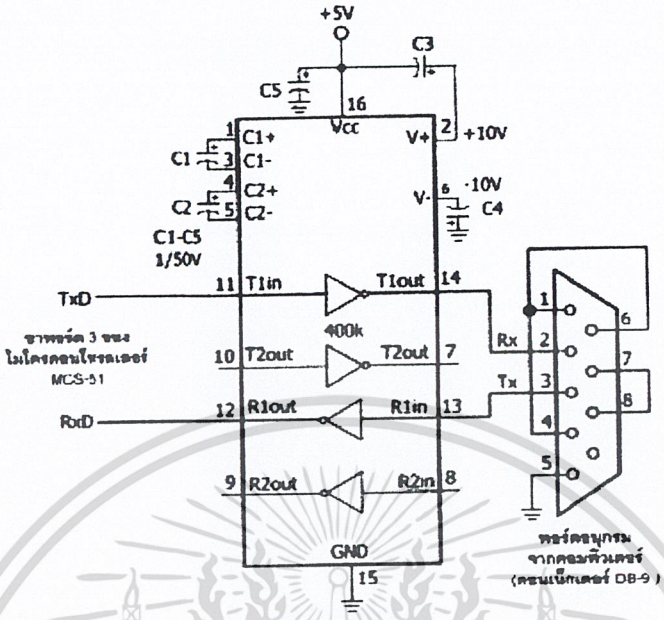
```
CLR    RI            ; เคลียร์บิต TI เพื่อเตรียมการส่งข้อมูลออก
JNB    RI , $        ; รอคอยการเซตของบิต RI อันเป็นการแจ้งให้ทราบว่า การรับข้อ
                       มูลเสร็จ
                     ; สมบูรณ์และมีข้อมูลเกิดขึ้นที่รีจิสเตอร์ SBUF
MOV    A , SBUF      ; อ่านค่าจากรีจิสเตอร์โดยการโอนย้ายข้อมูลผ่านทางรีจิสเตอร์A
CLR    RI            ; หลังจากทำการอ่านข้อมูลเรียบร้อยแล้ว ต้องทำการเคลียร์บิต
                       RI เสมอ
```

การเชื่อมต่อกับพอร์ตอนุกรมของคอมพิวเตอร์

การใช้งานวงจรพอร์ตอนุกรมของไมโครคอนโทรลเลอร์ MCS-51 มักนิยมใช้ในการติดต่อเพื่อแลกเปลี่ยนข้อมูลกับคอมพิวเตอร์ผ่านทางพอร์ตอนุกรมในมาตรฐาน RS-232 เป็นส่วนใหญ่ แต่เนื่องจากระดับสัญญาณของพอร์ตอนุกรม RS-232 มีระดับตั้งแต่ ±3 ถึง ±12V ในขณะที่ระดับสัญญาณของไมโครคอนโทรลเลอร์ MCS-51 อยู่ในระดับทีทีแอล ดังนั้นจึงไม่สามารถเชื่อมต่อพอร์ตอนุกรมของไมโครคอนโทรลเลอร์ MCS-51 เข้ากับพอร์ตอนุกรมของคอมพิวเตอร์ได้โดยตรง จึงต้องอาศัยการเชื่อมต่อผ่านไอซีพิเศษที่ทำหน้าที่ในการแปลงระดับสัญญาณ



รูปที่ 3-18 รายละเอียดเบื้องต้นของไอซีแปลงสัญญาณเพื่อเชื่อมต่อกับพอร์ตอนุกรมของคอมพิวเตอร์



รูปที่ 3-19 วงจรเชื่อมต่อ MAX232 หรือ ICL232 เข้าพอร์ตอนุกรมของคอมพิวเตอร์และไมโครคอนโทรลเลอร์

ไอซีที่ทำหน้าที่ในการแปลงระดับสัญญาณนี้ต้องทำการแปลงข้อมูลส่งของไมโครคอนโทรลเลอร์ MCS-51 จากระดับทีทีแอลไปเป็นระดับของ RS-232 และทำการแปลงข้อมูลรับจากคอมพิวเตอร์จากระดับของ RS-232 เป็นระดับทีทีแอลเพื่อให้สามารถถ่ายทอดไปยังไมโครคอนโทรลเลอร์ MCS-51 ได้อย่างสมบูรณ์ ไอซีดังกล่าวมีด้วยกันหลายเบอร์จากหลายผู้ผลิต อาทิ MAX232 จาก MAXIM หรือ ICL232 จาก HARRIS เป็นต้น ในรูปที่ 3-19 แสดงการจัดขาของไอซี ICL232 ซึ่งใช้ในการแปลงสัญญาณ RS-232 ส่วนวงจรของการต่อกับไมโครคอนโทรลเลอร์ MCS-51 แสดงในรูปที่ 3-19

3.3 การเขียนโปรแกรม Visual Basic เพื่อใช้งานพอร์ตอนุกรม

3.3.1 คอนโทรล MSComm

สำหรับการใช้งาน Visual BASIC ตั้งแต่เวอร์ชัน 2 เป็นต้นมา ใน Visual BASIC จะมีคัสตอมคอนโทรลสำหรับการสื่อสารอนุกรมผ่านทางพอร์ตอนุกรมของคอมพิวเตอร์มาให้ โดยใน Visual BASIC เวอร์ชัน 2 และเวอร์ชัน 3 จะใช้ชื่อว่า MSCOMM.VBX ส่วนเวอร์ชัน 4 ใช้ชื่อว่า MSCOM16OCX สำหรับการทำงานกับระบบปฏิบัติการ 16 บิต และ MSCOMM32.OCX สำหรับการทำงานกับระบบปฏิบัติการ 32 บิต สำหรับใน Visual BASIC เวอร์ชัน 5 จะมีเพียง MSCOMM32.OCX เท่านั้นเพราะถูกอกแบบมาให้ใช้งานกับระบบปฏิบัติการ 32 บิต

MSComm จัดเตรียมทางเลือกเอาไว้ 2 ทางเพื่อความสะดวกในการสื่อสารข้อมูล ทางแรกคือ การสื่อสารข้อมูลที่กระตุ้นด้วยเหตุการณ์ (event-driven communications) เป็นรูปแบบการใช้งานที่มีประสิทธิภาพมากสำหรับการตอบสนองแบบทันทีทันใด เช่น เมื่อตัวอักษรถูกส่งมาที่พอร์ตอนุกรมหรือเกิดการเปลี่ยนแปลงที่ขา Data Carrier Detect (DCD) หรือขา Request To Send (RTS) เหตุการณ์ Oncomm ของ MSComm จะสามารถตรวจจับสัญญาณนั้นได้ทันที ซึ่งจะกล่าวถึงรายละเอียดในหัวข้อคุณสมบัติ CommEvent ต่อไป ส่วนทางเลือกที่สองเป็นการคอยตรวจสอบค่าเหตุการณ์และความผิดพลาดที่เกิดขึ้นด้วยการดูค่าที่เปลี่ยนแปลงภายในคุณสมบัติ CommEvent หลังจากให้โปรแกรมทำงานในฟังก์ชั่นต่างๆ ไปเรียบร้อยแล้ว ซึ่งวิธีนี้ใช้งานได้ดีในกรณีที่โปรแกรมมีขนาดเล็ก

คอนโทรล MSComm 1 ตัวสามารถควบคุมการทำงานของพอร์ตอนุกรมได้ 1 พอร์ต ถ้าในโปรแกรมที่ใช้งานต้องการติดต่อกับพอร์ตอนุกกรมมากกว่า 1 พอร์ตจะต้องใช้คอนโทรล MSComm มากว่า 1 ตัวเพื่อควบคุมพอร์ตอนุกรมในแต่ละพอร์ต แอดเดรสของพอร์ตอนุกรมและแอดเดรสของการเกิดอินเตอร์รัปต์สามารถเปลี่ยนแปลงได้จากการแก้ไขค่าที่ Control Panel

ถึงแม้ว่า คอนโทรล MSComm จะมีคุณสมบัติ (property) มากมายแต่สามารถทำความเข้าใจได้ไม่ยากดังนี้

Commport

ใช้ในการกำหนดและอ่านค่าพอร์ตอนุกรมที่ติดต่ออยู่ (COM1,COM2,COM3,COM4)

รูปแบบการใช้งาน

object.CommPort [= value]

โดย Value เป็นค่าของพอร์ตอนุกรม ชนิดของข้อมูลเป็น Iteger ค่า Value สามารถกำหนดได้ในช่วง 1-16 (ค่าเริ่มต้นกำหนดไว้ที่ 1) เมื่อมีการกำหนดค่าแล้วทำการเหิดพอร์ตโดยใช้คุณสมบัติ PortOpen แต่ว่าพอร์ตนั้นไม่มีอยู่ในระบบ MSComm จะสร้างสัญญาณแสดงข้อผิดพลาด error 68 ขึ้นมาซึ่งหมายถึง อุปกรณ์ตัวนี้ไม่มีอยู่ในระบบ ดังนั้นการเขียนโปรแกรมจึงจำเป็นต้องกำหนดตำแหน่งของพอร์ตอนุกรมก่อนที่ใช้คำสั่ง OpenPort

Setting

ใช้ในการกำหนดและอ่านค่าอัตราบอด ,พาริตี้ ,จำนวนของบิตข้อมูล,จำนวนของบิตปิดท้าย

รูปแบบการใช้งาน

object.Setting [= value]

ค่า Value มีชนิดข้อมูลเป็นแบบ Sting มีรูปแบบเป็น “BBBB,P,D,S” โดย BBBB เป็นค่าอัตราบอด ,P เป็นค่าพาริตี้ , D เป็นจำนวนของบิตข้อมูล และ S เป็นจำนวนของบิตปิดท้าย ปกติแล้วค่านี้ถูกกำหนดไว้เป็น “9600,N,8,1”

ค่าบอดเรตมาตรฐานที่ใช้กับ MSComm มีดังนี้

100 บิตต่อนาที

300 บิตต่อนาที

600 บิตต่อนาที

1,200 บิตต่อนาที

2,400 บิตต่อนาที

9,600 บิตต่อนาที **(ค่าปกติ)**

14,400 บิตต่อนาที

19,200 บิตต่อนาที

28,800 บิตต่อนาที

38,400 บิตต่อนาที **(สงวน)**

56,000 บิตต่อนาที **(สงวน)**

128,00 บิตต่อนาที **(สงวน)**

256,000 บิตต่อนาที **(สงวน)**

สำหรับค่ามาตรฐานในการกำหนดค่าพาริตี้มีดังนี้

สัญลักษณ์	รายละเอียด
E	พาริตี้คู่ (Even)
M	ลอจิก “1” (MARK)
N	ไม่ใช้ (ค่าปกติ)
O	พาริตี้คี่ (Odd)
S	ลอจิก “0” (Space)

ค่าที่ใช้ในการกำหนดจำนวนบิตมี 5 ค่าคือ 4,5,6,7 และ 8 (เป็นค่าปกติ)

ค่าที่ระบุจำนวนบิตปิดท้ายมี 3 ค่าคือ 1 (เป็นค่าปกติ) 1,5, และ 2

ตัวอย่างการใช้งานคำสั่ง Setting โดยจะเป็นการกำหนดค่าบอดเรตเท่ากับ 9600 ไม่มีพาริตี้จำนวนบิตข้อมูล 8 บิตและบิตปิดท้าย 1 บิต สามารถเขียนโปรแกรมได้ดังนี้

MSComm1.Setting = “9600,N,8,1”

หมายเหตุ สาเหตุที่ค่าที่กำหนดจะต้องอยู่ภายในเครื่องหมายคำพูด “” เนื่องจาก ค่าที่กำหนดนี้อยู่ในรูปตัวแปร String

PortOpen

ใช้ในการกำหนดและอ่านค่าสถานะของพอร์ตอนุกรม เพื่อเปิดและปิดพอร์ตอนุกรม

รูปแบบการใช้งาน

object.PortOpen [= value]

ค่า Value มีชนิดข้อมูลเป็นแบบบูลีนคือ True กับ False โดย True หมายถึงการเปิดพอร์ตอนุกรมและ False หมายถึงการปิดพอร์ตอนุกรม สำหรับการปิดพอร์ตนั้นจะมีการเคลียร์บัฟเฟอร์รับข้อมูลและบัฟเฟอร์ส่งข้อมูลด้วย คอนโทรล MSComm จะปิดพอร์ตอนุกรมโดยอัตโนมัติเมื่อออกจากโปรแกรม ก่อนที่จะใช้คุณสมบัติ PortOpen ต้องตรวจสอบให้แน่ใจก่อนว่าคุณสมบัติ CommPort นั้นได้ทำการกำหนดตำแหน่งของพอร์ตอนุกรมไว้ถูกต้องหรือไม่ มิเช่นนั้น MSComm จะแสดงข้อผิดพลาด Error 68 แจ้งแก่ผู้ใช้งาน หรือถ้าพอร์ตอนุกรมนั้นถูกเปิดเอาไว้แล้ว โปรแกรมก็จะแจ้งข้อผิดพลาดออกมาเช่นเดียวกัน

ถ้าคุณสมบัติ DTREnable หรือ RTSEnable ถูกกำหนดให้เป็น True ก่อนที่จะทำการเปิดพอร์ต ค่าคุณสมบัติของ DTREnable หรือ RTSEnable จะถูกเซตเป็น False หลังจากปิดพอร์ตแต่ถ้าเซตเป็น False หลังจากปิดโปรแกรมแล้ว ค่าที่กำหนดไว้จะเป็นค่าเดิม

ตัวอย่างการใช้คำสั่งเปิดพอร์ต เพื่อติดต่อสื่อสารกับพอร์ตอนุกรม COM1 และมีบอดเรต 9,600 บิตต่อนาที ไม่มีพาริตี้ จำนวนบิตข้อมูล 8 บิต และบิตปิดท้าย 1 บิตมีดังนี้

```
MSComm.Setting = "9600,n,8,1"
MSComm.CommPort = 1
MSComm1.PortOpen = True
```

Input

อ่านค่าและลบค่าขบวนข้อมูลจากบัฟเฟอร์ภาครับ

รูปแบบการใช้งาน

object.Input

คุณสมบัติ InputLen เป็นตัวกำหนดจำนวนของตัวอักษรที่จะอ่านโดยคุณสมบัติ Input การกำหนดค่าให้ InputLen เท่ากับ 0 เป็นการกำหนดให้คุณสมสัติ Input ทำการอ่านค่าข้อมูลในบัฟเฟอร์รับข้อมูลทั้งหมด

คุณสมบัติ InputMode เป็นตัวกำหนดชนิดของข้อมูลที่คุณสมบัติ Input รับเข้ามา ถ้า InputMode ถูกกำหนดเป็น cominputModeText คุณสมบัติ Input จะส่งค่าข้อมูลกลับมาในรูปแบบของข้อความชนิดข้อมูลเป็นแบบ Variant ถ้า InputMode กำหนดเป็น cominputModeBibary คุณสมบัติ Input จะส่งข้อมูลกลับมาในรูปของไบนารี่และชนิดข้อมูลเป็นแบบ Variant

ตัวอย่างโปรแกรมแสดงให้เห็นถึงวิธีในการรับข้อมูลการบัฟเฟอร์รับข้อมูลทั้งหมด

```
private Sub Command1_Click()
Dim InString as String
    MSComm1.InputLen = 0            ' Retrieve all available data.
    If MSComm1.InBufferCount Then   ' Check for data.
    InString = MSComm1.Input        ' Read data.
    End If
End Sub
```

InBufferCount

ส่งค่าจำนวนของตัวอักษรที่อยู่ในบัฟเฟอร์ภาครับ

รูปแบบการใช้งานคำสั่ง

object.InBufferCount [= value]

คำสั่ง InBufferCount จะแสดงค่าจำนวนของตัวอักษร ซึ่งรับมาจากภายนอกและยังเก็บอยู่ในฟัฟเฟอร์ภาครับ เพื่อให้ผู้ใช้งานอ่านค่าออกไป สำหรับการเคลียร์ค่าบัฟเฟอร์ภาครับทำได้โดยกำหนดให้ InBufferCount มีค่าเป็น O

หมายเหตุ อย่าสับสนระหว่างคำสั่ง InBufferSize และ InBuffeCount คำสั่ง InBufferSize นั้นใช้เพื่อกำหนดขนาดของบัฟเฟอร์ภาครับ

InBufferSize

กำหนดและคืนค่าขนาดของบัฟเฟอร์ภาครับในหน่วยเป็นไบต์

รูปแบบการใช้งานคำสั่ง

object.InBufferSize [= value]

คำสั่ง InBufferSize ใช้เพื่อกำหนดขนาดของบัฟเฟอร์ภาครับ โดยค่าเริ่มต้นถูกกำหนดไว้ที่ 1,024 ไบต์

หมายเหตุ การกำหนดค่าบัฟเฟอร์ภาครับขนาดใหญ่จะทำให้หน่อยความจำที่เหลือสำหรับเการใช้งานส่วนอื่นๆจะเหลือน้อย อย่างไรก็ตามการกำหนดค่า ฟับฟเฟอร์ภาครับที่น้อยเกินไปจะทำให้เกิดการโอเวอร์โฟลวหรือข้อมูลล้นบัฟเฟอร์ เว้นแต่จะมีการใช้แฮนด์เช็ก ดังนั้นค่าปานกลางที่เหมาะสมก็คือค่า 1,024 ซึ่งเป็นค่าเริ่มต้นนั่นเอง แต่ถ้าโปรแกรมมีการเกิดโอเวอร์โฟลวแบ้วจึงค่อยปรับเพิ่มขนาดของฟัฟเฟอร์ให้มีค่ามากขึ้น

InputLen

กำหนดค่าและคืนค่าจำนวนของตัวอักษรที่อ่านจากบัฟเฟอร์ภาครับ

รูปแบบการใช้งานคำสั่ง

object.InputLen [= value]

ค่าเริ่มต้นของคุณสมบัติ InputLen มีค่าเท่ากับ “0” การกำหนดค่าเท่ากับ “0” จะทำให้ คำสั่ง Input ของ MSComm อ่านค่าข้อมูลที่อยู่ภายในบัฟเฟอร์ภาครับทั้งหมด

ถ้าไม่มีข้อมูลอยู่ในบัฟเฟอร์ภาครับมากเท่ากับจำนวน InputLen คำสั่ง Input จะส่งค่าว่าง (“”) กลับออกมา ผู้ใช้งานสามารถตรวจสอบข้อมูลในบัฟเฟอร์ภาครับได้โดยใช้คุณสมบัติ

InBufferCount โดยกำหนดให้มีข้อมูลอยู่บัฟเฟอร์ภาครับก่อนแล้วจึงค่อยอ่านข้อมูลจากบัฟเฟอร์ภาครับ

คุณสมบัตินี้มักใช้กับการอ่านค่าข้อมูลจากเครื่องมือหรือเครื่องจักรที่มีการกำหนดค่าขนาดความยาวของข้อมูลเอาไว้แล้ว

ตัวอย่างโปรแกรมการอ่านค่าตัวอักษรออกมา 10 ตัวอักษร

```
Private Command1_Click()
Dim ComData as String
MSComm 1.InputLen = 10              'Specify a 10 character block of data.
CommData = MSComm1.Input            'Read data
End Sub
```

InputMode

กำหนดค่าและคืนค่าชนิดของข้อมูลที่รับโดยคำสั่ง Input

รูปแบบการใช้งานคำสั่ง

object.InputMode [= value]

คุณสมบัติ InputMode ใช้กำหนดว่าข้อมูลชนิดไหนที่รับเข้ามาผ่านคำสั่ง Input โดยข้อมูลจะเลือกได้ 2 ประเภทคือ

comInputModeText สำหรับข้อมูลที่อยู่ในรูปข้อความตัวอักษรตามมาตรฐาน ANSI โดยจะต้องกำหนดค่าเป็น "0" และค่าเริ่มต้นของการรับค่าข้อมูลก็จะเป็นค่านี้

comInputModeBinary สำหรับข้อมูลอื่นๆซึ่งจะเก็บในรูปไบนารี่รวมกันอยู่เป็นไบต์ข้อมูล

ตัวอย่างการใช้งาน InputMode ต่อไปนี้จะทำการอ่านค่าข้อมูล 10 ไบต์จากพอร์ตอนุกรมและเก็บข้อมูลไว้ในตัวแปรแบบอาเรย์ ชนิดข้อมูลเป็นแบบไบต์

```
Private Sub Command1_click()
Dim Buffer as Variant
Dim Arr() as Byte
Mscomm1.ComPort = 1
MSComm1.PortOpen = True
MSComm1.InputMode = comInputModeBinary
Do Until MSComm1.InBufferCount < 10
```

```
Input buffer
        DoEvents
Loop
Buffer = MSComm1.Input
Arr = Buffer
End Sub
```

Output

ใช้ในการส่งขบวนของข้อมูลไปยังบัฟเฟอร์ส่งข้อมูล

รูปแบบการใช้งาน

Object.Output (= value)

ค่า value เป็นค่าของตัวอักษรที่เขียนไปยังบัฟเฟอร์ส่งข้อมูล คุณสมบัติ Output สามารถใช้ในการส่งข้อมูลตัวอักษรหรือข้อมูลไบนารี่ก็ได้ โดยการส่งข้อมูลเป็นรูปแบบตัวอักษรจะต้องกำหนดข้อมูลเป็นแบบ Variant และมีข้อมูลภายในเป็นแบบ String สำหรับการส่งข้อมูลไบนารี่จะต้องกำหนดชนิดของข้อมูลเป็นแบบ Variant และมีข้อมูลภายในเป็นแบบ Byte

ตัวอย่างโปรแกรมการส่งค่าที่ป้อนจากคีย์บอร์ดไปยังพอร์ตอนุกรม โดยใช้คุณสมบัติ Output

```
Private Sub Form_KeyPress ( KeyAscii As Integer )
        Dim Buffer as Variant
        MSComm1.CommPort = 1
        MSComm1.PortOpen = True
        Buffer = Chr$ ( KeyAscii )
        MSComm1.Output = Buffer
End Sub
```

OutBufferCount

คืนค่าจำนวนของข้อมูลตัวอักษรที่เก็บอยู่ในบัฟเฟอร์ภาคส่ง และสามารถใช้คำสั่งนี้เพื่อเคลียร์บัฟเฟอร์ภาคส่งได้ด้วย

รูปแบบการใช้งานคำสั่ง

Object.OutBufferCount (= value)

ผู้ใช้งานสามารถเคลียร์บัฟเฟอร์ภาคส่งได้โดยการกำหนดค่า OutBufferCount เท่ากับ "0"

หมายเหตุ ระวังการสับสนระหว่างคุณสมบัติ OutBufferCount กับ OutBufferSize ซึ่ง OutBufferSize ใช้เพื่อกำหนดขนาดของบัฟเฟอร์ภาคส่ง

OutBufferSize

กำหนดค่าและคืนค่าขนาดของบัฟเฟอร์ภาคส่ง ชนิดตัวแปรเป็นแบบไบต์

รูปแบบการใช้งานคำสั่ง

Object.OutBufferSize (= object)

คุณสมบัติ OutBufferSize ใช้สำหรับกำหนดขนาดของบัฟเฟอร์ภาคส่ง โดยค่าปกติที่ใช้งานจะมีค่าเท่ากับ 512 ไบต์

หมายเหตุ การกำหนดค่าบัฟเฟอร์ภาคส่งที่มากเกินไปจะทำให้ มีหน่วยความจำเหลือให้ใช้งานน้อย แต่อย่างไรก็ตามถ้ากำหนดค่าน้อยเกินไป จะทำให้เกิดข้อมูลบัฟเฟอร์ขึ้นได้ ยกเว้นจะมีการใช้ แฮนเช็ค วิธีการที่ถูกต้องในการกำหนดค่าคือ ทดลองใช้ค่าเริ่มต้นคือค่า 512 ไบต์ดูก่อนถ้าโปรแกรมทำงานแล้วเกิดการล้นของข้อมูลค่อยเพิ่มค่าของ OutBufferSize ให้มากขึ้น

ParityReplace

กำหนดแบะคืนค่าตัวอักษรที่ไปวางแทนในตำแหน่งที่เกิดข้อผิดพลาดจากพาริตี้

รูปแบบการใช้งานคำสั่ง

Object.ParityReplace (= value)

บิตพาริตี้ เป็นบิตที่ทางภาคส่งข้อมูลทำการส่งมาพร้อมกับข้อมูล เพื่อตรวจสอบข้อผิดพลาดของข้อมูล โดยเมื่อมีการใช้บิตพาริตี้ คอนโทรล MSCOM จะทำการบวกบิตทุกบิตที่มีค่าลอจิก "1" ในแต่ละไบต์ และทำการตรวจสอบผลลัพธ์ว่าบิตที่อ่านได้นั้นมีจำนวนบิตทุกลิตที่มีค่าลอจิก "1" เป็นเลขคู่หรือคี่ และตรงกับค่าที่กำหนดไว้แต่ต้นหรือไม่ ถ้าค่าที่นำมาบวกแล้วมีพาริตี้ไม่ตรงแสดงว่าการรับส่งข้อมูลผิดพลาด

การกำหนดค่าเริ่มต้นให้กับ ParityReplace นั้นกำหนดให้ใช้เครื่องหมาย (?) ไปวางไว้ที่ตำแหน่งที่เกิดพาริตี้ผิดพลาด ถ้ากำหนดค่า ParityReplace ให้เป็นค่าว่าง ("") จะเป็นการยกเลิกการใช้งาน ParityReplace และไม่มีการป้อนข้อมูลแทนเมื่อตรวจพบข้อผิดพลาด

ParityReplace ใช้ชนิดข้อมูลเป็นแบบสตริง แต่จะการกำหนด จะกำหนดได้เพียงไบต์เดียวเท่านั้น ซึ่งจะสามารถใช้ค่าใดๆก็ได้ที่เป็นโค้ด ANSI มีค่าอยู่ระหว่าง 0-255

DTREnable

ใช้ในการกำหนดสถานะลอจิกของขา Data Terminal Ready (DTR) โดยสัญญาณของขา DTR จะส่งจากคอมพิวเตอร์ไปยังโมเด็มเพื่อแสดงว่าคอมพิวเตอร์พร้อมที่จะรับข้อมูลแล้ว ชนิดของข้อมูลเป็นแบบบูลีน

รูปแบบการใช้งาน

Object.DTREnable (= value)

ค่า Value เป็นค่าสถานะ True หรือ False เพื่อกำหนดลอจิกของขา DTR ให้เป็น “0” หรือ “1” โดย

True หมายถึงให้ขา DTR มีลอจิก “1”

False หมายถึงให้ขา DTR มีลอจิก “0” (เป็นค่าปกติ)

หมายเหตุ เมื่อขา DTR ถูกกำหนดสถานะให้เป็น True ที่ขา DTR จะมีสถานะลอจิก “1” เมื่อทำการเปิดพอร์ตและจะมีสถานะเป็น “0” เมื่อมีการปิดพอร์ต เมื่อขา DTR ถูกกำหนดสถานะเป็น False ที่ขา DTR จะมีสถานะลอจิก “0” ตลอดเวลาไม่ว่าจะใช้คำสั่งเปิดพอร์ตหรือปิดพอร์ต

สำหรับการใช้งานกับโมเด็ม การทำให้ขา DTR เป็นลอจิก “0” จะเป็นการวางหูโทรศัพท์หรือยกเลิกการติดต่อ

RTSEnable

ใช้เพื่อกำหนดสถานะลอจิกให้ขา Request To Send (RTS) โดยขา RTS จะเป็นสัญญาณที่ส่งจากคอมพิวเตอร์ไปยังโมเด็มเพื่อร้องขอส่งข้อมูล ชนิดของข้อมูลเป็นแบบ Boolean

รูปแบบการใช้งาน

Object.RTSEnable (= value)

ค่า Value เป็นค่าสถานะ True หรือ False เพื่อกำหนดลอจิก “0” หรือ “1” ให้ขา RTS โดย

True หมายถึง ให้ขา RTS มีลอจิก “1”

False หมายถึง ให้ขา RTS มีลอจิก “0” (เป็นค่าปกติ)

หมายเหตุ เมื่อขา RTSEnable ถูกกำหนดให้เป็น True ขา RTS จะมีสถานะลอจิก "1" เมื่อเปิดพอร์ตและมีสถานะลอจิก "0" เมื่อปิดพอร์ต

EOFEnable

เป็นการกำหนดให้ MSComm รอสัญลักษณ์แสดงส่วนท้ายสุดของไฟล์ (End of file : EOF) ระหว่างการรับอินพุตเข้ามา ถ้าพบสัญลักษณ์ EOF ภาคอินพุตจะหยุดรับข้อมูล และเหตุการณ์ OnComm ถูกการตุ้นให้ทำงาน คุณสมบัติ CommEvent จะมีค่าเท่ากับ 7 หรือ ComEvEOF

รูปแบบการใช้งาน

Object.EOFEnable (= value)

โดย value เป็นค่าสถานะ True หรือ False เพื่ออีนาเบิลหรือดิสเอเบิลการทำงานของเหตุการณ์ OnComm เมื่อตรวจพบสัญลักษณ์ EOF โดย

True หมายถึง เหตุการณ์ OnComm จะถูกกระตุ้นให้ทำงานด้วย EOF

False หมายถึง เหตุการณ์ OnComm จะไม่ถูกกระตุ้นให้ทำงานด้วย EOF (เป็นค่าปกติ)

เมื่อ EOFEnable กำหนดให้เป็น False ส่วนควบคุมจะไม่มีการตรวจสอบสัญลักษณ์ EOF

CTSHolding

ผู้ใช้งานสามารถตรวจสอบการทำงานของขา Clear To Send (CTS) ได้ว่ามีสถานะลอจิก "0" หรือ "1" โดยค่าที่อ่านได้จะเป็นบูลีน True และ False ถ้าค่า CTSHolding เป็น True ขา CTS จะมีสถานะลอจิกเป็น "1" ถ้าค่า CTSHolding เป็น False ขา CTS จะมีสถานะลอจิกเป็น "0"

รูปแบบการใช้งาน

Object.CTSHolding

เมื่อขา CTS เป็นลอจิก "0" (CTSHolding = False) และเกิดไทม์เอาต์คอนโทรล MSComm จะกำหนดให้คุณสมบัติ CommEvent มีค่าเป็น comEventCTSTO (Clear To Send Time) และกระตุ้นให้เกิดเหตุการณ์ OnComm

CDHolding

ผู้ใช้งานสามารถตรวจสอบการทำงานของขา Data Carrier Detect (DCD) ได้ว่ามีสถานะลอจิกเป็น “1” หรือ “0” โดยค่าที่อ่านได้จะเป็นบูลีน True และ False ถ้าค่า CDHolding เป็น True ขา DCD จะมีสถานะลอจิก “1” ถ้าค่า CDHolding เป็น False ขา DCD จะมีสถานะลอจิก “0”

รูปแบบการใช้งาน

Object.CDHolding

เมื่อขา DCD มีลอจิก “1” (CDHolding = True) และเกิดไทม์เอาต์คอนโทรล MSComm จะกำหนดให้คุณสมบัติ CommEvent มีค่าเป็น comEventCDTO (Carrier Detect Timeout Eror) และกระตุ้นให้เกิดเหตุการณ์ OnComm

DSRHolding

ผู้ใช้งานสามารถตรวจสอบการทำงานของขา DSR (DSR) ได้ว่ามีสถานะลอจิก “1” หรือ “0” โดยค่าที่อ่านได้จะเป็นบูลีน True และ False ถ้าค่า DSRHolding เป็น True ขา DSR จะมีสถานะลอจิก “1” ถ้าค่า DSRHolding เป็น False ขา DSR จะมีสถานะลอจิก “0”

รูปแบบการใช้งาน

Object.DSRHolding

เมื่อขา DSR เป็นลอจิก “1” (DSRHolding = True) และเกิดไทม์เอาต์คอนโทรล MSComm จะกำหนดให้คุณสมบัติ CommEvent มีค่าเป็น comEventDSRTO (Data Set Ready Timeout) และกระตุ้นให้เกิดเหตุการณ์ Oncomm

Handshaking

กำหนดคุณสมบัติและคืนค่ารูปแบบแฮนด์เช็กทางฮาร์ดแวร์

รูปแบบการใช้งานคำสั่ง

Object.Handshaking (= value)

ค่าตัวแปร Value ที่กำหนดค่ากำหนดได้ 4 รูปแบบด้วยกันคือ

1. comNone ค่าที่กำหนดคือ 0 เป็นการกำหนดให้ไม่มีการแฮนด์เช็ก (เป็นค่าเริ่มต้น)
2. comXOnXOff ค่าที่กำหนดคือ 0 เป็นการกำหนดให้ใช้แฮนด์เช็กแบบ XON/XOFF
3. comRTS ค่าที่กำหนดคือ 2 เป็นการกำหนดใช้ขา RTS/CTS (Request To Send/Clear To Send)

4. comRTSXOnXOff ค่าที่กำหนดคือ 3 เป็นการกำหนดให้ใช้ทั้งแบบ Request To Send และ XON/XOFF

คุณสมบัติ Handshaking ใช้เพื่อกำหนดรูปแบบการสื่อสารภายใน ระหว่างที่ข้อมูลถูกส่งไปยังบัฟเฟอร์ภาครับ เมื่อข้อมูลตัวอักษรถูกมาถึงพอร์ตอนุกรม อุปกรณ์สื่อสารข้อมูลจะทำการย้ายข้อมูลไปยังบัฟเฟอร์ภาครับ เพื่อที่จะให้โปรแกรมสามารถอ่านค่าไปใช้งานได้ ถ้าไม่มีบัฟเฟอร์ภาครับโปรแกรมที่ใช้งานจะต้องทำการอ่านค่าข้อมูลโดยตรงจากฮาร์ดแวร์ของพอร์ตอนุกรม ซึ่งผู้ใช้งานจะเกิดปัญหาข้อมูลสูญหายได้ เนื่องจากว่าการเปลี่ยนแปลงของข้อมูลที่ส่งเข้ามามีการเปลี่ยนแปลงอย่างรวดเร็ว

คุณสมบัติ handshaking ช่วยให้ผู้ใช้งานแน่ใจได้ว่าข้อมูลที่ได้รับมานั้นไม่มีการสูญหาย เมื่อบัฟเฟอร์ภาครับที่รับข้อมูลนั้นเกิดข้อมูลล้นหรือโอเวอร์โฟลว (overflow) โดยใช้วิธีการตรวจสอบความพร้อมของบัฟเฟอร์ของบัฟเฟอร์ว่าพร้อมรับข้อมูลหรือไม่ก่อนที่จะส่งข้อมูลมาให้

Break

ใช้ในการเซตและเคลียร์ค่าสัญญาณ Break ชนิดของข้อมูลเป็นแบบ Boolean

รูปแบบการใช้งาน

Object.Break (= value)

โดย Value เป็นค่าบูลีน ถ้า Value = True หมายถึง การส่งสัญญาณ Break ออกไป ถ้า Value = False หมายถึงการเคลียร์สัญญาณ Break

เมื่อกำหนดให้สัญญาณ Break เป็น True จะเป็นการหยุดการส่งข้อมูลชั่วคราวจนกว่าจะมีการสั่งให้สัญญาณ Break เป็น False

ตัวอย่าง เป็นวิธีการส่งสัญญาณ Break ออกไปเป็นช่วงเวลาสั้นๆที่ 1/10 ของวินาที

```
MSComm1.Break = True
Duration! = Timer + .1
Do Until Timer > Duration!
Dummy = DoEvents ()
Loop
MSComm1.Break = False
```

เหตุการณ์ OnComm

เหตุการณ์ OnComm จะถูกสร้างขึ้นเมื่อค่าของคุณสมบัติ CommEvent มีการเปลี่ยนแปลงเพื่อแสดงผลการเปลี่ยนแปลงเหล่านั้นแบบทันทีทันใดหรือแสดงข้อผิดพลาดที่เกิดขึ้น ตัวอย่างโปรแกรมย่อย OnComm เพื่อนำเหตุการณ์ CommEvent มาแสดง

```
Private Sub MSComm_OnComm ()
    Select Case MSComm1.CommEvent
    ' Handle each event or error by placing
    ' code below each case statement
    ' Errors
        Case comEventBreak      ' A Break Was received.
        Case comEventCDTO       ' CD ( RLSD ) Timeout.
        Case comEventCTSTO      ' CTS Timeout.
        Case comEventDSRTO      ' DSR Timeout.
        Case comEventFrame      ' Frameing Error
        Case comEventOverrun    ' Data Lost.
        Case comEventRxOver     ' Receive buffer overflow.
        Case comEventRxParity   ' Parity Error.
        Case comEventTxFull     ' Transmit buffer full.
    ' Events
        Case comEvCD            ' Change in the CD line.
        Case comEvCTS           ' Change in the CTS line.
        Case comEvDSR           ' Change in the DSR line.
        Case comEvRing          ' Change in the Ring Indecator.
        Case comEvReceive       ' Received Rthreshold # of chars.
        Case comEvSend          ' Sthreshold number in the ' transmit
buffer.
        Case comEvEof           ' An EOF charater was found in the input
stream
    End Select
```

End Sub

3.3.2 ค่าคงที่คุณสมบัติของคอนโทรล MSComm

ค่าคงที่สำหรับคุณสมบัติ Handshake

ค่าคงที่	ค่า	รายละเอียด
comNone	0	ส่งค่าเหตุการณ์ (send event)
comXonXoff	1	ไม่ใช้การตรวจสอบแฮนด์เชกแบบ Xon/Xoff
comRTS	2	ใช้การตรวจสอบแฮนด์เชกผ่านทางขา RTS และ CTS
comRTS	3	กำหนดการตรวจสอบแฮนด์เชกทั้งแบบ RTS,CTS และ Xon/Xoff

ค่าคงที่สำหรับคุณสมบัติ OnComm

ค่าคงที่	ค่า	รายละเอียด
ComEvSend	1	ส่งค่าเหตุการณ์ (send event)
ComEvReceive	2	รับค่าเหตุการณ์ (receive event)
ComEvCTS	3	มีการเปลี่ยนแปลงที่ขา CTS
ComEvDSR	4	มีการเปลี่ยนแปลงที่ขา DSR
ComEvCD	5	มีการเปลี่ยนแปลงที่ขา DCD
ComEvRing	6	ตรวจจับสัญญาณกระดิ่งของโทรศัพท์
ComEvEOF	7	ตรวจพบตำแหน่งท้ายสุดของไฟล์ (End of file)

ค่าคงที่สำหรับคุณสมบัติ Error

ค่าคงที่	ค่า	รายละเอียด
ComEventBreak	1001	ได้รับสัญญาณ Break
ComEventCTSTO	1002	ขา CTS เกิดไทม์เอาต์
ComEventDSRTO	1003	ขา DSR เกิดไทม์เอาต์
ComEventFrame	1004	เกิดข้อผิดพลาดที่เฟรมข้อมูล (Framing error)
ComEventOverrun	1006	พอร์ตอนุกรมเกิดโอเวอร์รัน (Port overrun)
ComEventCDTO	1007	ขา DCD เกิดไทม์เอาต์
ComEventRxOver	1008	บัฟเฟอร์รับข้อมูลเกิดโอเวอร์โฟลว

ComEventRxParity	1009	เกิดข้อผิดพลาดที่พาริตี้ (Parity error)
ComEventTxFull	1010	บัฟเฟอร์ส่งข้อมูลเต็ม

ค่าคงที่สำหรับคุณสมบัติ InputMode

ค่าคงที่	ค่า	รายละเอียด
ComInputModeText	0	ข้อมูลที่รับมีคุณสมบัติเป็นข้อความ (ค่าปกติ)
ComInputModeBinary	1	ข้อมูลที่รับเข้ามาเป็นข้อมูลไบนารี่

3.3.3 การใช้ MSComm เพื่อการติดต่อฮาร์ดแวร์

จากรายละเอียดของ MSComm ที่กล่าวไปในตอนต้นนั้น จะเห็นได้ว่าวิธีการที่จะอ่านค่าหรือเขียนค่าไปยังขาสถานะและขาควบคุมของพอร์ตอนุกรมสามารถทำได้ง่ายดายมาก โดยใช้คำสั่งเหล่านี้

DTREnable	สำหรับสั่งให้ขา DTR มีลอจิก “0” หรือ “1”
RTSEnable	สำหรับสั่งให้ขา RTS มีลอจิก “0” หรือ “1”
CTSHolding	สำหรับอ่านค่าสถานะจากขา CTS ว่ามีลอจิก “0” หรือ “1”
CDHolding	สำหรับอ่านค่าสถานะจากขา DCD ว่ามีลอจิก “0” หรือ “1”
DSRHolding	สำหรับอ่านค่าสถานะจากขา DSR ว่ามีลอจิก “0” หรือ “1”
Break	สำหรับการสั่งให้ขา Txd มีลอจิก “0” หรือ “1”

การเขียนซอฟแวร์เพื่อควบคุมขาเอาต์พุต

การติดต่อกับพอร์ตอนุกรมบนระบบปฏิบัติการวินโดวส์ จะต้องเพิ่มอุปกรณ์ทางซอฟแวร์สำหรับการติดต่อกับพอร์ตอนุกรม นั่นคือ MSComm ซึ่งกล่าวรายละเอียดไว้แล้วในข้างต้น

ผู้ใช้งานสามารถเขียนโปรแกรมด้วย Visual Basic เพื่อส่งค่าออกไปยังขาเอาต์พุตต่างๆ ของพอร์ตอนุกรมได้ โดยใช้คำสั่งดังนี้

MSComm1.DTREnable = True	สำหรับการกำหนดให้ขา DTR มีลอจิก “1”
MSComm1.DTREnable = False	สำหรับการกำหนดให้ขา DTR มีลอจิก “0”
MSComm1.RTSEnable = True	สำหรับการกำหนดให้ขา RTS มีลอจิก “1”
MSComm1.RTSEnable = False	สำหรับการกำหนดให้ขา RTS มีลอจิก “0”
MSComm1.Break = True	สำหรับการกำหนดให้ขา TxD มีลอจิก “1”
MSComm1.Break = False	สำหรับการกำหนดให้ขา TxD มีลอจิก “0”

หมายเหตุ ก่อนที่จะใช้งานคำสั่งของคอนโทรล MSComm จะต้องทำการเปิดพอร์ตก่อน โดยเขียนโปรแกรมดังนี้

```
Private Sub Form_Load ()
MSComm1.PortOpen =True
End Sub
```

พร้อมกันนั้นจะต้องตรวจสอบพอร์ตที่ใช้งานให้ดีว่าพอร์ตอนุกรมที่ใช้นั้นถูกต้องหรือไม่ มิฉะนั้นโปรแกรมจะแสดงข้อผิดพลาดขึ้นมา

การอ่านค่าลอจิกจากพอร์ตอนุกรม RS-232

พอร์ตอนุกรมมีขาที่ทำหน้าที่อินพุตได้แก่ DSR, CTS, RI และ DCD โดยรีจิสเตอร์ที่ทำหน้าที่ควบคุมขาเหล่านี้คือรีจิสเตอร์แสดงสถานะโมเด็ม (MSR k) มีแอดเดรสถัดจากรีจิสเตอร์หลักของพอร์ตอนุกรม 6 ตำแหน่ง สำหรับบิตต่างๆ บนรีจิสเตอร์มีรายละเอียดดังนี้

บิต7	บิต6	บิต5	บิต4	บิต3	บิต2	บิต1	บิต0
DCD	RI	DSR	CTS	DDCD	DRI	DDSR	DCTS

4 บิตบนของรีจิสเตอร์จะแสดงสถานะการเปลี่ยนแปลงที่ขาอินพุตทั้ง 4 ขาของพอร์ตอนุกรมโดยตรง ส่วน 4 บิตล่าง จะมีสถานะเป็น “1” ก็ต่อเมื่อ 4 บิตบนมีการเปลี่ยนแปลงสถานะเมื่อเทียบกับการอ่านค่าครั้งก่อนหน้า

สำหรับการอ่านค่าจากขา DCD, CTS และ DSR โดยโปรแกรม Visual BASIC จะใช้ MSComm ร่วมกับคำสั่ง CDHolding, CTSHolding และ DSRHolding ตามลำดับ ซึ่งค่าที่อ่านได้นั้นจะเป็นบูลีน มีค่าเป็น True หรือ False ผู้ใช้งานสามารถตรวจสอบผลจากขาอิพุตเหล่านี้ได้โดยใช้คำสั่ง IF THEN

เนื่องจากสัญญาณของ RS-232 ต้องมีระดับแรงดัน ±3 ถึง ±12V แต่สัญญาณอินพุตที่เกิดขึ้นเป็นระดับทีทีแอล ดังนั้นเมื่อเกิดสัญญาณอินพุตขึ้น จะต้องส่งผ่านวงจรขับเพื่อปรับระดับแรงดันให้เหมาะสมเสียก่อน

บทที่ 4
การทดลองและผลการทดลอง

Project ส่วนแรก

ในส่วนโปรเจ็ก 1 นี้ได้ทำในส่วนของ Hardware เป็นส่วนใหญ่ ได้ออกแบบตัวหุ่นและฐานที่ตั้งของหุ่น ตัวหุ่นนั้นได้ไปหาโมเดลหุ่นที่ลงตัวกับงานโปรเจ็ก หาอยู่หลายแห่งหลายห้างอยู่จึงจะได้เจ้าหุ่นต้นแบบได้ ในส่วนเรื่องฐานได้คำนวนระยะการกระทบกันของตัวหุ่นและทิศทางในการโจมตีต่างๆและเขียนแบบฐานหุ่นออกมา ต่อไปก็หาวัสดุที่จะนำมาทำ เช่น พวกมอเตอร์ในการขับเคลื่อนต่างๆ ลูกปืน สายพาน และทำฐานตอนแรกคิดว่าจะใช้อลูมิเนียมการทำมันค่อนข้างลำบากและราคาแพงจึงหันมาซื้อไม้มาทำแทนซึ่งผมก็นั่งทำกับเพื่อนทั้งเลื่อย ทั้งขัดผิวไม้ กว่าจะได้อย่างที่เห็นในชิ้นงานน่ะครับ ส่วนเรื่องวงจรก็มี วงจรแปลงไฟที่จ่ายให้มอเตอร์และวงจร และทำวงจร MCS-51 ต่อกับบัฟเฟอร์และแม็ก 232 เพื่อที่จะได้ใช้ในคอมพิวเตอร์ควบคุมโดยใช้โปรแกรม Visual Basic ส่วนของวงจรควบคุมมอเตอร์ที่จะควบคุมให้มอเตอร์หมุนได้ 2 ทิศทางทั้งซ้าย และขวา ครั้งแรกได้ทดลองวงจรโมโนสเตเบิล เพื่อควบคุมทิศทางของมอเตอร์ซ้ายขวาแต่เนื่องจากว่าการทำงานของมันไม่เสถียรหรือไม่คงที่เท่าไหร่ จึงหันมาลองใช้อีกวงจรที่ใช้ในการควบคุมมอเตอร์โดยมี ไอซี L293 เป็นตัวทำงาน แต่ก็มีปัญหาเรื่องแรงไฟที่ใช้ในการขับมอเตอร์จึงเจออีกวงจร เป็นวงจรที่ใช้ ULN 2004 ร่วมกับ ชุดรีเลย์ ซึ่งใช้งานได้กับโปรเจ็กได้ลงตัว

PROJECT ส่วนที่สอง

ในโปรเจ็ก 2 นี้ก็ได้ทำการฮาร์ดแวร์เพิ่มเติมอีกชุด เพราะโรบอตไฟเตอร์วงจรและตัวหุ่นใช้ทั้ง 2 ชุด มีปัญหาหลายอย่าง เช่นมอเตอร์ DC ต้องใช้เฟืองทดให้กับหุ่น เพราะแขนที่ใช้ในการโจมตี มอเตอร์แรงบิดไม่พอเลยเจาะหาลองไปหาซื้อมอเตอร์ที่มีทดเฟืองในตัวเองมาใช้ แต่ขนาดต้องเล็กพอที่จะเข้าไปในหุ่นโรบอตได้ และปัญหาอื่นๆ ในส่วนของ Solfware ให้ควบคุม โดยศึกษาการเขียนโปรแกรม MCS-51 ซึ่งต้องศึกษาอย่างโดยละเอียดเพราะเป็นโปรแกรมสำคัญที่ใช้ในการควบคุมให้ทำงานตามคำสั่งต่างๆที่ใช้ในการควบคุมหุ่น ปัญหาเรื่องโปรแกรมก็มีเพราะเรื่องการควบคุมโดยใช้พอร์ตอนุกรมของ MCS-51 หนังสือกล่าวถึงจุดนี้น้อยมากลายละเอียดก็น้อย ตัวอย่างโปรแกรมก็น้อย เลยต้องหาหนังสือเกี่ยวกับพอร์ตอนุกรมมาหลายๆเล่มเพื่อใช้ในการอ่านและเขียนโปรแกรม ต้องใช้เวลานานเพื่อศึกษาคำสั่งต่างๆและทดลองให้ได้การทำงานควบคุมมอเตอร์ในการโจมตีและบราเรียป้องกัน และคำสั่งรับค่าความเสียหายจากการถูกโจมตีให้แสดงค่า ENERGY

พลังงานที่สูญเสียของหุ่นโรบอตได้ดังตามต้องการ ปัญหาก็มีมากมายในการเขียนโปรแกรม เช่น การเขียนอินเตอร์รัป MCS-51 ในการรับส่งข้อมูลเข้าออกของหุ่นโรบอต คือเมื่อโรบอตถูกอินเตอร์รัปจาก VB VB จะสั่งงาน RS-232 ผ่าน Serial port ของ Computer แล้วส่งข้อมูลผ่านไปยัง MCS-51 ตัว MCS-51 มันจะทำงานคำสั่งอินเตอร์รัปของ VB ที่เข้ามาก่อน เป็นอย่างนี้จะโกงกันได้โดยก็คำสั่งโจมตีอย่างเดียวมันก็จะทำตามคำสั่งนั้น พลังงานก็ไม่สามารถลดลงได้ เลยต้องทดลองและใช้เทคนิคในการเขียนโปรแกรมจนให้ได้การทำงานให้เป็นไปให้ได้

บรรณานุกรม

1. วรพจณ์ กรแก้ววัฒนกุล, ชัยวัฒน์ ลิ้มพรจิตรวิไล, เรียนรู้และปฏิบัติการไมโครคอนโทรลเลอร์ MCS-51, © Innovative Experiment Co.,Ltd
2. กฤษดา ใจเย็น, อรรถพล บุญยะโภคา, ชัยวัฒน์ ลิ้มพรจิตรวิไล, เรียนรู้และปฏิบัติการเชื่อมต่อคอมพิวเตอร์กับอุปกรณ์ภายนอกผ่านพอร์ตอนุกรม, © Innovative Experiment Co.,Ltd

เว็บไซต์

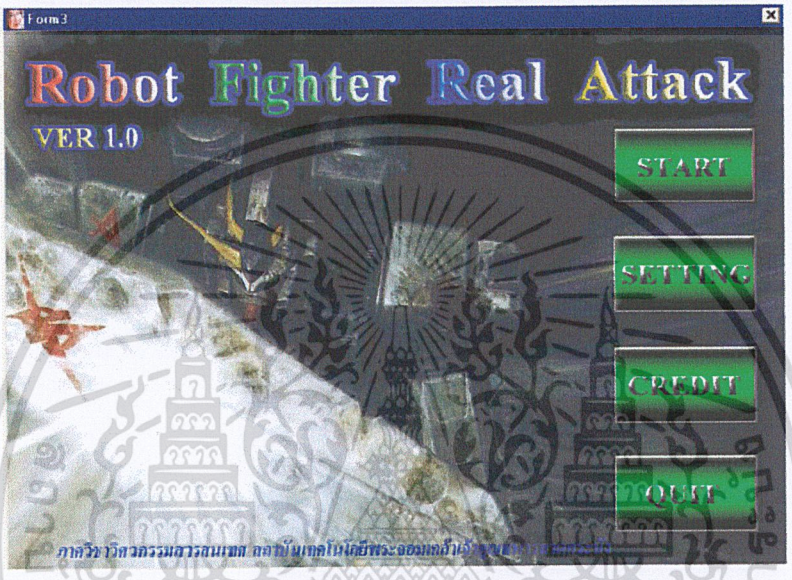
1. www.thaiio.com
2. www.se-ed.com
3. www.pantip.com

ภาคผนวก ก.

วิธีการเล่นเครื่องเล่น Robot Fighter สำหรับคอมพิวเตอร์

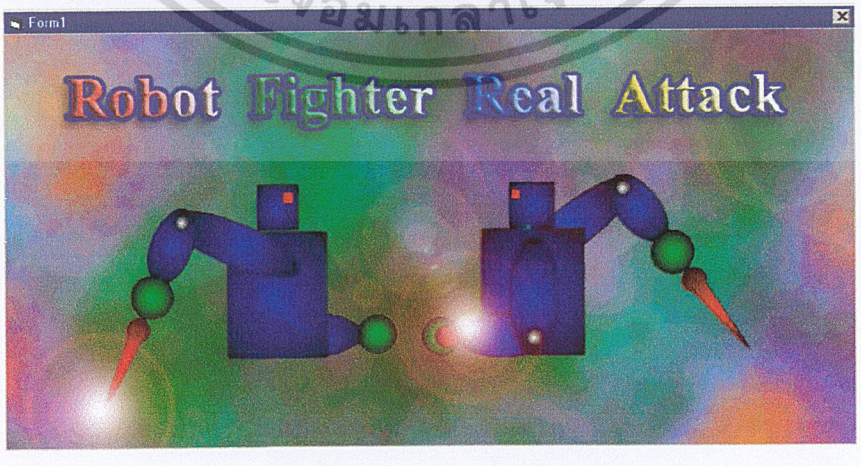
วิธีการเล่นเครื่องเล่น Robot Fighter สำหรับคอมพิวเตอร์

1. เริ่มเล่นเครื่องเล่นเกมนี้ให้นำพอร์ต Serial Port ทั้ง 2 พอร์ต จากเครื่องเล่นต่อเข้ากับคอมพิวเตอร์ตาม Serial Port คอม 1 และคอม 2 เปิดสวิตซ์เครื่อง หลังจากนั้นที่คอมเปิดโปรแกรม Robot Attack เป็นโปรแกรมเล่นเกม หน้าแรกจะปรากฏเมนูต่างๆของเกม เมื่อเริ่มเล่นเกมให้ไปคลิ๊กที่ปุ่ม START ดังรูป A



รูปภาพ A แสดงหน้าแรกของโปรแกรมเล่นเกม

2. เลือกไปที่ START จะปรากฏภาพโปรแกรมที่ใช้เล่นเครื่องเล่น ดังรูป b เมื่ออยู่ในรูปหน้านี้ก็สามารถเริ่มเล่นกันได้แล้ว

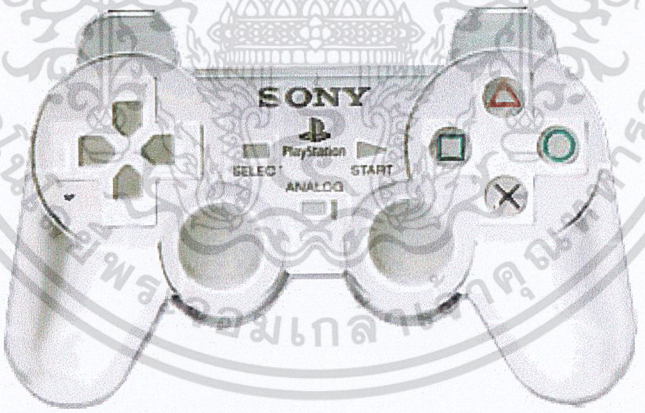


รูปภาพ B แสดงภาพโปรแกรมใช้เล่นเครื่องเล่น

3. จากรูป B โปรแกรมเล่นสามารถบังคับเล่นได้ 2 คน จากคีย์บอร์ดโดยผู้เล่นคนที่ 1 บังคับ Robot Blue ใช้ปุ่มคีย์บอร์ด 1, 2, 3, 4, 5 โดยกด 1 ใช้ในการเคลื่อนที่ซ้าย กด 2 เคลื่อนที่ขวา กด 3 โจมตีด้านบน กด 4 โจมตีด้านข้าง กด 5 จะเป็นบราเรีย ส่วนผู้เล่นคนที่ 2 บังคับ Robot Green ใช้ปุ่มคีย์บอร์ด 6, 7, 8, 9, 0 โดยกด 6 ใช้ในการเคลื่อนซ้าย กด 7 เคลื่อนขวา กด 8 โจมตีด้านบน กด 9 โจมตีด้านข้าง กด 0 บราเรีย

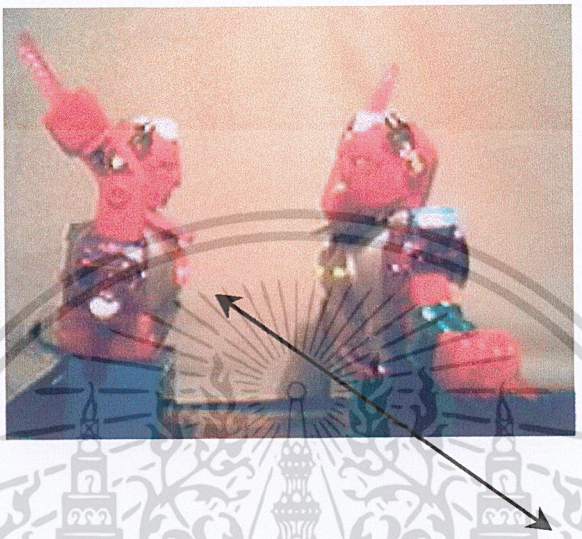
4. สามารถควบคุมได้อีกทางโดยการใช้จอยเล่นได้ จอยที่ใช้ก็เป็นจอยเครื่อง PlayStation สามารถเล่นได้โดยเปิดโปรแกรม Joy PS เป็นโปรแกรมใช้จอย Play เล่นโดยต่อจากพอร์ต Parallel ออกจะมีตัวต่อแถบแยกออกเล่นได้ 2 จอย การคอนโทรลโดยใช้จอย Play

- กดปุ่มซ้าย, ขวา ทำการเคลื่อนไหว ซ้ายและขวาของหุ่น
- ปุ่มสี่เหลี่ยมจะทำการโจมตีด้านบน
- ปุ่มสามเหลี่ยมโจมตีด้านข้าง
- ส่วนปุ่มวงกลมจะเป็นบราเรีย



รูปภาพ C ใช้จอย Play ควบคุมหุ่นยนต์

5. ที่ตัวหุ่นจะมีไฟที่ ดวงตา, อก, ไหล่ทั้งสองข้างแสดงสภาวะพร้อมจะรับการควบคุม การเคลื่อนที่ของหุ่นไปได้ 2 ทิศทาง คือ ทั้งซ้าย และขวา โดยจะมีมอเตอร์ควบคุมสายพานคอยหมุนทำให้หุ่นเคลื่อนไหวได้ การบังคับนี้ใช้ในหาจังหวะการโจมตีและ การหลบหลีกการโจมตี ดังรูป D

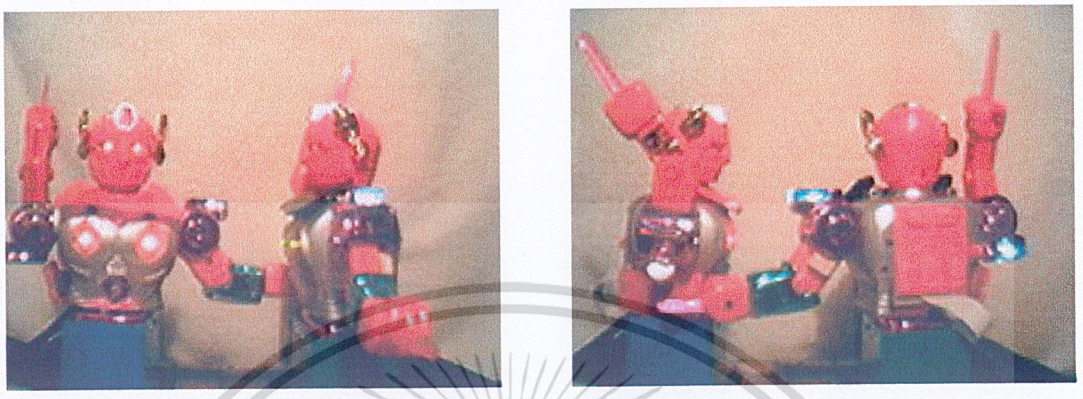


รูปภาพ D ลูกศรแสดงการเคลื่อนที่ซ้ายขวาของหุ่น

6. การโจมตีด้านบนจากการคอนโทรลของจอย คือปุ่มสี่เหลี่ยม ตัวหุ่นเมื่อโจมตีจะมีแสงที่ตากระพริบหนึ่งครั้ง แล้วแขนหุ่นจะโจมตีลงมา โดยจะมีแสงจากดาบ (ทำเป็นดาบเลเซอร์) ในขณะโจมตี แสดงดังรูป E

รูปภาพ E แสดงการโจมตีดาบเลเซอร์ด้านบนของหุ่น

7. การโจมตีด้านข้างจากการคอนโทรลของจอย คือปุ่มสามเหลี่ยม ตัวหุ่นจะโจมตีจะมีแสงที่ตากระพริบหนึ่งครั้งแล้วแขนหุ่นจะหมุนตัวโจมตีด้านข้าง โดยจะมีแสงจากดาบขณะโจมตี แสดงดังรูป F



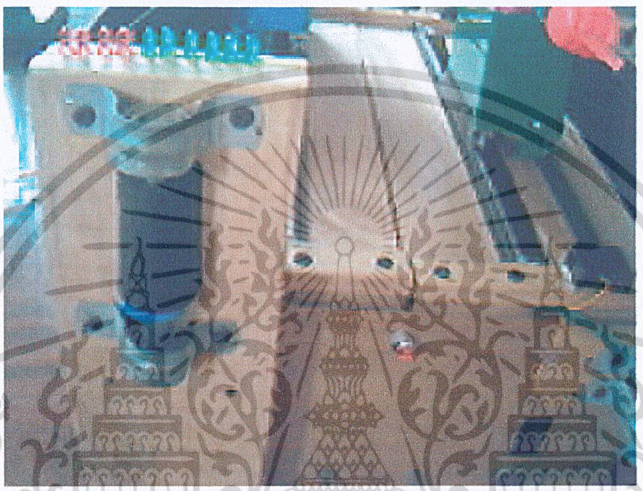
รูปภาพ F แสดงการโจมตีด้านข้างของหุ่นทั้งสอง

8. การใช้บราเรียป้องกันพลังจากการโจมตีทั้งหมด การคอนโทรลของจอย คือปุ่มวงกลม ตัวหุ่นปกติจะมีแสงไฟจากตัวหุ่น เมื่อกดบราเรียจากที่มีแสงของหุ่นจะดับหนึ่งวินาที ช่วงนี้เมื่อถูกโจมตีพลังงานจะไม่ลดแสดงดังรูป G

ก่อนใช้บราเรีย **หลังใช้บราเรีย**

รูปภาพ G แสดงภาพก่อนใช้บราเรียและหลังใช้บราเรียไฟที่ตัวหุ่นจะดับหนึ่งวินาที

9. หุ่นเมื่อถูกโจมตีพลังงานของหุ่นยนต์จะลดลง ไฟพลังงานของหุ่นแต่ละตัวจะมีอยู่ 10 ดวง 6 ดวงแรกจะเป็นไฟสีเขียว ส่วนดวงที่ 7-10 จะเป็นไฟสีแดง เพื่อเตือนว่าพลังงานของหุ่นยนต์ใกล้จะหมดแล้ว เมื่อไฟของหุ่นหมด ไฟที่ตัวหุ่นจะกระพริบจนดับ (เปรียบแสดงว่าหุ่นกำลังระเบิด)หลังจากนั้นหุ่นตัวที่พลังงานหมดหุ่นก็ไม่สามารถบังคับได้อีกแล้ว เป็นอันว่าหุ่นตัวไหนหมดก่อนแสดงว่าแพ้ เมื่อจะเล่นเกมใหม่จะมีสวิตซ์รีเซ็ตอยู่ข้างๆ เมื่อกดหุ่นก็จะเซ็ตเริ่มเกมใหม่ได้ ดังแสดงในรูป H



รูปภาพ H แสดงภาพ LED แสดงไฟพลังงานของหุ่นยนต์และปุ่มกดสวิตซ์รีเซ็ต

ภาคผนวก ข.

Source Code MCS-51 (CONTROL FOR ROBOT)

```
1:   ;***********************************
2:   ;
3:   ;           PROJECT [DESIGN OF COMPUTER ENTERTAINMENT SYSTEM]
4:   ;           (ROBOT FIGHTER REAL ATTACK)
5:   ;           BY (MATHEE & CHAINALONG)
6:   ;           DESIGN MATHEE & CHAINALONG
7:   ;           PROGRAMER MCS-51 BY HIM
8:   ;
9:   ;***********************************
10:  ;           MAIN PROGRAM
11:  ;***********************************
12:
13:              ORG   0000H              ; RESET VECTOR
14:              AJMP  MAIN               ; JUMP TO MAIN
15:
16:              ORG   0023H              ; TI+RI VECTOR
17:              LJMP  GO_TO_INT          ; GOTO SERIAL INTERRUPT SUBROUTINE
18:
19:  MAIN:       MOV   TMOD,#021H         ; T1 8BIT AUTO, TO 16BIT
20:              MOV   TH1,#0FDH          ; 9600 bps TIME1 DEFAULT
21:              MOV   TL1,#0FDH          ;
22:              MOV   IE,#10010000B      ; En. EA, ES
23:              SETB  TR1                ; START TIMER1
24:              MOV   SCON,#050H         ; MODEL RX ENABLE
25:              CLR   RI                 ; CLEAR RX_OK FLAG
26:              MOV   P0,#11111111B
27:              MOV   P1,#01111111B
28:              MOV   P2,#11000000B
29:              SETB  P3.2
30:              CLR   P3.3
31:              MOV   R0,#00001010B
32:              MOV   R1,#1
33:              MOV   R2,#1
34:
35:  ;***********************************
36:  ;           RECEIVE DAMAGE
37:  ;***********************************
38:
39:  LOOP:       MOV   P1,#01111111B
40:              CJNE  R1,#1,DM1          ; CHECK DAMAGE FROM ROBOT WHILE MOVING
41:              CJNE  R2,#1,DM2
42:              LJMP  SWITCH1
43:  DM1:        CLR   ES
44:              LCALL DAMAGE
45:              CLR   RI
46:              SETB  ES
47:              LCALL ENERGY
48:              CLR   TI
49:              CLR   A
50:              MOV   A,#31H             ;********** ROBOT2 CHANGE 33 *********
     ***
51:              SETB  TI
52:              MOV   R1,#1
53:              LJMP  LOOP
54:
55:  DM2:        CLR   ES
56:              LCALL DAMAGE
57:              CLR   RI
58:              SETB  ES
59:              LCALL ENERGY
60:              CLR   TI
61:              CLR   A
62:              MOV   A,#32H             ;********** ROBOT2 CHANGE 34 *********
     ***
```

```
63:              SETB  TI
64:              MOV   R2,#1
65:              LJMP  LOOP
66:
67:  SWITCH1:    JB    P1.0,SWITCH2       ; RECEIVE DAMAGE SWITCH1
68:              CLR   ES                 ; CLEAR INTERUPT
69:              LCALL DAMAGE
70:              CLR   RI                 ; CLEAR BUFFER (PROTECT WAIT BUFFER)
71:              SETB  ES                 ; SET READY SERIAL INTERUPT
72:              LCALL ENERGY             ; REDUCE ENERGY
73:              CLR   TI
74:              CLR   A
75:              MOV   A,#31H             ;********** ROBOT2 CHANGE 33 *********
     ***
76:              SETB  TI
77:              LJMP  LOOP
78:
79:  SWITCH2:    JB    P1.1,LOOP          ; RECEIVE DAMAGE SWITCH2
80:              CLR   ES                 ; CLEAR INTERUPT
81:              LCALL DAMAGE
82:              CLR   RI                 ; CLEAR BUFFER (PROTECT WAIT BUFFER)
83:              SETB  ES                 ; SET READY SERIAL INTERUPT
84:              LCALL ENERGY             ; REDUCE ENERGY
85:              CLR   TI
86:              CLR   A
87:              MOV   A,#32H             ;********** ROBOT2 CHANGE 34 *********
     ***
88:              SETB  TI
89:              LJMP  LOOP
90:
91:  DAMAGE:     CLR   P1.2               ; SHOW DAMAGE (EYE)
92:              CLR   P1.5               ; SHOW DAMAGE (BODY)
93:              CLR   P1.6               ; SHOW DAMAGE (LEFT)
94:              CLR   P3.2               ; SHOW DAMAGE (RIGHT)
95:              LCALL DELAY_100ms
96:              SETB  P1.2               ; SHOW DAMAGE (EYE)
97:              SETB  P1.5               ; SHOW DAMAGE (BODY)
98:              SETB  P1.6               ; SHOW DAMAGE (LEFT)

99:              SETB  P3.2               ; SHOW DAMAGE (RIGHT)
100:             LCALL DELAY_100ms
101:             CLR   P1.2
102:             CLR   P1.5
103:             CLR   P1.6
104:             CLR   P3.2
105:             LCALL DELAY_100ms
106:             SETB  P1.2
107:             SETB  P1.5
108:             SETB  P1.6
109:             SETB  P3.2
110:             LCALL DELAY_100ms
111:             CLR   P1.2
112:             CLR   P1.5
113:             CLR   P1.6
114:             CLR   P3.2
115:             LCALL DELAY_100ms
116:             SETB  P1.2
117:             SETB  P1.5
118:             SETB  P1.6
119:             SETB  P3.2
120:             RET
121:
122: ;***********************************
123: ;    SHOW ENERGY (HP)
124: ;***********************************
```

```
125:
126: ENERGY:      CJNE  R0,#00001010B,ENERGY1    ;CHECK COMPARE ENERGY
127:              DEC   R0
128:              CLR   P2.6
129:              RET
130:
131: ENERGY1:     CJNE  R0,#09H,ENERGY2
132:              DEC   R0
133:              CLR   P2.7
134:              RET
135:
136: ENERGY2:     CJNE  R0,#08H,ENERGY3
137:              DEC   R0
138:              CLR   P0.7
139:              RET
140:
141: ENERGY3:     CJNE  R0,#07H,ENERGY4
142:              DEC   R0
143:              CLR   P0.6
144:              RET
145:
146: ENERGY4:     CJNE  R0,#06H,ENERGY5
147:              DEC   R0
148:              CLR   P0.5
149:              RET
150:
151: ENERGY5:     CJNE  R0,#05H,ENERGY6
152:              DEC   R0
153:              CLR   P0.4
154:              RET
155:
156: ENERGY6:     CJNE  R0,#04H,ENERGY7
157:              DEC   R0
158:              CLR   P0.3
159:              RET
160:
161: ENERGY7:     CJNE  R0,#03H,ENERGY8
162:              DEC   R0
163:              CLR   P0.2
164:              RET
165:
166: ENERGY8:     CJNE  R0,#02H,ENERGY9
167:              DEC   R0
168:              CLR   P0.1
169:              RET
170:
171: ENERGY9:     CJNE  R0,#01H,BACK
172:              CLR   P0.0
173:              AJMP  WE_LOSS
174:
175: BACK:        RET
176:
177: ;*********************************
178: ;            ROBOT LOSS
179: ;*********************************
180:
181: WE_LOSS:     CLR   EA
182:              LCALL DELAY_500ms
183: NO_ENERGY:   MOV   R3,#4
184: NO_ENERGY1:  DEC   R3
185:              CJNE  R3,#0,NO_ENERGY2
186:              AJMP  NO_ENERGY3
187:
188: NO_ENERGY2:  MOV   P0,#00000000B
189:              MOV   P2,#00000000B
```

```
190:              CLR   P1.2
191:              CLR   P1.5
192:              CLR   P1.6
193:              CLR   P3.2
194:              LCALL DELAY_100ms
195:              SETB  P1.2
196:              SETB  P1.5
197:              SETB  P1.6
198:              SETB  P3.2
199:              MOV   P0,#11111111B
200:              MOV   P2,#11000000B
201:              LCALL DELAY_100ms
202:              AJMP  NO_ENERGY1
203:
204: NO_ENERGY3:  MOV   P0,#00000000B
205:              MOV   P2,#00000000B
206:              CLR   P1.2
207:              CLR   P1.5
208:              CLR   P1.6
209:              CLR   P3.2
210:              LCALL DELAY_1s
211:              SETB  P1.2
212:              SETB  P1.5
213:              SETB  P1.6
214:              SETB  P3.2
215:              MOV   P0,#11111111B
216:              MOV   P2,#11000000B
217:              LCALL DELAY_1s
218:
219: NO_ENERGY4:  CLR   P1.2
220:              CLR   P1.5
221:              CLR   P1.6
222:              CLR   P3.2
223:              MOV   P0,#00000000B
224:              MOV   P2,#00000000B
225:              LCALL DELAY_1s
226:              AJMP  NO_ENERGY4
227:
228:
229: ;***********************************
230: ;           SERIAL INTERRUPT
231: ;***********************************
232:
233: GO_TO_INT:   PUSH  ACC
234:              JBC   RI,RX_STATUS      ; RX OR TX? IF RXTHEN RECEIVED AND CLEA
     R RI
235:
236: TX_CONTROL:  CLR   TI                ; CLEAR TI
237:              MOV   SBUF,A
238: TX_WAIT:     JBC   TI,TO_LOOP        ; WAIT UNTILL TX ALREADY(TI=1)THEN CLEA
     R TI
239:              SJMP  TX_WAIT           ; Wait for TX
240: TO_LOOP:     LJMP  EXIT
241: ;-----------------------------------
242: ;     ROBOT (MOVE,ATTACK,SHIELD)
243: ;-----------------------------------
244:
245: RX_STATUS:   CLR   RI                ; CLEAR RI
246:              MOV   A,SBUF            ; GET DATA TO ACC.
247:              MOV   R1,#1
248:              MOV   R2,#1
249:
250: KEY_L:       JNB   P1.3,KEY_R        ; LIMIT LEFT
251:              CJNE  A,#31H,KEY_R      ; MOVE LEFT ***********  ROBOT2 CHANGE )
     36 ************
```

```
252:              SETB  P2.0
253:              LCALL CHECK_200ms
254:              CLR   P2.0
255:              CLR   RI
256:              LJMP  EXIT
257:
258: KEY_R:       JNB   P1.4,KEY_Y          ; LIMIT RIGHT
259:              CJNE  A,#32H,KEY_Y        ; MOVE RIGHT***********  ROBOT2 CHANGE
     37 ************
260:              SETB  P2.1
261:              LCALL CHECK_200ms
262:              CLR   P2.1
263:              CLR   RI
264:              LJMP  EXIT
265:
266: KEY_Y:       CJNE  A,#33H,KEY_B        ; ATTACK ON ***********  ROBOT2 CHANGE
     38 ************
267:              CLR   P1.2                ; SIGNAL ATTACK
268:              LCALL CHECK_200ms         ; SIGNAL ATTACK
269:              SETB  P1.2                ; SIGNAL ATTACK
270:              SETB  P3.3                ; SWORD LASER
271:              LCALL CHECK_600ms
272:              SETB  P2.2                ; ATTACK
273:              LCALL CHECK_500ms
274:              CLR   P2.2
275:              SETB  P2.3
276:              LCALL CHECK_600ms
277:              CLR   P2.3
278:              LCALL CHECK_1s
279:              CLR   P3.3
280:              CLR   RI
281:              LJMP  EXIT
282:
283: KEY_B:       CJNE  A,#34H,KEY_LR       ; ATTACK SIDE;*********  ROBOT2 CHANGE
     39 ************
284:              CLR   P1.2                ; SIGNAL ATTACK
285:              LCALL CHECK_200ms         ; SIGNAL ATTACK
286:              SETB  P1.2                ; SIGNAL ATTACK
287:              SETB  P1.7                ; SWORD LASER
288:              SETB  P2.5                ; ATTACK
289:              LCALL CHECK_1s
290:              LCALL CHECK_350ms
291:              CLR   P2.5
292:              SETB  P2.4
293:              ACALL CHECK_1s
294:              LCALL CHECK_300ms
295:              CLR   P2.4
296:              LCALL CHECK_600ms
297:              CLR   P1.7
298:              CLR   RI
299:              LJMP  EXIT
300:
301: KEY_LR:      CJNE  A,#35H,T1           ; SHIELD   ;***********  ROBOT2 CHANGE
     30 ************
302:              MOV   P1,#00011011B
303:              CLR   P3.2
304:              LCALL DELAY_1s
305:              MOV   P1,#01111111B
306:              SETB  P3.2
307:              CLR   RI
308:              LJMP  EXIT
309:
310: ;-------------------------------------
311: ;      ROBOT (MOVE,ATTACK,SHIELD)
312: ;-------------------------------------
```

```
313: T1:          CJNE  A,#41H,T2          ; MOVE LEFT;***********  ROBOT2 CHANGE
     45 ************
314:              SETB  P2.3
315:              LCALL CHECK_50ms
316:              CLR   P2.3
317:              CLR   RI
318:              LJMP  EXIT
319:
320: T2:          CJNE  A,#42H,T3          ; MOVE RIGHT;**********  ROBOT2 CHANGE
     46 ************
321:              SETB  P2.2
322:              LCALL CHECK_50ms
323:              CLR   P2.2
324:              CLR   RI
325:              LJMP  EXIT
326:
327: T3:          CJNE  A,#43H,T4          ; MOVE LEFT;***********  ROBOT2 CHANGE
     47 ************
328:              SETB  P2.4
329:              LCALL CHECK_50ms
330:              CLR   P2.4
331:              CLR   RI
332:              LJMP  EXIT
333:
334: T4:          CJNE  A,#44H,EXIT        ; MOVE RIGHT;**********  ROBOT2 CHANGE
     48 ************
335:              SETB  P2.5
336:              LCALL CHECK_50ms
337:              CLR   P2.5
338:              CLR   RI

339:
340: EXIT:        POP   ACC                ; POP ACC. TO MAIN
341:              RETI
342:
343: ;-----------------------------------
344: ;             CHECK DAMAGE
345: ;-----------------------------------
346:
347: CHECK_DM:    MOV   P1,#01111111B
348: CHECK_DM1:   CJNE  R1,#1,CHECK_DM2
349:              JB    P1.0,CHECK_DM2
350:              DEC   R1
351:              RET
352:
353: CHECK_DM2:   CJNE  R2,#1,OUT
354: CHECK_DM3:   JB    P1.1,OUT
355:              DEC   R2
356: OUT:         RET
357:
358: ;*********************************
359: ;             DELAY TIME
360: ;*********************************
361:
362: DELAY_10ms:        MOV   R7,#010            ; Do 10 times
363: DELAY_10ms_1:      MOV   R6,#0E6H           ; Each loop = 1 ms
364: DELAY_10ms_2:      NOP
365:                    NOP
366:                    DJNZ  R6,DELAY_10ms_2
367:                    DJNZ  R7,DELAY_10ms_1
368:                    RET
369:
370: DELAY_50ms:        MOV   R4,#5              ; Do 50 times
371: DELAY_50ms_1:      ACALL DELAY_10ms
372:                    DJNZ  R4,DELAY_50ms_1
```

```
373:                    RET
374:
375:
376: DELAY_100ms:       MOV   R4,#10            ; Do 50 times
377: DELAY_100ms_1:     ACALL DELAY_10ms
378:                    DJNZ  R4,DELAY_100ms_1
379:                    RET
380:
381: DELAY_200ms:       MOV   R4,#20            ; Do 50 times
382: DELAY_200ms_1:     ACALL DELAY_10ms
383:                    DJNZ  R4,DELAY_200ms_1
384:                    RET
385:
386: DELAY_500ms:       MOV   R4,#50            ; Do 50 times
387: DELAY_500ms_1:     ACALL DELAY_10ms
388:                    DJNZ  R4,DELAY_500ms_1
389:                    RET
390:
391: DELAY_1s:          MOV   R5,#100           ; Do 100 times
392: DELAY_1s_1:        ACALL DELAY_10ms
393:                    DJNZ  R5,DELAY_1s_1
394:                    RET
395:
396: DELAY_2s:          MOV   R5,#200           ; Do 100 times
397: DELAY_2s_1:        ACALL DELAY_10ms
398:                    DJNZ  R5,DELAY_2s_1
399:                    RET
400: ;-----------------------------------------
401: ;                  DELAY CHECK
402: ;-----------------------------------------
403:
404: CHECK_10ms:        MOV   R7,#010           ; Do 10 times
405: CHECK_10ms_1:      MOV   R6,#070H          ; Each loop = 1 ms
406: CHECK_10ms_2:      NOP
407:                    NOP
408:                    DJNZ  R6,CHECK_10ms_2
409:                    DJNZ  R7,CHECK_10ms_1
410:                    RET
411:
412: CHECK_50ms:        MOV   R4,#5             ; Do 5 times
413: CHECK_50ms_1:      ACALL CHECK_10ms
414:                    ACALL CHECK_DM          ; CHECK DAMAGE
415:                    DJNZ  R4,CHECK_50ms_1
416:                    RET
417:
418: CHECK_100ms:       MOV   R4,#10            ; Do 10 times
419: CHECK_100ms_1:     ACALL CHECK_10ms
420:                    ACALL CHECK_DM          ; CHECK DAMAGE
421:                    DJNZ  R4,CHECK_100ms_1
422:                    RET
423:
424: CHECK_200ms:       MOV   R4,#20            ; Do 20 times
425: CHECK_200ms_1:     ACALL CHECK_10ms
426:                    ACALL CHECK_DM          ; CHECK DAMAGE
427:                    DJNZ  R4,CHECK_200ms_1
428:                    RET
429:
430: CHECK_300ms:       MOV   R4,#30            ; Do 30 times
431: CHECK_300ms_1:     ACALL CHECK_10ms
432:                    ACALL CHECK_DM          ; CHECK DAMAGE
433:                    DJNZ  R4,CHECK_300ms_1
434:                    RET
435:
436: CHECK_350ms:       MOV   R4,#35            ; Do 35 times
437: CHECK_350ms_1:     ACALL CHECK_10ms
```

```
438:                    ACALL CHECK_DM          ; CHECK DAMAGE
439:                    DJNZ  R4,CHECK_350ms_1
440:                    RET
441:
442: CHECK_400ms:       MOV   R4,#40            ; Do 40 times
443: CHECK_400ms_1:     ACALL CHECK_10ms
444:                    ACALL CHECK_DM          ; CHECK DAMAGE
445:                    DJNZ  R4,CHECK_400ms_1
446:                    RET
447:
448: CHECK_500ms:       MOV   R4,#50            ; Do 50 times
449: CHECK_500ms_1:     ACALL CHECK_10ms
450:                    ACALL CHECK_DM          ; CHECK DAMAGE
451:                    DJNZ  R4,CHECK_500ms_1
452:                    RET
453:
454: CHECK_600ms:       MOV   R4,#60            ; Do 60 times
455: CHECK_600ms_1:     ACALL CHECK_10ms
456:                    ACALL CHECK_DM          ; CHECK DAMAGE
457:                    DJNZ  R4,CHECK_600ms_1
458:                    RET
459:
460: CHECK_1s:          MOV   R5,#100           ; Do 100 times
461: CHECK_1s_1:        ACALL CHECK_10ms
462:                    ACALL CHECK_DM          ; CHECK DAMAGE
463:                    DJNZ  R5,CHECK_1s_1
464:                    RET
465:
466: ;****************************************
```

ภาคผนวก ค.

Source Code Visual Basic (SOFTWARE FOR CONTROL ROBOT)

Source Code Form1

```
MSComm1.Settings = "9600,N,8,1"
MSComm1.CommPort = 1
MSComm1.InputLen = 1
MSComm1.PortOpen = True
MSComm1.RThreshold = 1
MSComm2.Settings = "9600,N,8,1"
MSComm2.CommPort = 2
MSComm2.InputLen = 1
MSComm2.PortOpen = True
MSComm2.RThreshold = 1
End Sub

Private Sub Form_KeyDown(KeyCode As Integer, Shift As Integer)
Select Case KeyCode
Case Is = vbKey1: MSComm1.Output = "1"
Case Is = vbKey2: MSComm1.Output = "2"
Case Is = vbKey3: MSComm1.Output = "3"
Case Is = vbKey4: MSComm1.Output = "4"
Case Is = vbKey5: MSComm1.Output = "5"
Case Is = vbKey6: MSComm2.Output = "6"
Case Is = vbKey7: MSComm2.Output = "7"
Case Is = vbKey8: MSComm2.Output = "8"
Case Is = vbKey9: MSComm2.Output = "9"
Case Is = vbKey0: MSComm2.Output = "0"
Case Is = vbKeyZ: End
End Select
End Sub
```

```
Private Sub Timer1_Timer()
If Label6.Left + Label6.Width > 10 Then
   Label6.Move Label6.Left - 300
Else
   Label6.Move 10000
End If
End Sub

Private Sub Timer2_Timer()
If Label8.Left + Label8.Width > 10 Then
   Label8.Move Label8.Left + 30
Else
   Label6.Move 10000
End If
End Sub
```

Source Code Form2

```
Private Sub Command11_Click()
End
End Sub

Private Sub Command1_Click()
MSComm1.Output = "1"
End Sub

Private Sub Command10_Click()
MSComm2.Output = "0"
End Sub

Private Sub Command13_Click()
MSComm1.Output = "A"
```

```
End Sub

Private Sub Command14_Click()
MSComm1.Output = "B"
End Sub

Private Sub Command15_Click()
MSComm1.Output = "C"
End Sub

Private Sub Command16_Click()
MSComm1.Output = "D"
End Sub

Private Sub Command17_Click()
MSComm2.Output = "E"
End Sub

Private Sub Command18_Click()
MSComm2.Output = "F"
End Sub

Private Sub Command19_Click()
MSComm2.Output = "G"
End Sub

Private Sub Command2_Click()
MSComm1.Output = "2"
End Sub
```

```
Private Sub Command20_Click()
MSComm2.Output = "H"
End Sub

Private Sub Command3_Click()
MSComm1.Output = "3"
End Sub

Private Sub Command4_Click()
MSComm1.Output = "4"
End Sub

Private Sub Command5_Click()
MSComm1.Output = "5"
End Sub

Private Sub Command6_Click()
MSComm2.Output = "6"
End Sub

Private Sub Command7_Click()
MSComm2.Output = "7"
End Sub

Private Sub Command8_Click()
MSComm2.Output = "8"
End Sub

Private Sub Command9_Click()
MSComm2.Output = "9"
End Sub
```

```
Private Sub Form_Load()
MSComm1.Settings = "9600,N,8,1"
MSComm1.CommPort = 1
MSComm1.InputLen = 1
MSComm1.PortOpen = True
MSComm1.RThreshold = 1
MSComm2.Settings = "9600,N,8,1"
MSComm2.CommPort = 2
MSComm2.InputLen = 1
MSComm2.PortOpen = True
MSComm2.RThreshold = 1
End Sub
```

Source Code Form3

```
Private Sub Command1_Click()
Form1.Show
Form3.Hide
End Sub

Private Sub Command2_Click()
Form2.Show
Form3.Hide
End Sub

Private Sub Command6_Click()
Label1.Caption = Text1.Text
Label1.Move 250
Timer1.Interval = 100
End Sub
```

```
Private Sub Command7_Click()
End
End Sub

Private Sub Command3_Click()
Form3.Hide
Form4.Show
End Sub

Private Sub Command4_Click()
End
End Sub

Private Sub Form_Load()
End Sub

Private Sub Timer1_Timer()
If Label1.Left + Label1.Width > 10 Then
   Label1.Move Label1.Left - 4
Else
   Label1.Move 600
End If
End Sub
```

Source Code Form4

```
Private Sub Command1_Click()
Form4.Hide
Form3.Show
End Sub
```