MONITOR RS-232C

1. นายไกรเลิศ จึงมาริศกุล. 34131202

2. นายรณฤชัย แจ่มพุ้ง 34131224

โครงงานนี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาอุตสาหกรรมศาสตร์บัณฑิต

ภาควิชาเทคนิคอุตสาหกรรม คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2536

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการ

033145

MONITOR RS-232C

1. MR.KRAILERD CHUNGMARISAKUL 34131202

2. MR.RONRUECHAI JAMFOONG 34131224

Project Report Submitted in Partial Fulfillment of the Requirement for the Bachelor`s Degree Department of Industrial Technology Faculty of Engineering

King Mongkut`s Institute of Technology

1993

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

หัวข้อปริญญานิพจน์	เครื่อง MONITOR RS-232C
ชื่อนักศึกษา	นายไกรเลิศ จิงมาริศกุล
	นายรณชัย แจ่มฟุ้ง
อาจารย์ที่ปรึกษา	อาจารย์ไพศาล สิทธิไชยภาสกุล
ภาควิชา	เทคนิคอุตสาหกรรม
ปีการศึกษา	2536

ภาควิชาเทคนิคอุตสาหกรรม คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง อนุมัติให้นับปริญญานิพนธ์ฉบับนี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาอุตสาหกรรมศาสตร์บัณฑิต

คณะกรรมการสอบ

_____ประธานกรรมการ
( )

_____กรรมการ
( )

_____กรรมการ
( )

_____กรรมการ
( )

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

หัวข้อปริญญานิพนธ์ เครื่อง MONITOR RS-232C

ชื่อนักศึกษา นายไกรเลิศ จิงมาริศกุล

นายรณฤชัย แจ่มฟุ้ง

อาจารย์ที่ปรึกษา อาจารย์ไพศาล สิทธิโยภาสกุล

ภาควิชาเทคนิคอุตสาหกรรม

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2536

บทคัดย่อ

เครื่อง Monitor RS-232C เป็นเครื่องที่ใช้สำหรับเฝ้าดูการสื่อสารข้อมูลในระบบ BUS RS-232C โดยอาศัย IC 8274 Multi Protocol เป็นตัวรับสัญญาณ RS-232C แล้วแปลงเป็นตัวอักษรโดยอาศัยโปรแกรมที่เราเขียนขึ้นมาเพื่องานนี้โดยเฉพาะ ซึ่งจะทำให้เราตรวจสอบหาความผิดพลาดที่เกิดขึ้นในการรับส่งได้ นอกจากนี้ข้อมูลทั้งหมดที่ปรากฏขึ้นเรายังสามารถเก็บลงในแผ่นจานแม่เหล็ก(Disk) เพื่อเรียกดูข้อมูลในภายหลังได้อีกด้วย

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

PROJECT MONITOR RS-232C

STUDENT MR.KRAILERD CHUNGMARISAKUL

MR.RUNRUECHAI JAMFOONG

ASVISOR MR.PAISAN SITHIYOPASAKUL

DEPARTMENT OF INDUSTRIAL TECHNOLOGY

KING MONGKUT'S INSTITUTE OF TECHNOLOGY LADKRABANG

1993

ABSTRACT

MONITOR RS-232C is device for monitor the communication with RS-232C Bus System. IC 8274 Multi Protocol receive data signal from bus RS-232C and decode data signsl from RS-232C to the Alphabet and display the Alphabet on screen. The decoding process depend with program for RS-232C MONITOR. And the error in communication is found by check data on screen. The Data of RS-232C {Alphabet} can save to Hard disk and Read for check error next time.

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

กิตติกรรมประกาศ

ปริญญานิพนธ์เรื่อง เครื่องMonitor RS-232C ฉบับนี้สำเร็จลงได้โดยได้รับความช่วยเหลือและคำแนะนำแนวทางในการทำโครงงานเป็นอย่างดีจาก อาจารย์ไพศาล สิทธิโยภาสกุล ซึ่งเป็นอาจารย์ที่ปรึกษาภาควิชาเทคนิคอุตสาหกรรม รวมทั้งคุณอำนาจ ตันทีปธรรม พนักงานบริษัทการบินไทยที่คอยให้คำปรึกษา และคอยให้ความช่วยเหลือเกี่ยวกับโครงงานเป็นอย่างดี และขอขอบคุณ คุณรุ่งระวี จึงมาริศกุล และคุณสุปราณี นิ่มสุวรรณ ที่ช่วยเหลือในเรื่องงานพิมพ์ ผู้จัดทำใคร่ขอขอบพระคุณทุก ๆ ท่านเป็นอย่างสูง

คณะผู้จัดทำ

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

สารบัญ

บทที่		หน้า
	บทคัดย่อ	ก
	ABSTRACT	ข
	กิตติกรรมประกาศ	ค
1.	บทนำ	
	ความเป็นมา และความสำคัญของปัญหา	1
	วัตถุประสงค์ และขอบเขตของโครงงาน	1
	ลำดับขั้นตอน และวิธีดำเนินงาน	2
	ประโยชน์ที่คาดว่าจะได้รับ	3
2.	ทฤษฎี และหลักการที่เกี่ยวข้อง	
	การรับส่งข้อมูลแบบอนุกรม	4
	BAUDRATE	6
	START BIT	7
	PARITY BIT	9
	STOP BIT	11
	การเปลี่ยนข้อมูลแบบขนานเป็นข้อมูลแบบอนุกรม	12
	หลักการเบื้องต้นของการรับส่งข้อมูลแบบอนุกรม	14
	การสื่อสารข้อมูลตามมาตราฐาน RS-232C	15
	การเชื่อมต่อระบบ RS-232C เข้าด้วยกัน	17
	Serial Asyncronous Data Links	18
	Protocol	20
	Binary Syncronous Protocol	21
	หลักการของ RS-232C Monitor	26

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

3. การออกแบบ และการพัฒนา

รายละเอียดของระบบ 28

การวิเคราะห์ระบบ 28

การติดต่อกับผู้ใช้งาน 29

4. การออกแบบและโครงสร้างทางฮาร์ดแวร์

รายละเอียดเกี่ยวกับ 8255 30

รายละเอียดการจัดเรียงขาของ 8255 32

การต่อ 8255 เข้ากับ CPU 33

8255 READ และ WRITE REGISTER 34

โหมด 0 BASIC REGRSTER I/O 36

การใช้งาน 8255 ในโหมด 1 39

การใช้งาน 8255 ในโหมด 2 45

การออกแบบโครงสร้างทางฮาร์ดแวร์ 47

5. โปรแกรม Monitor RS-232C

โปรแกรมหลัก 56

การติดตั้งตัวอักษรสำหรับการแสดงผล 57

โปรแกรมค่าให้อุปกรณ์ 57

เลือก Function 57

เลือก Function ย่อย 58

กระทำ Function ย่อย 59

แก้ไขการกำหนด Protocol 61

ตรวจจับข้อมูล 62

อ่านโปรแกรมจากแผ่นจานแม่เหล็ก 63

เก็บโปรแกรมลงแผ่นจานแม่เหล็ก 64

เลือกลักษณะการทดสอบ Bit Error Rate 64

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

การทดสอบ Bit Error Rate 65

การแสดงตัวอักษร 66

การออกจากโปรแกรม 67

การแสดงเกี่ยวกับโปรแกรม 67

Flow Chart แสดงการทำงานของ Program 68

เอกสารอ้างอิง 80

ภาคผนวก ก. PROGRAM MONITOR RS-232C

ภาคผนวก ข. 8274 DATA SHEET

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

บทที่ 1
บทนำ

โครงงาน MONITOR RS-232C

ความเป็นมาและความสำคัญของปัญหา

ในปัจจุบัน การใช้งานเครื่องคอมพิวเตอร์มักจะนำมาต่อเชื่อมกันเป็นเครือข่ายหรือนำมาต่อกันโดยผ่านทาง Communication port ทั้งนี้ก็เพื่อที่จะได้สามารถแลกเปลี่ยนข้อมูลซึ่งกันและกันได้ และเพิ่มขีดความสามารถในการใช้งานเครื่องคอมพิวเตอร์ แต่ถ้าหากระบบเครือข่ายนั้นเกิดมีปัญหาในการรับส่งข้อมูลขึ้นมา ซึ่งอาจเกิดจากสาเหตุหลายประการ เช่น มีสัญญาณรบกวนในสายรับส่งสัญญาณ อุปกรณ์ปลายทางเกิดการขัดข้องข้อมูลที่ได้รับเข้ามาบางตัวผิดพลาดเป็นต้น จากปัญหาเหล่านี้เราไม่สามารถทราบได้เลยว่าการแลกเปลี่ยนข้อมูลระหว่างคอมพิวเตอร์นั้นเป็นไปอย่างสมบูรณ์ การแก้ปัญหาจะต้องมีเครื่องมือที่มีประสิทธิภาพสูงในการตรวจจับสัญญาณในสาย วัดอัตราการผิดพลาดข้อมูลในสายได้ จำลองและวิเคราะห์ลักษณะของโปรโตคอลทั้งนี้เพื่อจะได้หาสาเหตุของการขัดข้องได้อย่างถูกต้อง

วัตถุประสงค์ และขอบเขตของโครงงาน

1. เครื่อง Monitor RS-232C จะมีลักษณะเป็นแผ่นวงจรต่อเชื่อมกับเครื่องคอมพิวเตอร์ IBM PC และแสดงผลบนจอภาพใน Text Mode
2. สามารถ Monitor
   1) Asynchronous
3. สามารถรับข้อมูลได้ 3 รหัสคือ EBCDIC ASCII และ HEX
4. อัตราการรับส่งข้อมูลสามารถเลือกได้ 21 อัตรา ดังนี้

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

50 75 110 134.5 150 300 600 1200
1800 2000 2400 3600 4800 7200 9600 12000
14400 19200 บิทต่อวินาที

5. สามารถ Monitor Async. โปรโตคอลมีดังนี้

1) สามารถเลือกความยาวเป็น 5 6 7 8 bits/char

2) สามารถเลือก Parity bit ได้ ดังนี้ ODD EVEN NONE

3) สามารถเลือก Stop bit ได้ 3 แบบคือ 1 1.5 2

6. สามารถ Monitorโปรโตคอลแล้วเก็บสัญญาณที่ตรวจวัดลงในแผ่นจานเหล็ก (Diskette) เพื่อที่จะนำมาตรวจดูความผิดพลาดอีกครั้งได้

7. การแสดงผลสัญญาณที่ Monitor สามารถเลือกได้ 4 แบบคือ

1) แสดงเฉพาะข้อมูลที่ส่งจาก DTE

2) แสดงเฉพาะข้อมูลที่ส่งจาก DCE

3) แสดงทั้งข้อมูลที่ส่งจาก DTE และ DCE

4) แสดงข้อมูลและลักษณะสัญญาณของขาต่าง ๆ

ลำดับขั้นตอนและวิธีดำเนินงาน

1. ศึกษาการสื่อสารข้อมูลแบบอนุกรมบนระบบ RS-232C โครงสร้างและคุณลักษณะของโปรโตคอลแบบ Asynchronous

3. ศึกษาการทำงานของอุปกรณ์แต่ละตัวที่จะนำมาออกแบบวงจร

4. ออกแบบวงจรให้มีคุณสมบัติตามที่ได้กล่าวไว้ในขอบเขตของโครงงาน

5. ประกอบวงจรวิเคราะห์โปรโตคอล

6. ออกแบบ Font เพื่อใช้ในการแสดงผลบนจอภาพ

7. ศึกษาถึงวิธีการเขียนโปรแกรมควบคุม Hardware และการ Interface กับ IBMPC คอมพิวเตอร์

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

8. เขียนโปรแกรมควบคุมการทำงานของวงจรการแสดงผลบนจอภาพ

9. ทดสอบการทำงานของเครื่อง RS232C Monitor ดังนี้

1) ทดสอบการตรวจจับสัญญาณ และการแสดงผล

2) ทดสอบการตรวจจับสัญญาน และการแสดงผลของ DTE DCE

ประโยชน์ที่คาดว่าจะได้รับ

1. เป็นพื้นฐานในการสร้างเครื่องวิเคราะห์โปรโตคอลที่มีประสิทธิภาพต่อไป

2. เป็นประโยชน์ต่อการวิเคราะห์ปัญหา ในด้านการสื่อสารข้อมูลของระบบคอมพิวเตอร์

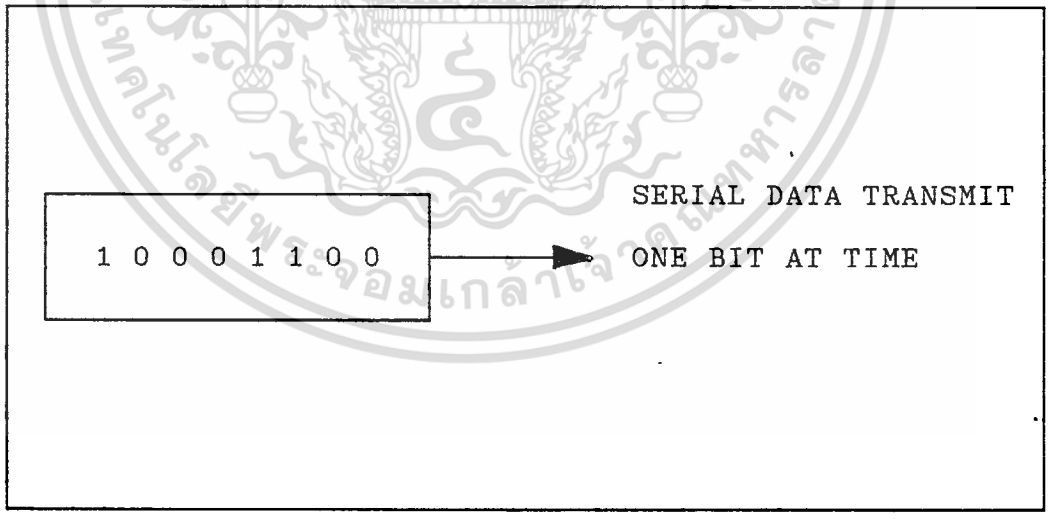
เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

บทที่ 2
ทฤษฎีและหลักการที่เกี่ยวข้อง

การรับส่งข้อมูลแบบอนุกรม

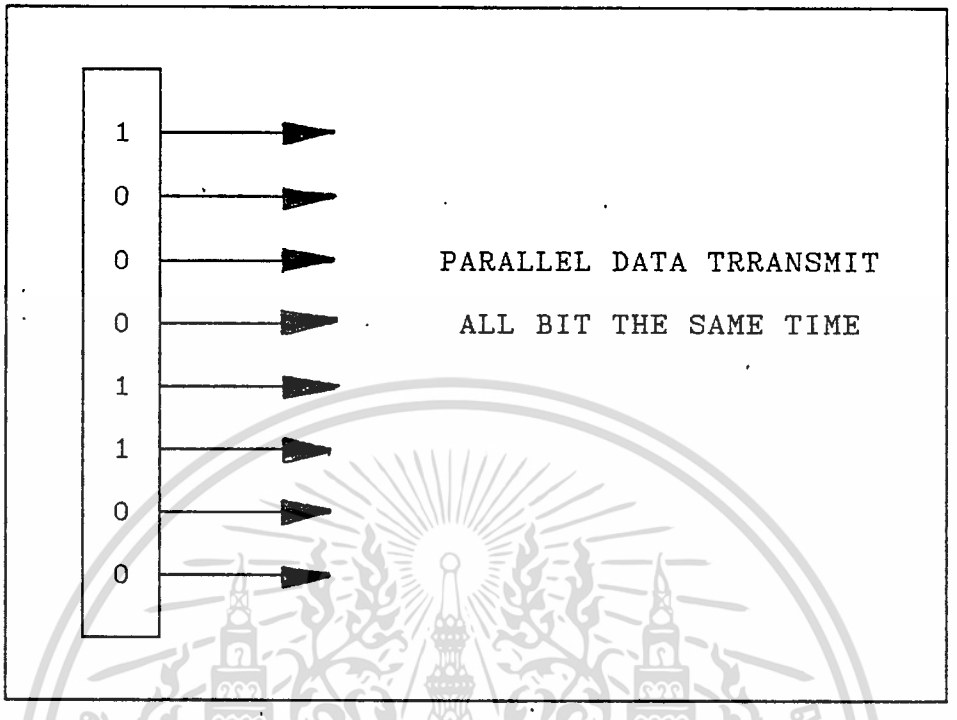
การรับส่งข้อมูลโดยทั่ว ๆ ไปนั้น ข้อมูลทุก ๆ Bit จะถูกรับ หรือส่งออกไปในเวลาเดียวกัน เช่น การอ่าน (Read) หรือเขียน (Write) ข้อมูลลงในหน่วยความจำ (Memory) ซึ่งเราเรียกการรับส่งข้อมูลในลักษณะนี้ว่า "การรับส่งข้อมูลแบบขนาน (PARALLEL COMMUNICATION)"

สำหรับในที่นี้จะกล่าวถึงการรับส่งข้อมูลในอีกรูปแบบหนึ่ง ซึ่งเป็นการรับส่งข้อมูลทีละ Bit แทนที่จะทำการส่งข้อมูลพร้อมกันทุก Bit ในเวลาเดียวกัน การรับส่งข้อมูลแบบนี้มีชื่อว่า "การรับส่งข้อมูลแบบอนุกรม (SERIAL COMMUNICATION)"



รูปที่ 1 แสดงการส่งข้อมูลแบบอนุกรม ซึ่งจะทำการส่งข้อมูลทีละ 1 Bit

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้



รูปที่ 2 แสดง Bit ต่าง ๆ ของข้อมูลที่จะทำการส่งแบบขนาน โดยที่ทุก Bit ของข้อมูลจะถูกส่งออกไปในเวลาเดียวกัน

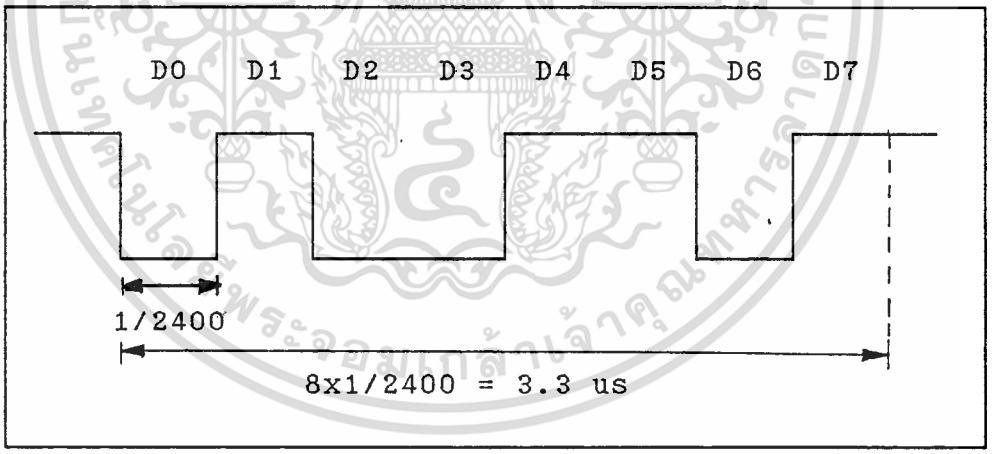
สำหรับการรับส่งข้อมูลแบบขนานนั้น ถึงแม้ว่าจะมีความเร็วสูงกว่าแบบอนุกรมอยู่มากก็ตาม แต่ก็ต้องใช้จำนวนสายในการส่งผ่านข้อมูลเป็นจำนวนมากกว่าแบบอนุกรม ทำให้สิ้นเปลืองค่าใช้จ่ายในการวางสายไปโดยไม่จำเป็น และยังมีการลดทอนของสัญญาณมากกว่าแบบอนุกรมอีกด้วย ทำให้เกิดความผิดพลาดในการส่งผ่านข้อมูลขึ้นได้ง่าย ดังนั้นในการส่งผ่านข้อมูลในระยะทางไกล ๆ เรามักจะเลือกใช้การรับส่งข้อมูลแบบอนุกรม เพื่อลดจำนวนของสายส่งซึ่งจะช่วยในการลดค่าใช้จ่ายในการวางสายลงได้อย่างมาก ถึงแม้ว่าการรับส่งข้อมูลแบบนี้จะมีความยุ่งยากและช้ากว่าการรับส่งข้อมูลแบบขนานอยู่บ้างก็ตาม

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

BAUD RATE

สิ่งที่สำคัญมากสิ่งหนึ่งในการรับส่งข้อมูลแบบอนุกรมนี้ก็คือ ความถี่ที่ใช้ในการส่งข้อมูลซึ่งจะต้องสัมพันธ์กันระหว่างอุปกรณ์ที่ทำการรับและส่งข้อมูล ความถี่ที่ใช้นี้มีชื่อเรียกว่า "BAUD RATE" ซึ่งมีความหมายถึง "อัตราการรับส่งข้อมูลเป็นจำนวน Bit ใน 1 วินาที" ถ้าหากว่าเครื่องส่งใช้ BAUD RATE ที่ไม่สัมพันธ์กับเครื่องรับแล้ว ก็จะทำให้การรับส่งข้อมูลเกิดผิดพลาดขึ้นได้

โดยทั่วไปค่าของ BAUD RATE นั้นจะใช้ค่าต่าง ๆ ดังต่อไปนี้คือ 110, 150 ,300 ,1200 ,2400 ,4800 และ 9600 Bit/วินาที สำหรับในที่นี้จะสมมติว่าต้องการที่จะส่งข้อมูลแบบอนุกรมด้วยอัตรา 2400 BAUD(2400Bit/วินาที) และข้อมูลที่ต้องการจะส่งก็คือ OB2H หรือ 10110010B ซึ่งเราสามารถที่จะแสดงได้ในรูปของสัญญาณดังรูปที่ 3



รูปที่ 3 แสดงสัญญาณของข้อมูลที่ถูกส่งไปตามสายส่งสัญญาณ

จากรูปเห็นได้ว่าความกว้างของสัญญาณของแต่ละ Bit จะมีค่าเท่ากับ 1/BAUD RATE วินาที ซึ่งจาก BAUD RATE ที่เราต้องารใช้คือ 2400 BAUD นั้นจะทำให้ความกว้างของแต่ละBitมีค่าเท่ากับ 1/2400 วินาที หรือเท่ากับ 416 Microseconds ซึ่งจากความกว้างของแต่ละBitที่จะส่งไปตามสายส่งนี้ ทำให้เราสามารถที่จะคำนวณเวลาที่จะต้องใช้ในการรับส่งข้อมูลแต่ละ Byte (8 Bit)

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

ได้ดังนี้คือเท่ากับ 8x416 Microseconds หรือ 3328 Microseconds อย่างไรก็ตามเพื่อป้องกันความผิดพลาดที่อาจเกิดขึ้นได้ จึงมีการเพิ่ม Bit ต่างๆลงไปในแต่ละ Byte ของข้อมูล เพื่อช่วยในการตรวจสอบความถูกต้องของข้อมูลที่เครื่องรับได้รับเข้ามา (แต่ไม่ได้หมายความว่าเมื่อเพิ่ม Bit ต่าง ๆ เหล่านี้แล้วเข้าไปจะทำให้การส่งผ่านข้อมูลมีความถูกต้อง 100%) สำหรับ Bit ต่างๆที่เพิ่มเข้ามานี้ ก็คือ START BIT,STOP BIT และPARITY BIT ซึ่งจะทำให้ข้อมูลในแต่ Byte ที่ส่งออกไปนี้มีมากกว่า 8 bit และเวลาที่ใช้ในการรับส่งข้อมูลก็จะมากขึ้นตามไปนี้

START BIT

ในการส่งผ่านข้อมูลแบบอนุกรมนี้ เราจำเป็นที่จะต้องทำให้อุปกรณ์ที่จะรับข้อมูลทราบว่า ข้อมูลที่ส่งมานั้นเริ่มต้นที่จุดใด ดังนั้นเราจึงจำเป็นที่จะต้องเพิ่มข้อมูล 1 Bit ลงไปก่อนหน้าข้อมูลจริง (AXTUAL DATA) ที่จะทำการส่ง (การส่งอนุกรมจะส่ง Bit D0 เป็น Bit แรก และBit D7 เป็น Bit สุดท้าย) คือทำการเพิ่มBitนี้ลงไปข้างหน้า Bit D0 นั่นเอง และเรียกBitนี้ว่า "START BIT"

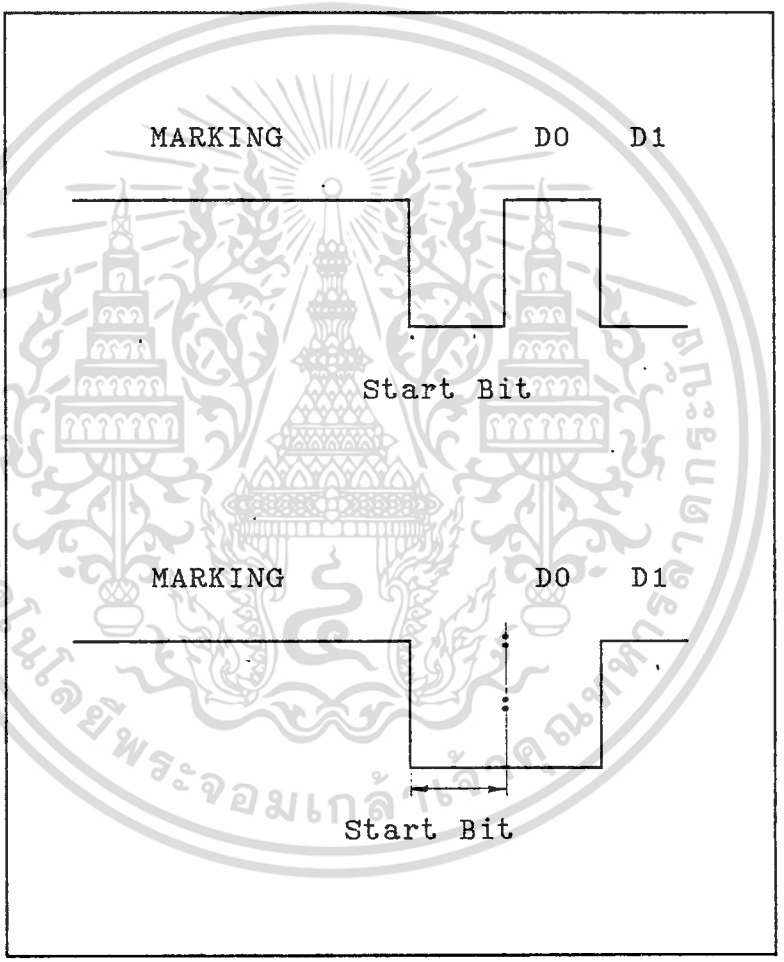
หน้าที่ของ START BIT นี้นอกจากจะใช้ในการบอกว่าข้อมูลนั้น เริ่มต้นที่ใดแล้ว ยังทำงานร่วมกับ STOP BIT (ซึ่งจะกล่าวถึงต่อไป เพื่อช่วยให้การแยกข้อมูลแต่ละชุดออกจากกัน และความกว้างของ Bit นี้จะเท่ากับความกว้างของ Bit อื่น ๆ ในข้อมูลที่จะส่ง (D0-D7)

เมื่ออุปกรณ์ที่จะส่งข้อมูลยังไม่ได้ทำการส่งข้อมุลใด ๆ ออกมานั้น สายส่งจะอยู่ในสภาวะที่เรียกว่า "MARKING" ซึ่งเป็นสภาวะที่ไม่มีการรับส่งข้อมูลใด ๆ เกิดขึ้นในที่นี้เราจะสมมติให้ MARKING ของสายส่งเป็นลอจิก "1" START BIT ที่จะเพิ่มเข้าไปนี้จะมีลอจิกที่ตรงข้ามกับลอจิกของ MARKING ดังนั้นในกรณีนี้ START BIT จะมีลอจิกเป็น "0"

สำหรับ START BIT นี้จะมีความกว้างเท่ากับ 1 Bitของข้อมูล เช่น ใน 1

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

Bitของข้อมูลมีความยาวเท่ากับ 413 Microseconds START BIT ก็จะมีความกว้างของสัญญาณเท่ากับ 413 Microseconds ด้วย ในรูปจะแสดงให้เห็นถึง START BIT ที่เพิ่มเข้าไปก่อนหน้าข้อมูล (ก่อนหน้า D0)



รูปที่ 4 แสดงการเพิ่ม START BIT เข้าไปก่อนหน้า Bit D0 ในกรณีที่ Bit D0 เป็น "1" และ "0" ตามลำดับ

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้



PARITY BIT

สำหรับ Bit นี้จะทำหน้าที่ในการบอกให้ส่วนรับข้อมูลทราบว่า ข้อมูลที่ได้รับเข้ามานั้นมีความถูกต้องเหมือนกับข้อมูลที่ถูกส่งออกมาหรือไม่ (ถึงแม้ว่าการตรวจสอบ Bit นี้จะไม่พบความผิดพลาด แต่ก็ไม่หมายความว่าข้อมูลที่รับเข้ามานี้จะมีความถูกต้อง 100%) โดยที่Bitนี้จะทำหน้าที่ในการบอกให้สวนรับข้อมูลทราบว่าข้อมูลที่ส่งออกมาแต่ละ Byte นั้นมีจำนวน Bit ที่เป็น "1" อยู่เป็นจำนวนคี่ หรือจำนวนคู่ เช่นข้อมูล 54H หรือ 0101011B จะมีจำนวน Bit ที่เป็น "1" อยู่เป็นจำนวนคี่เป็นต้น สำหรับ Bit ที่ใช้ในการตรวจสอบนี้เรียกว่า "PARITY BIT"

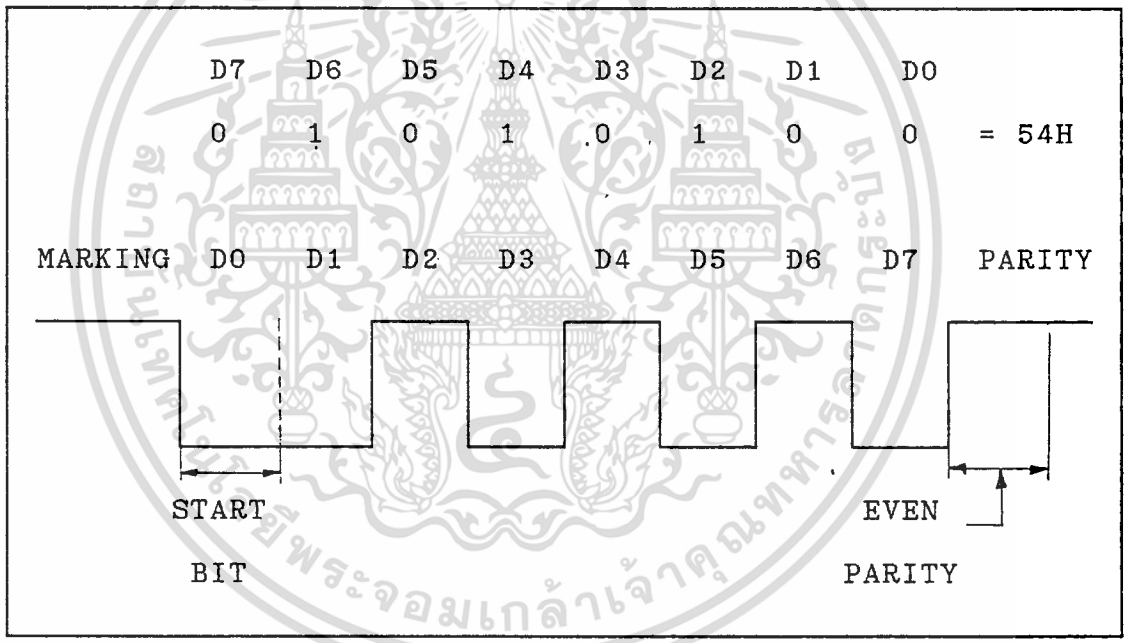
PARITY BIT นี้จะถูกส่งออกมาโดยอุปกรณ์ส่งข้อมูล ซึ่ง Bit นี้จะเป็น "1" หรือ "0" นั้น ขึ้นอยู่กับข้อมูลที่ส่งออกมา (D0-D7) ว่ามีจำนวน Bit ที่เป็น "1" เป็นจำนวนคู่หรือคี่ และยังขึ้นกับอุปกรณ์รับส่งข้อมูลด้วยว่าถูกออกแบบ (โปรแกรม) ไว้ให้รับส่ง PARITY BIT ในลักษณะของ PARITY คู่ หรือ PARITY คี่ อีกด้วย

ในกรณีที่อุปกรณ์รับส่งข้อมูลถูกออกแบบไว้ให้เป็น PARITY คู่ อุปกรณ์ส่งข้อมูลจะทำการส่ง PARITY BIT เป็นลอจิก "1" ออกไปเมื่อจำนวนBitที่เป็น "1" ของข้อมูล (D0-D7) เป็นจำนวนคี่ และจะทำการส่ง PARITY BIT เป็นลอจิก "0" เมื่อจำนวนBitที่เป็น "1" ของข้อมูลเป็นจำนวนคู่ (คือทำให้จำนวนBitที่เห็น "1" ของข้อมูล (D0-D7) รวมกับ PARITY BIT แล้วเป็นจำนวนคู่นั่นเอง) สำหรับ PARITY คี่ก็เช่นกัน คือ PARITY BIT จะเป็น "1" ในกรณีที่จำนวนBitที่เป็น "1" ของข้อมูลเป็นจำนวนคู่ และจะเป็น "0" ในกรณีที่เป็นจำนวนคี่ ในที่นี้จะสมมติว่าอุปกรณ์ถูกออกแบบไว้สำหรับ PARITY คู่ และเราต้องการที่จะส่งข้อมูลออกไปให้กับส่วนรับบข้อมูลเป็นจำนวน 2 Byte คือ 54H และ55H เมื่อเราส่งข้อมูล 54H ออกไป ซึ่งมีจำนวน Bit ที่เป็น "1" เป็นจำนวนคี่ ดังนั้นในกรณีนี้อุปกรณ์ส่งข้อมูลก็จะทำการส่ง PARITY BIT เป็นลอจิก "1" ตามออกมาด้วย เพื่อจะให้จำนวน Bit ที่เป็น "1" ของข้อมูล (54H) รวมกับ PARITY BIT แล้วได้เป็นจำนวนคู่ ส่วนข้อมูล 55H นั้นจำนวนบีเป็น "1" นั้นเป็นจำนวนคู่อยู่แล้ว ดังนั้นอุปกรณ์ส่งข้อมูลก็จะส่ง PARITY BIT เป็น "0" ให้กับส่วนรับข้อมูล ดังในรูปข้างล่าง สำหรับส่วนรับข้อมูลนั้น เมื่อทำการรับข้อมูลเข้ามาแล้วก็จะตรวจสอบสัญญาณว่า จำนวน Bit ที่

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

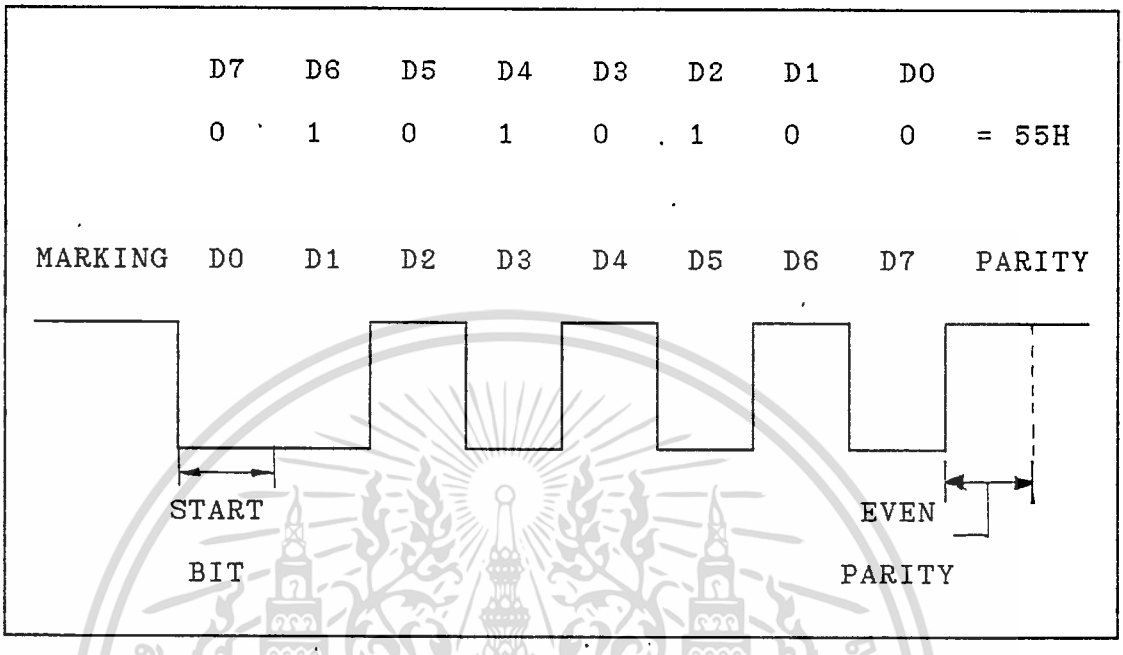
เป็น "1" ของข้อมูลรวมกับ PARITY BIT นั้นเป็นจำนวนคู่หรือไม่ ถ้าหากว่าเป็นจำนวนคี่ก็แสดงว่าข้อมูลที่ได้รับเข้ามามีความผิดพลาดเกิดขึ้น (แต่ไม่ได้หมายความว่า ถ้าเป็นจำนวนคู่แล้วข้อมูลที่รับเข้ามาจะถูกต้องเสมอไป)

สิ่งที่สำคัญอีกสิ่งหนึ่งก็คือ ถ้าอุปกรณ์ส่งข้อมูลทำการส่งในลักษณะ PARITY คู่หรือคี่ก็ตาม ส่วนรับข้อมูลก้จะต้องทำการรับในลักษณะ PARITY เดียวกับอุปกรณ์ส่งข้อมูลด้วย เช่นในกรณีที่อุปกรณ์ส่งข้อมูลทำการส่งข้อมูลในสักษณะของ PARITY คู่ อุปกรณ์รับข้อมูลก็จะต้องทำการรับข้อมูลในลักษณะของ PARITY คู่ด้วย เป็นต้น



รูปที่ 5 แสดงการเพิ่ม PARITY ที่เป็น "1" ลงไปในข้อมูลแต่ละ Byte

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

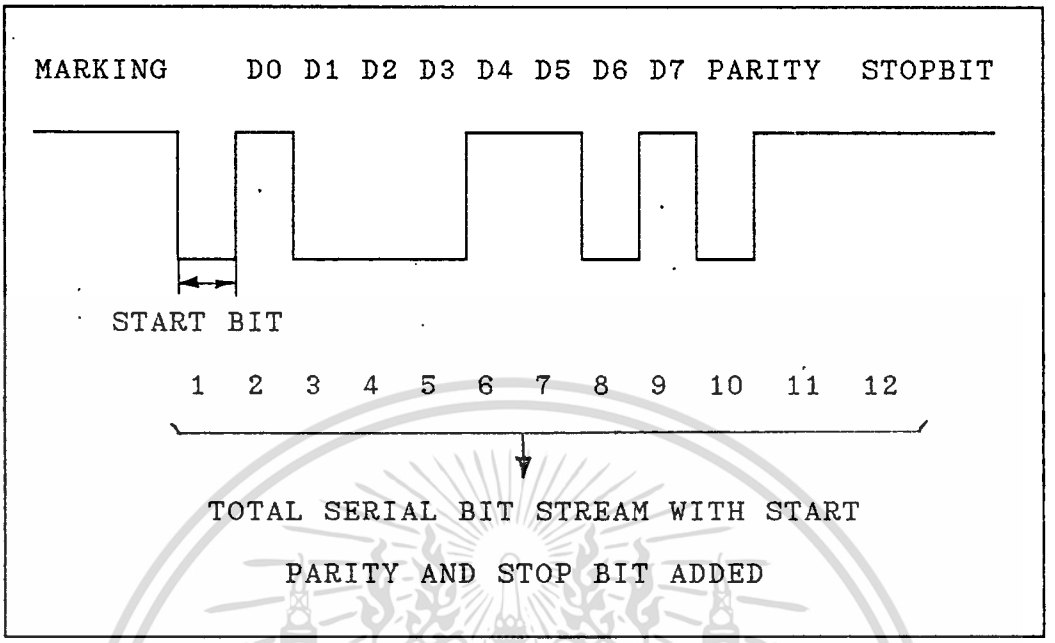


รูปที่ 6 แสดงการเพิ่ม PARITY ที่เป็น "0" ลงไปในข้อมูลแต่ละ Byte

STOP BIT

สำหรับ Bit สุดท้ายที่เพิ่มเข้าไปนี้ จะใช้ในการตรวจสอบจุดสิ้นสุดของข้อมูล Bit นี้จะถูกเพิ่มเข้าไปหลัง PARITY BIT ถ้าอุปกรณ์รับข้อมูลตรวจไม่พบ Bit นี้ก็แสดงว่าข้อมูลที่ได้รับเข้ามานั้นมีความผิดพลาดเกิดขึ้นสำหรับ STOP BIT นี้อาจมีจำนวน Bit เป็น 1 Bit ,1.5 Bit หรือ 2 Bit ก็ได้ รูปที่ 7 นี้จะแสดงข้อมูลทั้ง 8 Bit ที่ส่งออกมารวมทั้ง START BIT ,STOP BIT และPARITY BIT ด้วย ซึ่งจะเห็นว่าสิ่งที่ส่งออกมาในแต่ละ Byte นั้นไม่ใดมีเพียงข้อมูล 8 Bit เท่านั้น แต่อาจจะมีได้ถึง 12 Bit (กรณีที่ส่ง STOP BIT ออกมา 2 Bit) ดังนั้นถ้าเราทำการส่งด้วยอัตรา 2400 BAUD เราจะต้องใช้เวลาส่งทั้งหมดเท่ากับ 12x416 Microseconds หรือ 4.99 Milliseconds ไม่ใช่ 3328 Microseconds ดังที่คำนวณไว้ในตอนต้น

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้



รูปที่ 7 แสดงการเพิ่ม Bit ต่าง ๆ ที่ใช้ใช้สำหรับป้องกันความผิดพลาดที่จะเกิดขึ้นในการรับ ส่งแบบอนุกรม

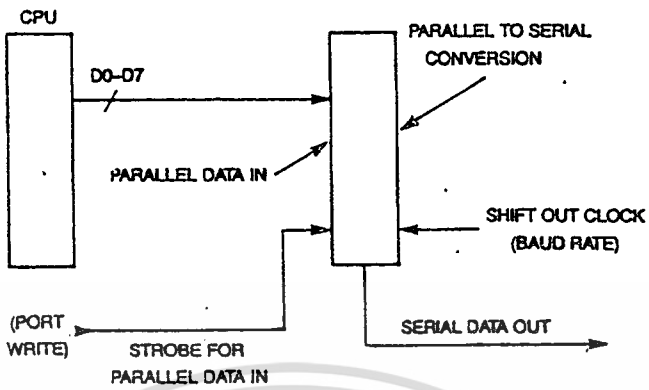
การเปลี่ยนข้อมูลจากแบบขนานเป็นข้อมูลแบบอนุกรม

โดยทั่วไปแล้ว การรับส่งข้อมูลภายในระบบมักจะเป็นข้อมูลแบบขนาน เนื่องจากมีความเร็วในการส่งผ่านข้อมูลที่สูงกว่าแบบอนุกรมมาก และยังมีความยุ่งยากน้อยกว่าอีกด้วย ดังนั้นในการรับสงข้อมูลในระยะทางไกล ๆ ที่จำเป็นจะต้องใช้การส่งผ่านข้อมูลแบบอนุกรม จึงจำเป็นที่จะต้องทำการเปลี่ยนรูปแบบของข้อมูลจากแบบขนาน ไปเป็นแบบอนุกรมก่อนที่จะทำการส่งข้อมูลออกไปตามสายส่ง สำหรับหลักการง่าย ๆ ที่ใช้ในการเปลี่ยนรูปแบบของข้อมูลนั้นมีขั้นตอนดังต่อไปนี้ คือ

1. ทำการเก็บข้อมูลแบบขนาน (ในที่นี้มีจำนวน 8 Bit) ไว้ใน SHIFT REGISTER

2. เลื่อนข้อมูลทั้ง 8 Bit ออกไปให้กับอุปกรณ์รับข้อมูลทีละ Bit โดยที่จะทำการส่งข้อมูลแต่ละ Bit ออกไปด้วยอัตราเดียวกับ BAUD RATE ที่กำหนดไว้

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้



รูปที่ 8 แสดง Block Diagarm ของการเปลี่ยนข้อมูลแบบขนานเป็นแบบอนุกรม

รูปที่ 8 แสดง Block Diagram ของการทำงานทั้ง 2 ขั้นตอน คือ ข้อมูลแบบขนานนั้นจะถูกส่งจาก CPU ให้กับ SHIFT REGISTER จากนั้นจึงทำการเลื่อนข้อมูลออกทีละ Bit ด้วยอัตราของ BAUD RATE ที่กำหนด (โดยเลื่อน Bit D0 ก่อน และ D7 เป็น Bit สุดท้าย)

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

หลักการเบื้องต้นของการรับส่งข้อมูลแบบอนุกรม

สำหรับหลักการเบื้องต้นของ การรับส่งข้อมูลแบบอนุกรม ที่ได้กล่าวถึงนั้น สามารถที่จะสรุปเป็นข้อ ๆ ได้ดังนี้

1.ข้อมูลแบบขนานจากระบบจะถูกเปลี่ยนให้เป็นข้อมูลแบบอนุกรมเพื่อเตรียมที่จะส่งออกไปให้กับส่วนรับข้อมูล

2.ข้อมูลจะถูกส่งออกไปด้วยอัตราคงที่ค่าหนึ่ง ซึ่งเรียกว่า "BAUD RATE" คือ ถ้าทำการส่งข้อมูลด้วยอัตรา 1200 BAUD ก็แสดงว่าเป็นการส่งข้อมูลด้วยอัตรา 1200 Bitต่อ 1 วินาที ซึ่งก็คือ การส่งข้อมูลโดยใช้ความถี่ 1200 Hz นั่นเอง

3.ข้อมูลอนุกรมจะถูกส่งออกไปทีละ Bit โดยทำการส่ง Bit D0 เป็น Bit แรก และBit D7 เป็น Bit สุดท้าย

4.ในขณะที่ยังไม่มีการส่งข้อมูลเข้าไปในสายส่ง สายส่งจะถูกทำให้อยู่ในสภาวะลอจิกใดลอจิกหนึ่ง และเราเรียกสภาวะนี้ว่า "MARKING"

5.อุปกรณ์ส่งข้อมูลจะเพิ่มข้อมูลอีก 1 Bit เข้าไปหน้า Bit D0 ของข้อมูลที่จะส่งให้กับเครื่องรับ Bit ที่เพิ่มเข้าไปนี้เรียกว่า "START BIT" สำหรับ Bit นี้จะมีลอจิกตรงข้ามกับของ MARKING เช่น ถ้าลอจิกของ MARKING เป็น "1" ลอจิกของ Bit นี้ก็จะเป็น "0"

6.อุปกรณ์ส่งข้อมูลจะทำการเพิ่ม PARITY BIT เข้าไปหลัง Bit D7 ของข้อมูล เพื่อใช้ในการตรวจสอบความผิดพลาดของข้อมูลที่เครื่องรับ (สำหรับBitนี้เครื่องส่งอาจจะเพิ่มเข้าไปหรือไม่ก็ได้ ขึ้นกับผู้ออกแบบว่าต้องการที่จะเพิ่ม Bit นี้เข้าไปหรือไม่

7.สำหรับ Bit สุดท้ายที่ถูกเพิ่มเข้าไปหลัง PARITY BIT เรียกว่า "STOP BIT" ซึ่งอาจจะมีจำนวน 1 Bit ,1.5 Bit หรือ 2 Bit ก็ได้ และลอจิกนี้จะเป็นลอจิกเดียวกับลอจิกของ MARKING

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

การสื่อสารข้อมูลตามมาตรฐาน RS-232C มีรายละเอียดพอสังเขปดังนี้

1. สัญญาณที่ขาทุก ๆ ขาของหัวต่อ (Connecter) จะมีระดับของสัญญาณไฟฟ้าเป็นดังนี้

Mark จะมีระดับของสัญญาณไฟฟ้าระหว่าง -25 ถึง -3 โวลท์

Space จะมีระดับของสัญญาณไฟฟ้าระหว่าง +25 ถึง +3 โวลท์

2. ค่าความจุไฟฟ้า (Capacitance) ของอุปกรณ์ปลายทางที่นำมาต่อจะต้องมีค่าไม่เกิน 2500pF ทั้งนี้ยังไม่รวมค่าความจุไฟฟ้าในสายที่ใช้ต่อเชื่อม

3. แรงดันขณะเปิดวงจรหรือไม่โหลดต่อร่วม จะต้องไม่เกิน 25โวลท์

4. วงจรรับสัญญาณที่ต่อร่วมกับ RS-232C ต้องสามารถทนการลัดวงจรได้โดยที่ไม่ทำความเสียหายให้กับอุปกรณ์ต่อร่วมอื่น ๆ

5. ลักษณะของหัวต่อจะต้องมีจำนวน25ขาโดยมากนิยมใช้แบบ DB-25

6. ในทางปฏิบัติ เพื่อให้ได้ข้อมูลที่ถูกต้องความยาวของสายที่นำมาต่อเชื่อมควรมีความยาวไม่เกิน 50 ฟุต

7. รายละเอียดของขาต่าง ๆ ที่สำคัญมีดังนี้

ขา 1 (Protective Ground) ใช้ต่อกับตัวถังเครื่อง

ขา 2 (Transmitted Data) [Tx] ใช้ส่งข้อมูลจาก DTE ไปยัง DCE โดยปกติมีสถานะเป็น Mark เมื่อไม่มีข้อมูลจะส่ง

ขา 3 (Receive Data) [Rx] ใช้รับข้อมูลจาก DCE

ขา 4 (Request to Send) [RTS] ใช้บอกว่า DTE ต้องการที่จะส่งข้อมูล

ขา 5 (Clear to Send)[CTS] เป็นสัญญาณตอบกลับจาก DCEเมื่อได้รับสัญญาณ RTS จาก DTE เพื่อที่จะบอกว่าพร้อมที่จะรับข้อมูลแล้ว

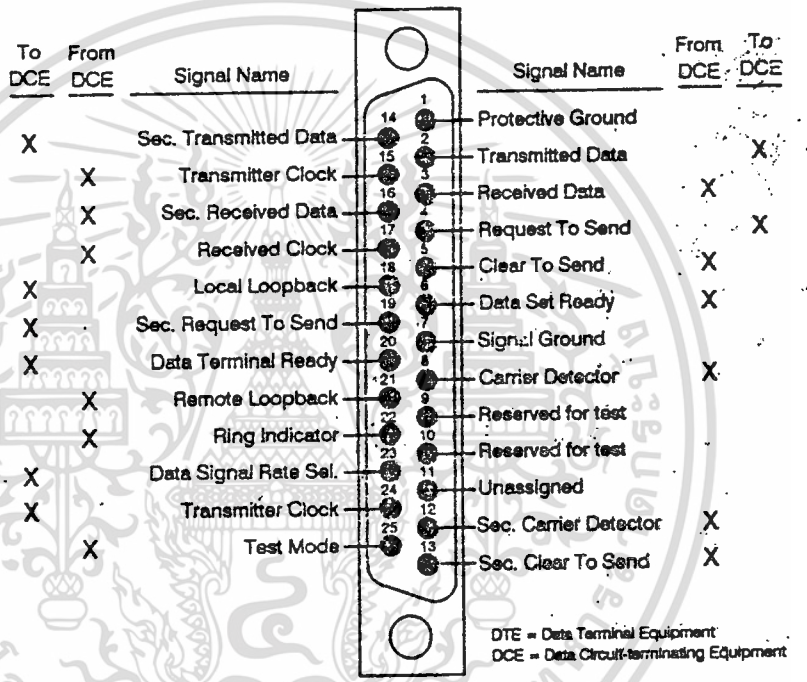
ขา 6 (Data Set Ready) [DSR] ใช้บอกความพร้อมของ DCE ถ้าเป็น Mark จะหมายถึงพร้อมที่จะติดต่อกับ DTE

ขา 7 (Signal Ground) ใช้เป็นจุดอ้างอิงของสัญญาณขาอื่นๆ

ขา 9 (Data Carrier Detector) [DCD] เป็นสัญญาณขา DCE ส่งไปยัง DTE เพื่อบอกว่าติดต่อกับ DCE อีกด้านหนึ่ง(Remote Side)ได้แล้ว

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

ขา 20 (Data terminal Ready) [DTR] ใช้บอกความพร้อม ของ DTE ซึ่งถ้าอยู่ในสภาวะ Mark จะหมายถึงพร้อมที่จะรับส่งข้อมูลกับ DCE

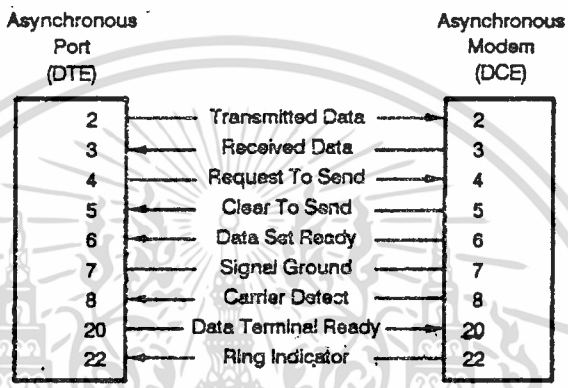


รูปที่ 9 EIA RS-232 pin layout and assignments

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

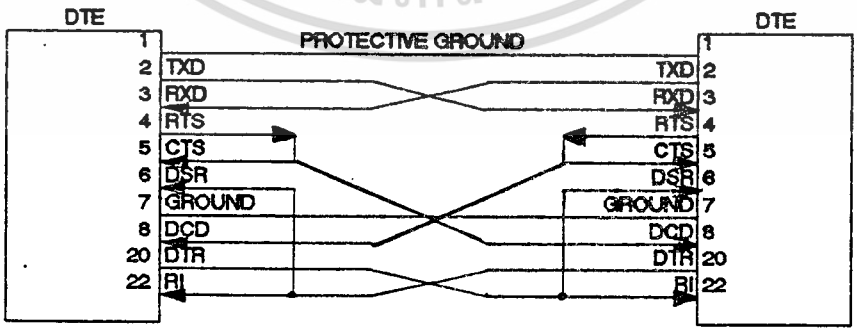
การต่อเชื่อมระบบ RS-232C เข้าด้วยกัน

1) การต่อเชื่อมระหว่าง DTE กับ DCE สามารถทำการเชื่อมต่อกันได้โดยตรง กล่าวคือเชื่อมขาที่เหมือนกันเข้าด้วยกัน ดังแสดงในรูปที่ 10



รูปที่ 10 การต่อเชื่อมระหว่าง DTE กับ DCE

2) การต่อเชื่อมระหว่าง DTE กับ DTE จะต้องมีการสลับสายบางเส้นทั้งนี้เพื่อต้องการที่จะทำให้เหมือนกับว่า DTE ถูกต่อกับ DCE ดังแสดงในรูปที่ 11

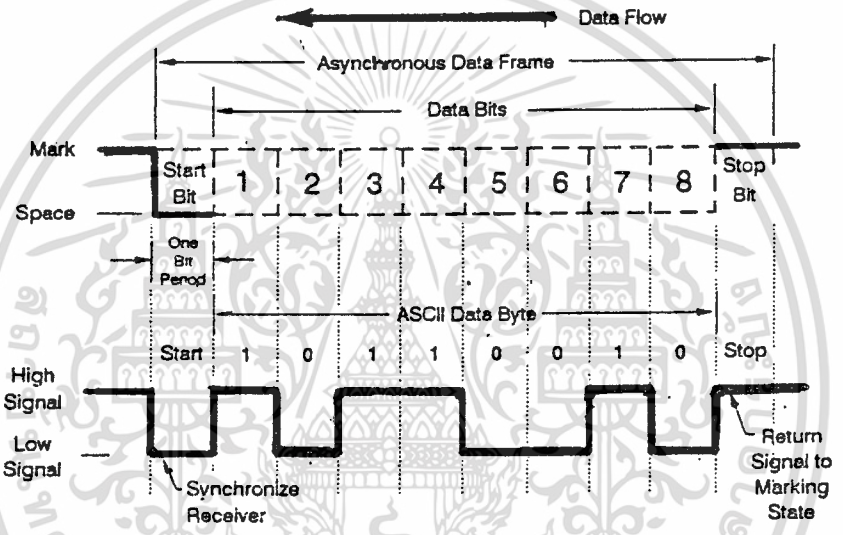


รูปที่ 11 การต่อเชื่อมระหว่าง DTE กับ DTE

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

Serial-Asynchronous Data Links

Serial Asynchronous เป็นวิธีการของการรับส่งข้อมูล ซึ่งใช้ในระบบการรับการส่ง ที่ตัวรับและตัวส่งไม่ต้อง Synchronized เมื่อข้อมูลถูกส่งโดยตัวส่งข้อมูลเป็น Framed โดยใช้ START และ STOP bit framed เป็นตัวบอกให้ตัวรับทราบ ดังรูปที่ 12



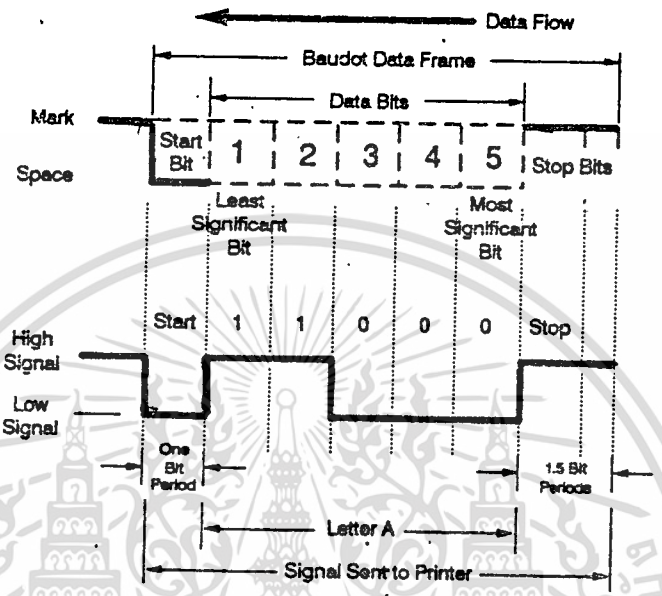
รูปที่ 12 เป็น ASCII 8 Bit/Char กับ None Parity

ปกติแล้วขาข้อมูลจะอยู่ในสภาวะ Idel หรือ Marking state(MARK) หรือ "1" เมื่อส่งข้อมูลออกไปก็จะอยู่ในเปลี่ยนจกาสภาวะ Mark เป็น Space หรือ "0" เรียกว่า Start bit ตัวรับจะทราบทันทีว่าบิตที่ตามมาจะเป็นข้อมูลตัวอักษร รูปที่ 12 แสดงการส่งข้อมูล ASCII 8 bit/Char เป็นตัว M ใหญ่ โดยมี Parity เป็น None ข้อมูลจะส่ง LSB ไปก่อน MSB และ Parity จะส่งตามข้อมูลตัวอักษรทันทีในกรณีที่มี Parity bit และจะเป็น Stop bit ตามมาแสดงว่าเป็นการส่งข้อมูล 1 Frame

ในโหมด Asynchronous ตัวอักษรจะมีขนาดตั้งแต่ 5 บิต ถึง 8 บิต ความยาวของตัวอักษรขึ้นอยู่กับ Code ที่ใช้ เช่น Code 7 bit สำหรับ ASCII

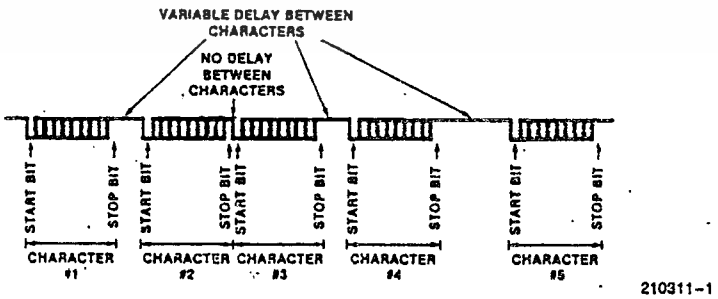
เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

Code 8 bit สำหรับ EBCDIC Code และ ข้อมูล Binary หรือ Baudot Code



รูปที่ 13 Baudot transmission of the letter A

การส่งข้อความข่าวสารซึ่งเป็นข้อมูลหลายตัวอักษรนั้น แต่ละตัวอักษร คือ Frame และจะส่งแยกกันเป็น Frame ดังรูปที่ 14



รูปที่ 14 การส่งข้อมูลตัวอักษรหลายๆ ตัวเป็นข้อความ

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

โปรโตคอล (Protocol)

เป็นรูปแบบของการสื่อสารที่ได้มีการกำหนดไว้เป็นมาตรฐานไว้ ปัจจุบันมีโปรโตคอลอยู่หลายชนิด ซึ่งสามารถแบ่งตามลักษณะของการกระทำกับข้อมูลได้เป็น 2 ประเภทคือ

1. Charecter-Orient Protocol เป็นโปรโตคอลที่ได้มีการแบ่งประเภทของตัวอักษรเอาไว้ โดยที่ตัวอักษรแต่ละตัวจะถูกกำหนดความหมายไว้อย่างชัดเจนว่าใช้เป็นรหัสควบคุมการสื่อสารหรือใช้เป็นข่าวสาร ดังนั้นจึงไม่สามารถส่งข้อมูลที่มีตัวอักษรบางตัวเป็นรหัสควบคุมได้ โปรโตคอลประเภทนี้ที่นิยมใช้ คือ Binary Synchronous Protocol (BiSync หรือ BSC)

2. Bit-Orient Protocol เป็นโปรโตคอลที่มีคุณสมบัติดังนี้

1) เป็น Bit-Orient Protocol นั่นคือ สามารถที่จะส่งตัวอักษรใดๆ ในส่วนของข่าวสารก็ได้ และจะไม่ยึดถือว่าจะมีอักษรตัวใดจะเป็นรหัสควบคุมเหมือนกับ BiSync ดังนั้นจึงสามารถส่งข่าวสารที่เป็นเลข Binaryได้ทั้งหมด

2) สามารถใช้ได้ทั้งการสื่อสารแบบ Full Duplex Half-duplex และสามารถใช้กับการสื่อสารที่มีสถานีปลายทางมากกว่า 1 สถานี (Multi point)ได้อีกด้วย

3) ส่วนตรวจสอบความผิดพลาดจะต้องมีประสิทธิภาพสูง

Protocol ที่ถูกสร้างขึ้นมาโดยอาศัยหลักเกณฑ์ 4 ข้อ ที่กล่าวมานั้น มีอยู่หลายชนิด เช่น

ก. HDLC (High Level Data Link Control)

ข. SDLC (Synchronous Data Link Control)

ค. BDLC (Borroughts Data Link Control)

ง. UDLC (Universal Data Link Control)

จ. ADCCP (Advanced Data Link Control)

โปรโตคอลที่เป็นที่นิยมใช้กันมาก คือ HDLC และ SDLC ซึ่ง SDLC Protocol เป็นส่วนที่เป็นกลุ่มย่อยของ HDLC

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

Binary Synchronous Protocol (BiSync)

เป็นโปรโตคอลแบบที่เรียกว่า Charactor Oriented Protocol ซึ่งหมายถึง ข้อมูลที่กระทำจะมองในลักษณะเป็นไบท์ (Byte) เสมอ ดังนั้นจึงต้องมีการแบ่งกลุ่มของตัวอักษรไว้ดังนี้

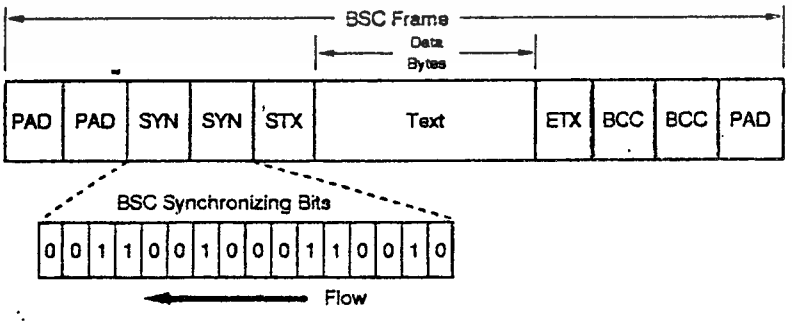
1. กลุ่มของตัวอักษร BiSync Protocol ได้แบ่งกลุ่มของตัวอักษรออกเป็น 3 กลุ่ม คือ

1) Information เป็นกลุ่มตัวอักษรที่เป็นตัวเลข ตัวอักษรและตัวอักษรทางด้าน Graphics ซึ่งจะใช้เป็นข่าวสาร

2) Device Control เป็นกลุ่มของรหัสควบคุมอุปกรณ์ ทางด้านเครื่องรับข้อมูลปลายทางประเภทเครื่องพิมพ์ หรือจอภาพ ตัวอย่างของรหัสเหล่านี้ เช่น form feed new line back space เป็นต้น

3) Transmission Controlเป็นรหัสที่ใช้สำหรับควบคุมการสื่อสารข้อมูล ซึ่งตัวอักษรกลุ่มนี้จะเป็นกลุ่มที่มีความสำคัญในการรับส่งข้อมูลเป็นอย่างมากและจะนำไปเป็นส่วนของข่าวสารไม่ได้ รหัสที่ใช้ควบคุมที่มีใช้ ได้แก่ Sync SOH STX ETB เป็นต้น

2. โครงสร้างของ BiSync Protocol ข้อมูลที่ส่งไปจะต้องเริ่มต้นด้วย รหัส Sync ตามด้วยส่วนของ Header และส่วนของข่าวสารเมื่อจบส่วนที่เป็นข่าวสารจะต้องปิดด้วย ETX หรือ ETB แล้วจึงตามด้วย BCC ดังในรูปที่ 15



รูปที่ 15 แสดงโครงสร้างของ BiSync Protocol (BSC)

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

1) SYN เป็นตัวอักษรซิงค์ ที่ใช้ส่งในขณะที่ไม่มีข้อมูลใด ๆ ส่งออกไป

2) SOH เป็นตัวอักษรบอกการเริ่มต้นของส่วน Header

3) Headerจะประกอบด้วย Address ลำดับของข้อความ Priority ส่วนของ Address จะเป็นตำแหน่งของสถานีปลายทางที่มีอยู่ในสายส่งสัญญาณ ถ้าหากมีสถานีปลายทางมากกว่า 1 สถานีค่า Address ของแต่ละสถานีจะต้องไม่ซ้ำกัน Address ของสถานีปลายทางและตัวต่อไปอาจจะระบุอุปกรณ์บางชิ้นที่ต่อกับสถานีนี้ เช่น Address ของเครื่องพิมพ์ที่ต้องการจะติดต่อด้วย ทั้งนี้ก็ขึ้นอยู่กับการกำหนดของผู้ใช้

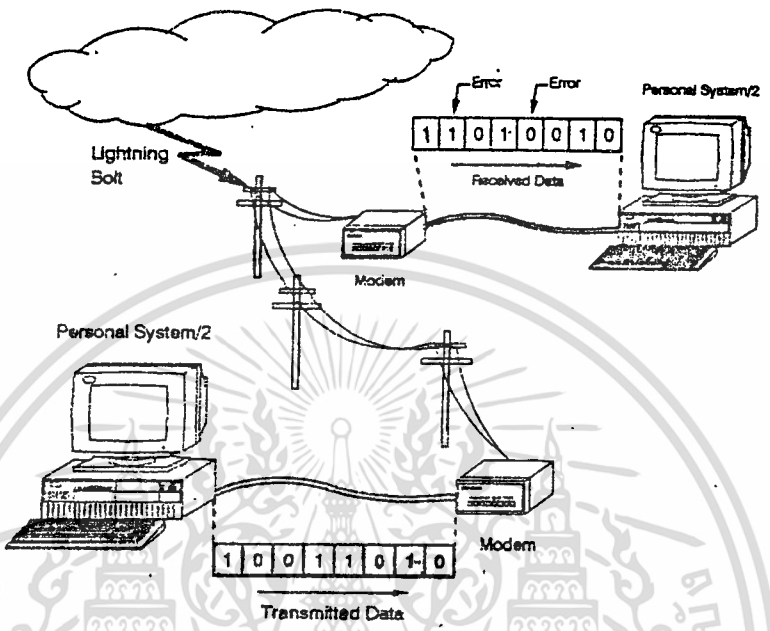
4) Data เป็นส่วนของข้อมูลที่เริ่มต้นด้วย STX ภายในส่วนนี้จะประกอบด้วยตัวข่าวสารที่ต้องการส่งไปยังสถานีรับ และปิดท้ายด้วย ETX ส่วนของข่าวสารอาจจะแยกออกเป็นกลุ่มย่อย ๆ ทั้งนี้เพื่อจะได้ควบคุมการผิดพลาดของข้อมูลได้ดียิ่งขึ้น โดยแต่ละกลุ่มจะเริ่มด้วย STX จบด้วย ETB จากนั้นจึงจะตามด้วย BCC หากมีข่าวสารกลุ่มอื่น ๆ อีกก็จะเริ่มด้วย STX และตามด้วยข่าวสารกลุ่มต่อ ๆ ไป เมื่อหมดข่าวสารก็จะจบด้วย ETX ดังแสดงในรูปที่ 16

SYN	SOH	HEADER	STX	DATA	ETB	BCC	STX	DATA	ETX	BCC

รูปที่ 16 แสดงการแยกส่วนของข่าวสารออกเป็นหลายๆส่วน

5) BCC (Block Check Charactor) เป็นส่วนที่ใช้สำหรับตรวจสอบการผิดพลาดของข้อมูลที่ได้รับเข้ามาเนื่องจากเกิดสายฟ้าแลบ Lightning bolt หรือ สัญญานรบกวน Noise ในสายส่ง ดังแสดงในรูปที่ 17 โดย BiSync Protocol นิยมใช้การตรวจสอบอยู่ 2 วิธีคือ

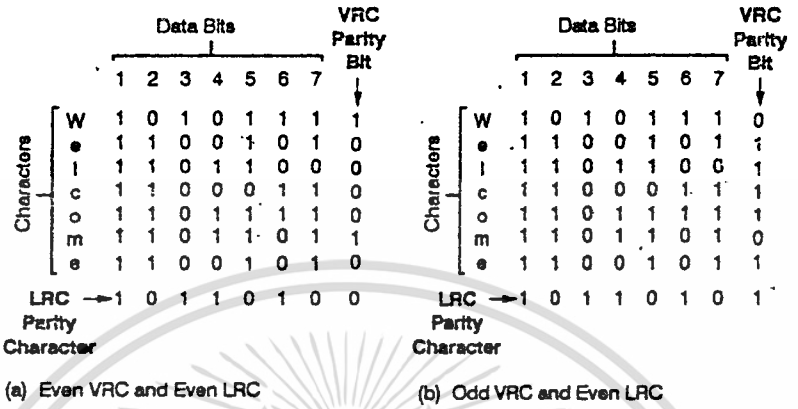
เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้



รูปที่ 17 Communications Errors induced by lighting

ก. VRC/LRC (Vertical and Longtitudinal Redundancy Checking) แยกออกเป็น 2 ประเภท โดย VRC เป็นการตรวจสอบในลักษณะที่เรียกว่า Charactor Parity ซึ่งจะเป็น Odd Parity ขนาด 1 บิท แล้วเพิ่มเข้ากับบิทสุดท้ายของรหัสของตัวอักษรนั้น ส่วน LRC เป็นการนำเอาบิท แต่ละบิทในแต่ละคอลัมน์ ของตัวอักษรมาหา Odd Parity ก็จะได้ผลลัพธ์เป็นตัวอักษร 1 ตัวการคำนวณจะเริ่มตั้งแต่ตัวอักษรแรกที่ส่งออกไป จนกระทั่งหมดข้อความที่จะส่งจากนั้นจึงนำค่าที่คำนวณได้ไปเป็นค่าใน BCC ส่วนการตรวจสอบของผู้รับก็ใช้วิธีการคำนวณแล้วนำมาเปรียบเทียบกันกับค่าที่ส่งมาให้ การตรวจสอบโดยวิธีนี้สามารถตรวจสอบความผิดพลาดได้เพียง 95-98 % ของการผิดพลาดทั้งหมดเท่านั้น และไม่สามารถตรวจสอบในกรณีที่มีการผิดพลาดถึง 2 บิท พร้อม ๆ กันในแถวเดียวกัน ดังแสดงในรูปที่ 18

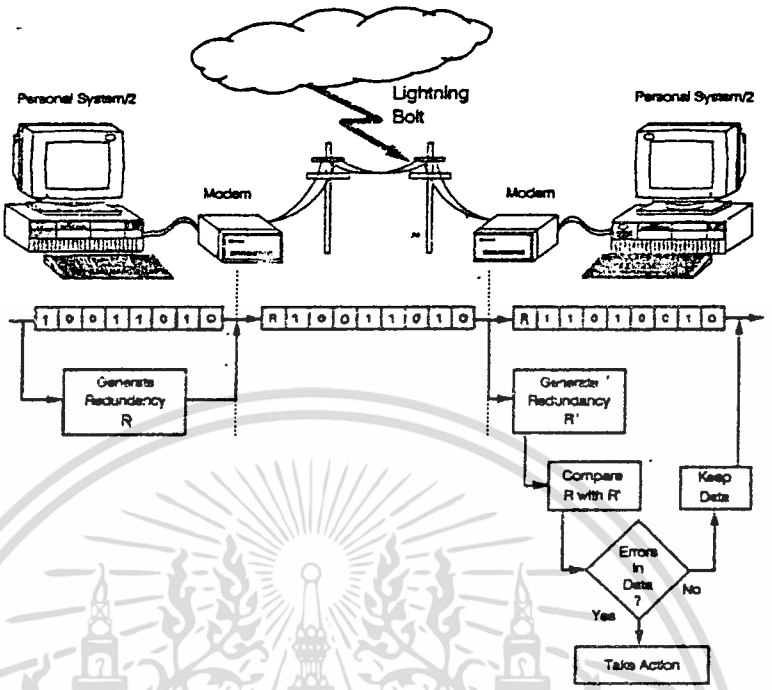
เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้



รูปที่ 18 Vertical and Longitudinal Redunancy Checking

ข. Cyclic Redundancy Check (CRC) เป็นวิธีการตรวจสอบหาข้อผิดพลาดที่ให้ประสิทธิภาพในการตรวจสอบถึง 99.95 % ของการผิดพลาดทั้งหมด โดยขนาดของ CRC จะมีขนาด 8 บิท วิธีการตรวจสอบนิยมใช้ตามมาตรฐานของ CCITT V.41 ซึ่งใช้ชื่อว่า CRC-16 ซึ่งเป็นวิธีการหาค่าเศษของการหารมาเป็นค่า BCC โดยที่ค่าตัวหารเป็นค่าคงที่ เขียนในรูปของโพลิโนเมียล คือ $X^{16+}X^{12+}X^{5+}1$ เมื่อถึงปลายทางผู้รับไม่จำเป็นที่จะต้องแยกค่า BCC ออกมาแล้วทำการเปรียบเทียบกับค่าที่คำนวณได้เพียงแต่นำค่าใน BCC ไปคำนวณรวมกับข้อมูลปกติ ถ้าได้ค่าเป็น 0 แสดงว่าข้อมูลไม่มีการผิดพลาด ดังแสดงในรูปที่ 19

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้



รูปที่ 19 Creating and checking redundancy error-detection data

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

หลักการของ RS-232C Monitor

เครื่อง RS-232c Monitor จะมีหน้าที่หลักอยู่ คือ การตรวจจับข้อมูลจาก DTE หรือ DCE เพื่อดูความผิดพลาดของสายส่งข้อมูลซึ่งสามารถแสดงเป็นบล็อกไดอะแกรมได้ดังรูปที่ 20

1. ภาค MPSC (Multiprotocol Serial Communications Controller) ก็คือ วงจรสำคัญที่ทำหน้าที่เป็นตัวกลางในการแลกเปลี่ยนข้อมูลระหว่างข้อมูลที่เป็นแบบขนานจากเครื่องคอมพิวเตอร์กับข้อมูลที่เป็นแบบอนุกรมจากระบบบัส RS-232C ซึ่งข้อมูลแบบอนุกรมที่วิ่งอยู่ในระบบบัส RS-232C นี้ยังมีลักษณะที่แตกต่างกันไปตามลักษณะของโปรโตคอลชนิดต่าง ๆ อีกด้วย

2. ภาค RS-232C Receiver ทำหน้าที่ปรับระดับของแรงดันไฟฟ้าของข้อมูลที่มาจากระบบบัส RS-232C ก่อนจะเข้ามายังภาค MPSC ก็จะต้องมีการลดระดับแรงดันทางไฟฟ้าลงเช่นเดียวกัน เพื่อให้อุปกรณ์ในวงจรสามารถทำงานได้อย่างถูกต้อง

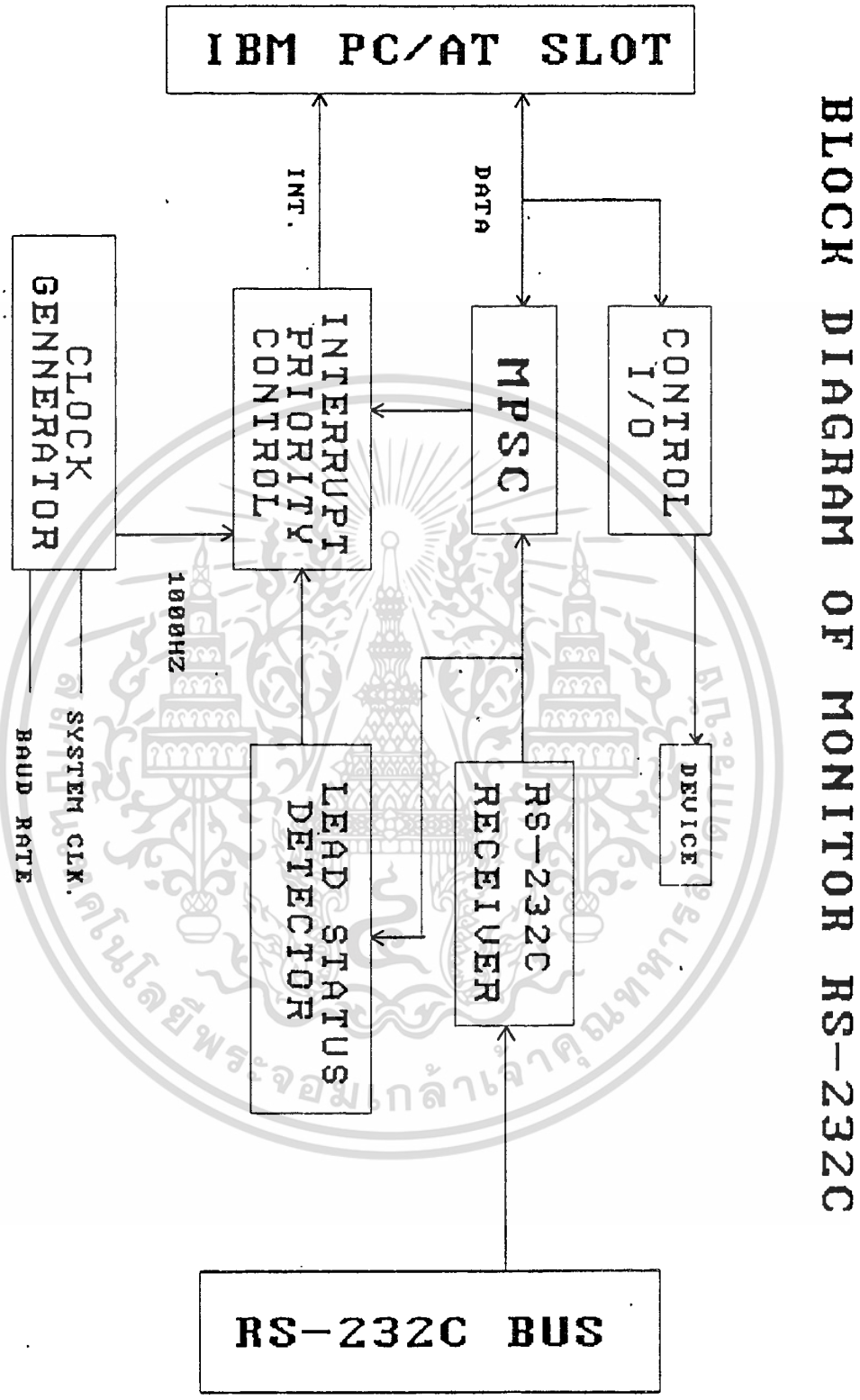
3. ภาค Modem Status Detecter สัญญาณจากระบบบัส RS-232C เฉพาะส่วนที่เป็นขาควบคุมทั้งหมด จะถูกลดระดับแรงดันไฟฟ้าลง จากนั้นจึงถูกป้อนเข้ามายังภาค Modem Status Detecter ทั้งนี้ก็เพื่อจะตรวจสอบการเปลี่ยนแปลงของขาควบคุมต่าง ๆ ในระบบบัส RS-232C แล้วรายงานผลการเปลี่ยนแปลงให้กับเครื่องคอมพิวเตอร์ทุกครั้ง

4. ภาค Interrupt Priority Controller ทำหน้าที่ตรวจสอบการขอขัดจังหวะการทำงานของเครื่องคอมพิวเตอร์ ในกรณีที่มีการขอขัดจังหวะพร้อม ๆ กัน จะยินยอมให้สัญญาณที่มีลำดับความสำคัญสูงกว่า สามารถขอขัดจังหวะการทำงานของ เครื่องคอมพิวเตอร์ได้

5. ภาค Clock Generator จะทำหน้าที่เป็นตัวกำเนิดสัญญาณนาฬิกาให้กับระบบ และสร้างสัญญาณนาฬิกาเพื่อใช้เป็นตัวกำหนดอัตราความเร็วของการรับส่งข้อมูล

6. การตรวจจับข้อมูล (Monitoring) ในกรณีนี้เครื่อง RS-232C Monitor จะนำเอาข้อมูลที่วิ่งอยู่บนระบบบัส RS-232C มาแสดงผลบนจอภาพ โดยจะไม่มีการโต้ตอบกับระบบบัส RS-232C แต่อย่างใด จึงไม่มีการรบกวนในระบบบัส

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้



รูปที่ 21 แสดง Block Diagram RS-232C MONITOR

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

บทที่ 3
การออกแบบและพัฒนา

รายละเอียดของระบบ

1. ตรวจจับข้อมูลที่วิ่งอยู่ในระบบบัสของ RS-232C ที่มีความเร็วในการรับส่งไม่เกิน 19200 บิทต่อวินาที

2. นำข้อมูลที่ตรวจจับได้เก็บลงหน่วยความจำหรือแผ่นจานแม่เหล็ก

3. ข้อมูลที่เก็บไว้สามารถนำมาแสดงใหม่ได้อีก

การวิเคราะห์ระบบ

เพื่อให้ง่ายต่อการใช้งาน จึงใช้เครื่องไมโครคอมพิวเตอร์ที่ใช้หน่วยประมวลผลกลาง (Central Processing Unit) 80386 ที่มีความถี่ของสัญญาณนาฬิกามากกว่า 20 MHz เป็นตัวควบคุมการทำงานหลักและจะต้องมีส่วนเพิ่มเติม คือ

1. วงจรรับส่งข้อมูลกับระบบบัส RS-232C เป็นส่วนของฮาร์ดแวร์ (Hardware) ติดตั้งอยู่บนการ์ด (Extension Card) ที่สามารถเสียบบนสล็อต (Slot) ของเครื่องไมโครคอมพิวเตอร์ได้ การติดต่อกับโปรแกรมจะใช้การขัดจังหวะการทำงาน (Interupt) เข้าไปที่ขา IRQ3 หรือ IRQ5 ของไอซี Interupt Priority Controller (8259) ที่อยู่ภายในเครื่องไมโครคอมพิวเตอร์ส่วนการสร้างสัญญาณขัดจังหวะการทำงานทุกๆ 1 มิลลิวินาที เข้าที่ขาเดียวกัน

2. โปรแกรม Monitor เป็นส่วนที่จะนำเอาข้อมูลที่ได้รับมาจากการขัดจังหวะการทำงานของไอซี MPSC ไปเก็บลงหน่วยความจำหรือแผ่นจานแม่เหล็ก และนำไปแสดงผลบนจอภาพการแสดงผลบนจอภาพจะต้องใช้ตัวอักษรในการแสดงผลที่ไม่เป็นไปตามมาตรฐานของการแสดงผลทั่ว ๆ ไป ดังนั้นจึงต้องมีการออกแบบตัวอักษรใหม่ ซึ่งการแสดงผลที่สามารถแก้ไขรูปแบบของตัวอักษรใน TEXT MODE ได้ก็

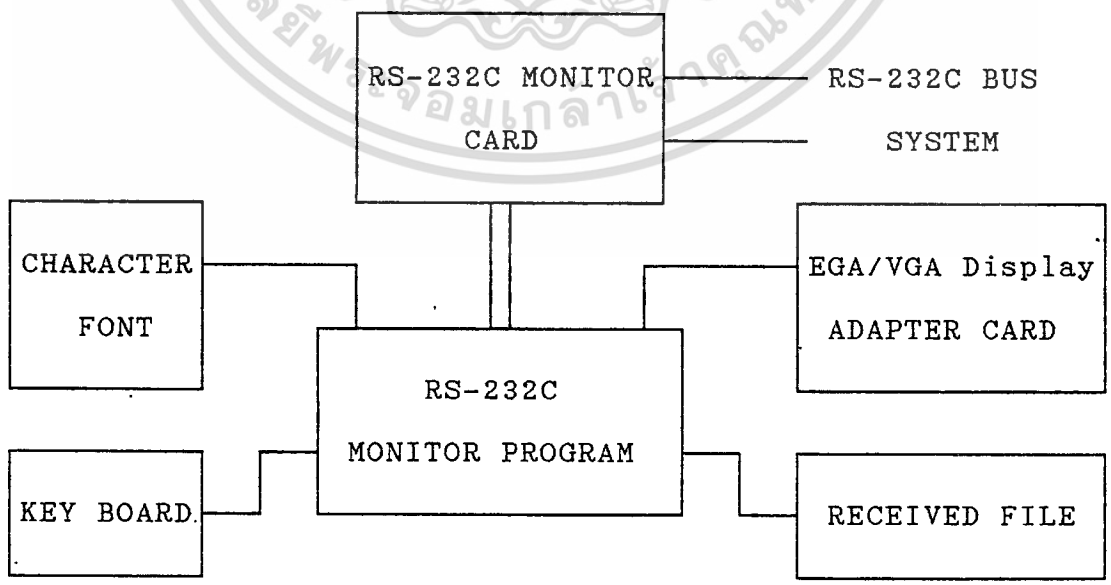
เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

คือ VGA (Video Graphics Arrays) หรือ EGA (Enhance Graphics Adapter) เท่านั้น ส่วนการแสดงในแบบกราฟฟิค (Graphics) นั้นช้าเกินกว่าที่จะแสดงข้อมูลที่รับเข้ามาด้วยความเร็วสูงมาก ๆ ได้

ภาษาที่ใช้เขียนโปรแกรมได้เลือกภาษาซี (C language)ของบริษัทบอร์แลนด์ จำกัด Borlandc++ Version 3.0 . เพราะสามารถเขียนได้กระชับและมีความรวดเร็วในการทำงานสูงสุด อีกทั้งยังง่ายต่อการพัฒนาโปรแกรมต่อไปในอนาคต และมี Utility ให้ใช้มากมาย

การติดต่อกับผู้ใช้งาน

การเลือกการใช้งานฟังก์ชั่น (Function) ต่าง ๆ ของโปรแกรมจะใช้ในระบบเมนูแบบ Pull Down และการทำงานของฟังก์ชั่นต่าง ๆ รวมถึงการจะใช้การแสดงผลที่เรียกว่า แบบหน้าต่าง (Window) ซึ่งจะปรากฏออกมาซ้อนทับกับฟังก์ชั่นที่เรียกใช้อยู่ก่อนหน้านี้ เมื่อจบการทำงานของฟังก์ชั่นใด ๆ หน้าต่างของฟังก์ชั่นนั้นจะหายไปเหลือแต่หน้าต่างของฟังก์ชั่นเดิมให้ใช้งานได้ตามปกติ



รูปที่ 22 โครงสร้างของ RS-232C Monitor

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

บทที่ 4

การออกแบบและโครงสร้างทางฮาร์ดแวร์

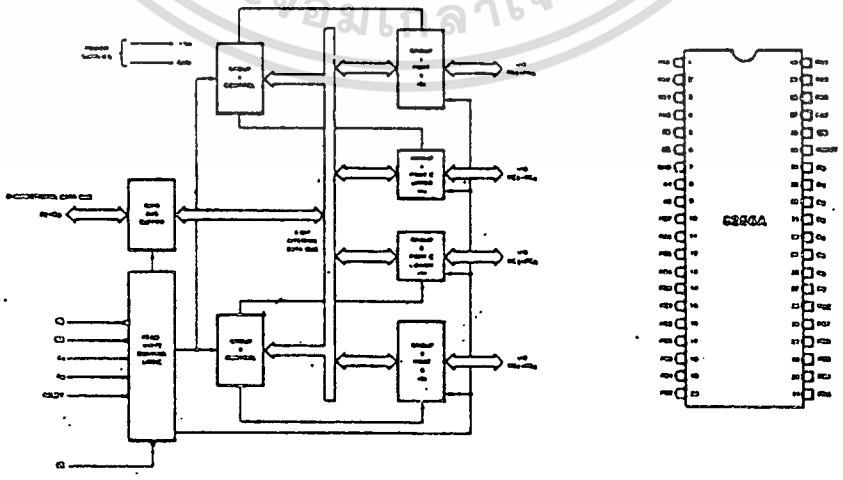
รายละเอียดเกี่ยวกับ 8255.

8255 เป็นอุปกรณ์ LSI (LARGE SCALE INTEGRATED CIRCUIT) บรรจุอยู่ใน PACKAGE 40 ขาแบบ DIP (DUAL-IN-LINE PACKAGE) เริ่มผลิตโดยบริษัท INTEL COPERATION ผู้ผลิตไมโครโปรเซสเซอร์เบอร์ 8080 จุดประสงค์เพื่อใช้งานร่วมกับ 8080 โดยเฉพาะแต่ในภายหลังได้มีการนำ 8255 ไปประยุกต์ใช้งานร่วมกับไมโครโปรเซสเซอร์เบอร์อื่น ๆ หากผู้อ่านเข้าใจการใช้งาน 8255 กับ CPU ที่จะกล่าวถึงในที่นี้แล้วก็จะนำไปประยุกต์ใช้งานในลักษณะอื่น ๆ ได้ไม่ยากนัก

8255A/8255A-5
PROGRAMMABLE PERIPHERAL INTERFACE

- MCS-85™ Compatible 8255A-5
- 24 Programmable I/O Pins
- Completely TTL Compatible
- Fully Compatible with Intel® Microprocessor Families
- Improved Timing Characteristics
- Direct Bit Set/Reset Capability Easing Control Application Interface
- 40-Pin Dual In-Line Package
- Reduced System Package Count
- Improved DC Driving Capability

The Intel® 8255A is a general purpose programmable I/O device designed for use with Intel® microprocessors. It has 24 I/O pins which may be individually programmed in 2 groups of 12 and used in 3 major modes of operation. In the first mode (MODE 0), each group of 12 I/O pins may be programmed in sets of 4 to be input or output. In MODE 1, the second mode, each group may be programmed to have 8 lines of input or output. Of the remaining 4 pins, 3 are used for handshaking and interrupt control signals. The third mode of operation (MODE 2) is a bidirectional bus mode which uses 8 lines for a bidirectional bus, and 5 lines, borrowing one from the other group, for handshaking.



รูปที่ 23 แสดง Block diagram ของ 8255

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

หน้าที่ของแต่ละ Block มีดังต่อไปนี้คือ

Block กลุ่มแรกที่เราจะพูดถึงนี้ ได้แก่ Block จำนวน 4 Block ที่อยู่ทางด้านขวาของรูป ซึ่งจะเป็นส่วนที่เชื่อมต่อกับอุปกรณ์ภายนอกอื่น ๆ โดยมีสาย PA0-PA7 PB0-PB7 และ PC0-PC7 . เป็นทางผ่านของข้อมูลระหว่างอุปกรณ์ภายนอกกับ 8255 สายสัญญาณเหล่านี้จะถูกแบ่งออกเป็น 3 Input/Output Port ได้แก่ Port A (PA), Port B (PB) และ Port C (PC) Port เหล่านี้แต่ละ Port สามารถเป็นได้ทั้ง Port input และPort output และแต่ละ Block จะมีสายสัญญาณเชื่อมเข้ากับ BUS ข้อมูลภายในของ 8255.

Block กลุ่มถัดไปได้แก่ GROUP A CONTROL และ GROUP B CONTROL ซึ่งจะเป็นตัวกำหนดลักษณะการทำงานของทั้ง 3 Input/Output Port (8255 มีลักษณะการทำงานที่แตกต่างกันอยู่ 3 โหมด สามารถกำหนดได้โดยการโปรแกรมส่ง CONROL WORD ให้กับ 8255. ซึ่งกล่าวถึงในภายหลัง) จากรูปจะเห็นว่า Port C นี้จะประกอบด้วย Port ขนาด 4 Bit 2 Port กลุ่มหนึ่งจะถูกควบคุมโดย GROUP A CONTROL และอีกกลุ่มหนึ่งจะถูกควบคุมโดย GROUP B CONTROL สำหรับเหตุผลนั้นจะกล่าวถึงในภายหลัง

Block กลุ่มสุดท้ายที่จะกล่าวถึงได้แก่ DATA BUS BUFFER และ READ/WRITE CONTROL LOGIC ซึ่ง Block เหล่านี้จะเป็นส่วนที่ติดต่อกับ CPU, DATA BUS BUFFER นี้จะเป็น Buffer ให้กับ Bus ข้อมูลของ CPU ส่วน READ /WRITE CONTROL LOGIC จะเป็นส่วนที่ควบคุมให้ข้อมูลเข้าหรือออกจากรีจิสเตอร์ภายใน ตัวที่ถูกต้อง และในเวลาที่เหมาะสม

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

รายละเอียดการจัดเรียงขาของ 8255

ในส่วนนี้เราจะพิจารณาหน้าที่ของขาแต่ละขาของ 8255 ซึ่งข้อมูลเหล่านี้จะมีประโยชน์ในการเชื่อมต่อเข้ากับระบบบัสของ CPU สำหรับการจัดขาแสดงไว้ดังรูป รายละเอียดของแต่ละขามีดังนี้คือ

D0-D7 (DATA BUS) : เป็นสายข้อมูล Input/Output แบบสองทิศทาง (BIDIRECTIONAL BUS) จะเป็นทางผ่านของข้อมูลระหว่าง Port ต่าง ๆ ของ 8255 กับ Bus ข้อมูลของ CPU

CS (CHIP SELECT INPUT) : เมื่อขานี้มีสถานะลอจิกเป็น "0" CPU จะสามารถที่จะอ่านหรือเขียนข้อมูลกับ 8255 ได้

RD (READ INPUT) : เมื่อขานี้มีสถานะลอจิกเป็น "0" และสัญญาณ CS มีลอจิกเป็น "0" ข้อมูลจาก 8255 จะปรากฏสู่ระบบ Bus ข้อมูล CPU ก็จะสามารถอ่านข้อมูลออกไปได้ (ในการตั้งชื่อของขาสัญญาณนี้จะถือเอา CPU เป็นหลัก)

WR (WRITE INPUT) : เมื่อขานี้มีสถานะลอจิกเป็น "0" และขา CS สัญญาณมีลอจิกเป็น "0" ข้อมูลจากระบบ Bus ข้อมูลจะถูกเขียนเข้าไปยัง 8255 ได้

A0-A1 (ADDRESS INPUT) : จะเป็นตัวกำหนดการเลือกใช้รีจิสเตอร์ภายในของ 8255 ซึ่งจะกล่าวรายละเอียดในภายหลัง

RESET : เมื่อขานี้มีสถานะเป็น "1" 8255 จะอยู่ในสภาวะรีเซ็ท ทุก ๆ Port ของ 8255 จะถูก Set ให้อยู่ในโหมด Input

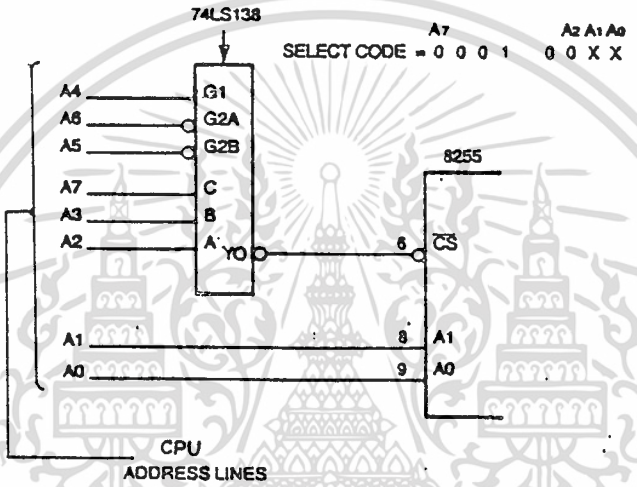
PA0-PA7,PB0-PB7 : ขาสัญญาณเหล่านี้จะถูกใช้เป็น Port I/O ขนาด 8 Bit ใช้ต่อเข้ากับอุปกรณ์ภายนอกอื่น ๆ

PC0-PC7 : ขาสัญญาณนี้ถูกใช้เป็น Port I/O ขนาด 8 Bit เช่นเดียวกับ PA0-PA7 และ PB0-PB7 แต่กลุ่มของขาสัญญาณเหล่านี้สามารถแบ่งออกเป็น 2 กลุ่ม โดยแต่ละกลุ่มมีขนาด 4 Bit ได้ กลุ่มแรกจะใช้ควบคุม PB0-PB7 และกลุ่มที่ 2 ใช้ควบคุม PA0-PA7 (ซึ่งจะกล่าวถึงรายละเอียดในภายหลัง)

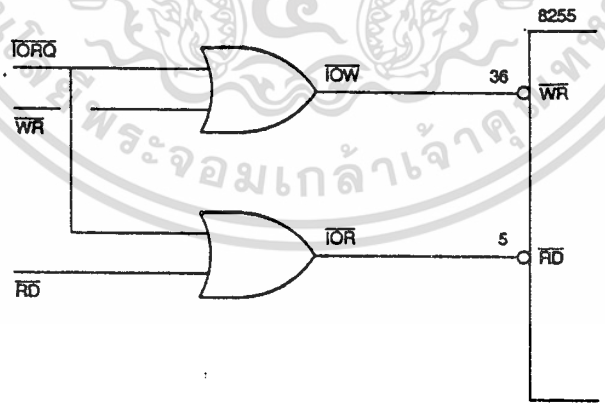
เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

การต่อ 8255 เข้ากับ CPU

ในการต่อ 8255 เข้ากับระบบของ CPU นั้น สัญญาณต่าง ๆ ที่เกิดขึ้นจะเหมือนกับขบวนการติดต่อกับ INPUT/OUTPUT โดยจะต้องเอาสัญญาณ AO-A7 จาก CPU มาถอดรหัสเพื่อสร้างสัญญาณเลือก Port แต่เนื่องจาก 8255 มีขา ADDRESS INPUT อยู่แล้ว 2 ขา(AO,A1) ซึ่งโดยปกติแล้วขา AO,A1 นี้จะต่อเข้าโดยตรงกับ AO,A1 จาก BUS ADDRESS นั่นคือ 8255 หนึ่งตัวจะใช้ค่า Port



รูปที่ 24.1 แสดงผังวงจรการถอดรหัสการเลือก Port ที่ติดต่อกับ 8255



รูปที่ 24.2 แสดงการต่อขา READ/WEITE เข้ากับระบบของ CPU

ADDRESS ถึง 4 ค่า(2^2) ส่วนสัญญาณอีก 6 เส้น (A2-A7) จะนำไปถอดรหัสเพื่อทำสัญญาณเลือกชิพ (CHIP SELECT) ให้แก่ 8255

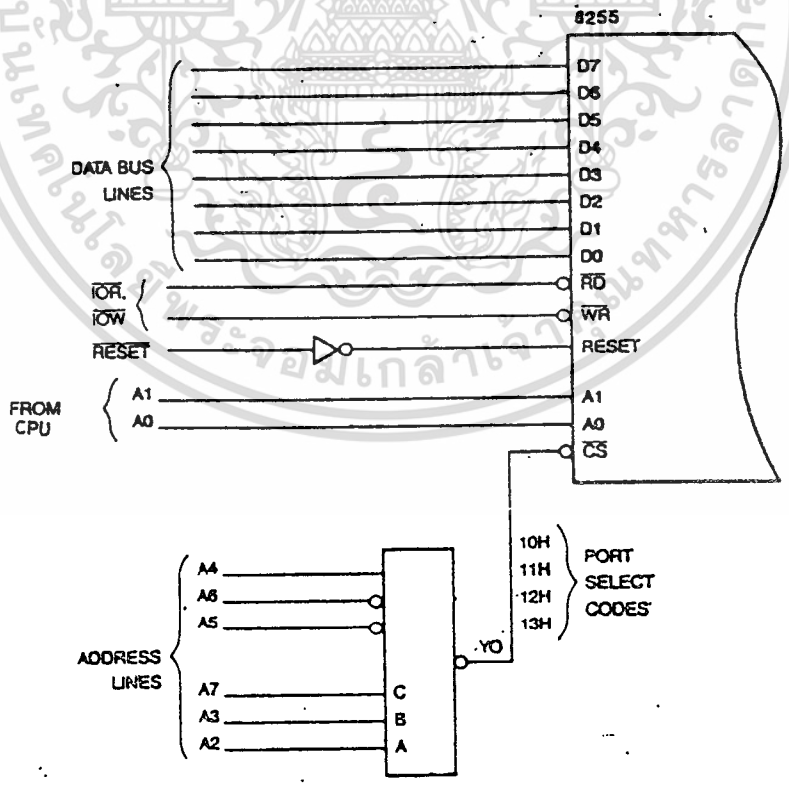
ในที่นี้จะสมมุติให้ 8255 มี Port ADRESS อยู่ที่ 10H, 11H,12H และ13H ซึ่งวิธีหนึ่งที่สามารถจะถอดรหัส Port เหล่านี้ได้แสดงไว้ดังรูป 24.1

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

การต่อขา RD และ WR ของ 8255 เข้ากับสัญญาณควบคุม IOR และ IOW ของระบบ เราจะไม่ต่อขา RD และ WR เข้ากับขา RD และ WR ของ CPU โดยตรง เพราะในตัวอย่างวิธีการถอดรหัสของเรานี้ อาจจะเกิดกรณีที่ A7-A0 มีค่าตรงกับ 000100XXB ซึ่งจะทำให้เกิดการอ่านหรือเขียนข้อมูลกับ 8255 ที่ไม่ต้องการได้ ในการแก้ปัญหานี้เราจึงใช้สัญญาณ IORQ จาก CPU มาทำให้เป็นสัญญาณ IOR และ IOW เพื่อแยกว่าเป็นการติดต่อกับ INPUT/OUTPUT ไม่ใช่หน่วยความจำ ดังแสดงในรูปที่ 24.2

8255 READ และ WRITE REGISTER.

ขณะนี้เราได้ทำการต่อ 8255 เข้ากับระบบของ CPU แล้ว ต่อไปเราจะศึกษาการโปรแกรมใช้งาน 8255 เพื่อที่จะให้ทำงานตามที่เราต้องการได้ จะเริ่มต้นพิจารณาที่รีจิสเตอร์ภายใน 4 ตัวของ 8255 สำหรับใช้ในตัวอย่างการถอดรหัสของเรานี้ สมมุติตำแหน่งของรีจิสเตอร์จะอยู่ที่ ADDRESS 10H, 11H, 12H และ 13H ซึ่งรายละเอียดของรีจิสเตอร์เหล่านี้มีดังนี้คือ

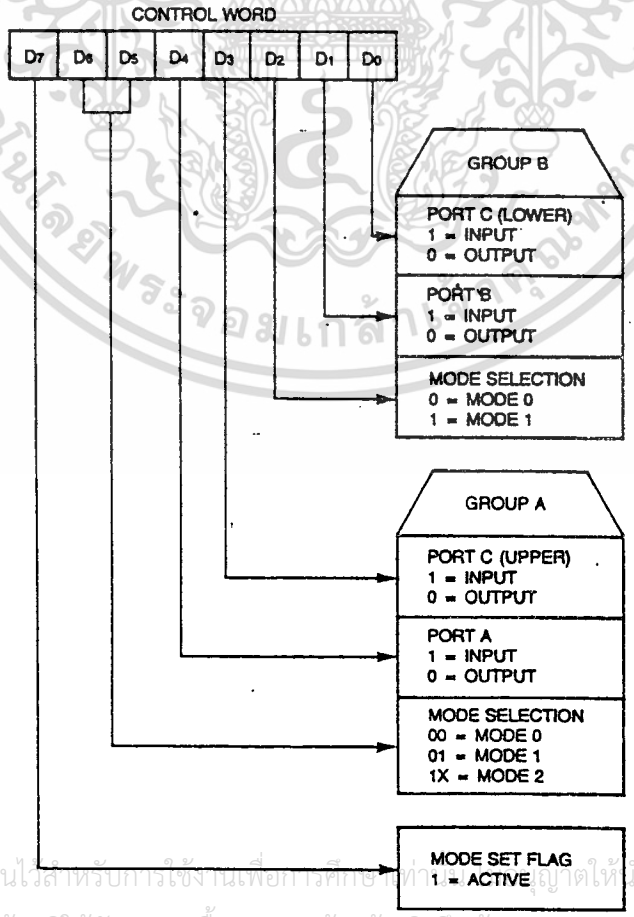


รูปที่ 25 แสดงผังวงจรสมบูรณ์ของการเชื่อมต่อ 8255 เข้ากับระบบของ CPU

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

DEVICE PIN				REGISTER NAME
RD	WR	A1	AO	
1	O	O	O	WRITE PORT A DATA
O	1	O	O	READ PORT A DATA
1	O	O	1	WRITE PORT B DATA
O	1	O	1	READ PORT B DATA
1	O	1	O	WRITE PORT C DATA
O	1	1	O	READ PORT C DATA
1	O	1	1	WRITE CONTROL DATA
O	1	1	1	ILLEGAL READ REGISTER

หน้าที่ของรีจิสเตอร์หมายเลข O-2 จะถูกกำหนดลักษณะการทำงานจากรีจิสเตอร์หมายเลข 3 (รีจิสเตอร์ควบคุม) ในรูปข้างล่างจะแสดงรายละเอียดของแต่ละ Bit ของรีจิสเตอร์ควบคุมนี้ ต่อไปเราจะกล่าวถึงลักษณะการทำงานของ 8255 ทั้ง 3 โหมด และการโปรแกรมให้อยู่ในโหมดต่าง ๆ ได้ดังต่อไปนี้คือ



เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

โหมด 0: BASIC REGISTER I/O.

ในการ Set 8255 ให้อยู่ในโหมด 0 นั้นเราจะต้องส่งคำสั่งควบคุม (CONTROL WORD) ให้แก่รีจิสเตอร์ควบคุมก่อน คำสั่งควบคุมนี้จะกำหนดลักษณะการทำงานให้แก่แต่ละ Port ของ 8255 ตัวอย่างหนึ่งของคำสั่งควบคุมที่สั่งให้ 8255 ทำงานอยู่ในโหมด 0 นี้ได้แก่

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

จากรูปรายละเอียดของแต่ละ Bit ของรีจิสเตอร์ควบคุมนี้เราจะสามารถรู้ได้ว่า

Bit D7 เป็นตัวกำหนดว่าเป็นคำสั่งควบคุม (CONTROL WORO)

Bit D6 และ D5 กำหนดโหมดการทำงานของ Port A D6, D5 เป็น "0" แสดงว่าอยู่ในโหมด 0

Bit D3 = "0" Set Port C 4 Bitบนเป็น Port Output

Bit D4 = "0" กำหนดให้ Port A เป็น Port Output

Bit D2 = "0" Setโหมดของ Port B ให้ Port B อยู่ในโหมด 0

Bit D1 = "0" Set Port B เป็น Port Output

Bit D0 = "0" Set Port C ให้ 4 Bit ล่างเป็น Port Output

คำสั่งควบคุมนี้จะกำหนดให้ Port ทั้ง 3 ของ 8255 ทำงานอยู่ในโหมด 0 และเป็น Port Output ซึ่งจะได้สายสัญญาณซึ่งสามารถติดต่อกับอุปกรณ์ภายนอกได้ถึง 24 สาย

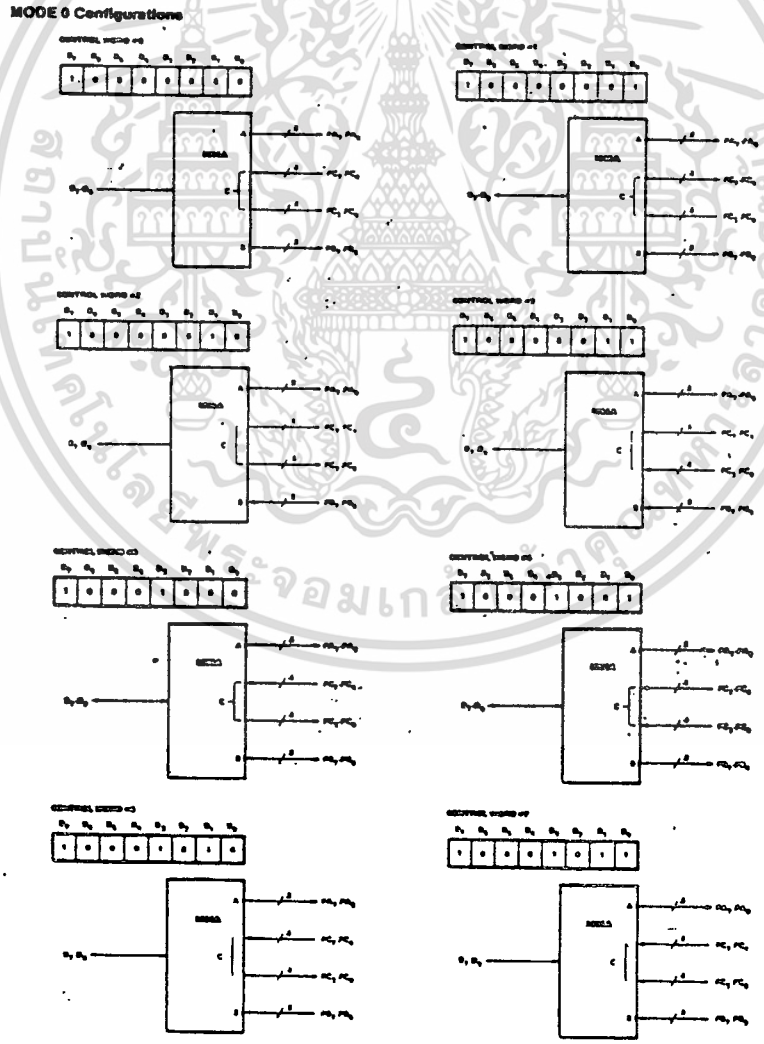
เมื่อ CPU ทำการ Set ให้ 8255 แล้ว คำสั่งข้างต้นแล้ว 8255 จะถูก Setให้ Port ทุก Port เป็น Port Output และอยู่ในโหมด 0 เราจะสามารถส่งข้อมูลออกไปยัง Port ต่าง ๆ ได้ด้วยคำสั่ง Out Port ของ CPU

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

ตัวนั้น ๆ

ในการทำงานในโหมด 0 ของ 8255 นี้อาจจะสั่งให้ Portของ 8255 เป็น Input หรือOutput ก็ได้อย่างเช่น ให้ Port A และ Port B เป็น Port Output และ Port C เป็น Port Input เราจะต้องส่งคำสั่งควบคุมใหแก่รีจิส-เตอร์ควบคุมในลักษณะดังนี้คือ

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

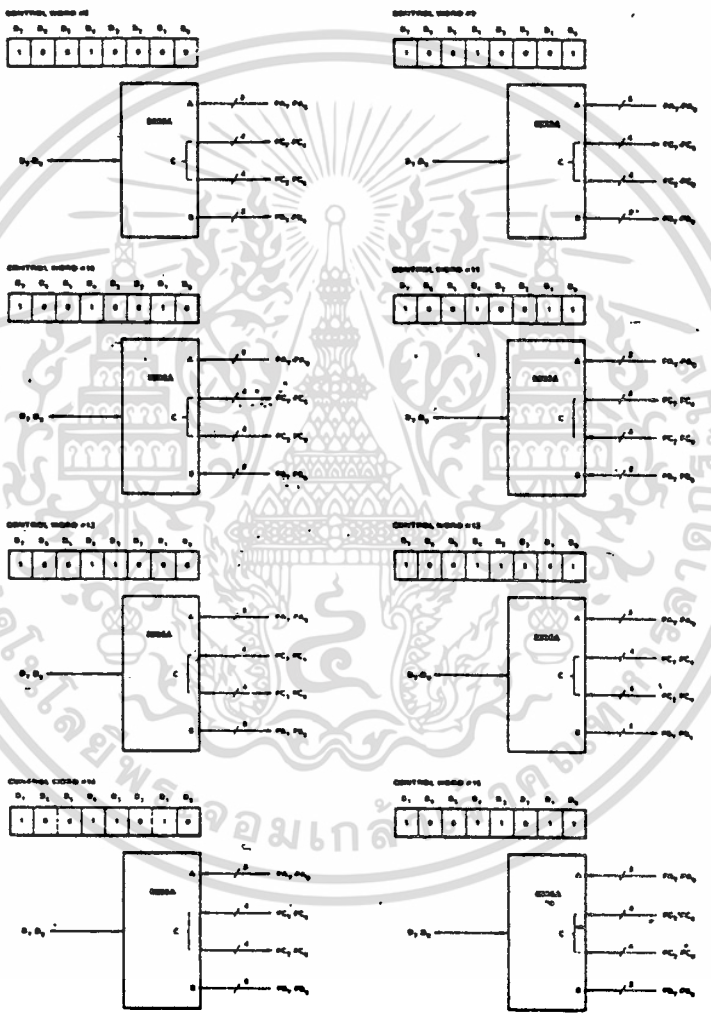


รูปที่ 26 แสดงลักษณะต่าง ๆ ในการใช้งานใน Mode 0

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

หลังจากที่ส่งคำสั่งควบคุมให้แก่รีจิสเตอร์ควบคุมแล้ว 8255 จะถูก Set ให้มีลักษณะการทำงานดังรูป เราจะสามารถใช้คำสั่งอ่านข้อมูลจาก Port B ได้

ลักษณะการทำงานของ Port ต่าง ๆ ที่สามารถกำหนดได้ในโหมด 0 แสดงไว้ในรูปที่ 26



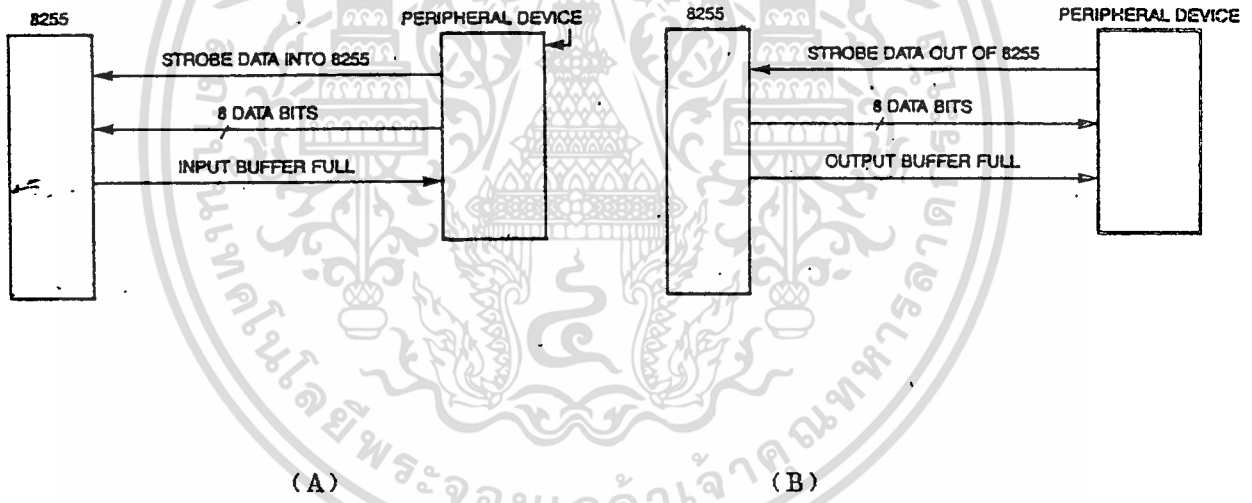
รูปที่ 26 (ต่อ) แสดงลักษณะต่าง ๆ ในการใช้งานใน Mode 0

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

การใช้งาน 8255 ในโหมด 1

การทำงานของ 8255 ในโหมด 1 นี้เป็นการทำงานในลักษณะของการ HANDSHAKE, Port A และ Port B จะเป็น Port ข้อมูล ส่วน Port C นี้จะถูกใช้เป็นสัญญาณ HANDSHAKE โดย 4 Bitบนจะเป็นสัญญาณ HANDSHAKE ให้กับ Port A และ 4 Bitล่างจะเป็นสัญญาณ HANDSHAKE ให้กับ Port B

หลักการรับส่งข้อมูลในวิธีการของ HANDSHAKE นี้ คือการให้อุปกรณ์ภายนอกส่งสัญญาณแสดงสภาวะความพร้อมให้กับ 8255 ดังแสดงในรูปที่ 27



รูปแสดง 27 Block Diagram แสดงลักษณะการทำงานของการติดต่อระหว่าง 8255กับอุปกรณ์ภายนอกในลักษณะ HANDSHAKE

ในรูปที่ 27 (A) นี้ข้อมูลจะถูกส่งออกจากอุปกรณ์ภายนอกเข้าสู่ 8255 ก่อนที่อุปกรณ์ภายนอกจะเขียนข้อมูลให้แก่ 8255 จะต้องมีการตรวจสอบ INPUT BUFFER FULL FLAG เสียก่อน ถ้า FLAG นี้เป็นจริงแสดงว่าข้อมูลใน Buffer 8255 นี้ยังไม่ถูกอ่านโดย CPU คือข้อมูลจากอุปกรณ์ภายนอกส่งข้อมูลให้กับ 8255 แล้ว CPU ยังไม่ได้อ่านเอาข้อมูลเข้าไป แต่ถ้า FLAG นี้เป็นเท็จแสดงว่า CPU อ่านข้อมูลออก

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

ไปแล้ว อุปกรณ์ภายนอกก็จะเขียนข้อมูลใหม่ให้ 8255 ได้ และเมื่ออุปกรณ์ภายนอกเขียนข้อมูลให้ 8255 แล้ว INPUT BUFFER FULL FLAG ก็จะกลับมาเป็นจริงอีกครั้งหนึ่ง

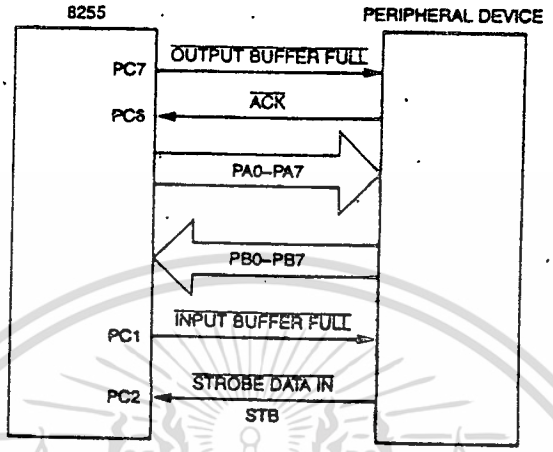
ในรูปที่ 27 (B) 8255 จะทำหน้าที่เป็นตัวส่งข้อมูลให้กับอุปกรณ์ภายนอกก่อนที่ 8255 จะส่งข้อมูลให้กับอุปกรณ์ภายนอกนั้นจะต้อง Set OUTPUT BUFFER FULL FLAG เสียก่อนเพื่อบ่งบอกให้อุปกรณ์ภายนอกทราบว่าขณะนี้ 8255 มีข้อมูลพร้อมที่จะส่งออกไปให้แล้ว เมื่ออุปกรณ์ภายนอกส่งสัญญาณ STROBE รับเอาข้อมูลเข้าไปแล้ว INPUT BUFFER FULL FLAG จะเปลี่ยนเป็นเท็จ เพื่อบ่งบอกให้อุปกรณ์ภายนอกรู้ว่าขณะนี้ไม่มีข้อมูลอยู่ใน 8255, CPU สามารถส่งข้อมูลใหม่ออกไปให้ 8255 ได้

วิธีการทำ HANDSHAKE นี้ มีประโยชน์มากในกรณีที่อุปกรณ์ภายนอกทำงานช้ากว่าระบบไมโครโปรเซสเซอร์ ด้วยวิธีการนี้ไมโครโปรเซสเซอร์สามารถที่จะส่งข้อมูลให้กับ 8255 แล้วไปทำงานอื่นได้ จนกว่าข้อมูลภายใน 8255 ถูกส่งออกไปแล้ว ไมโครโปรเซสเซอร์จึงจะส่งข้อมูลใหม่ออกไปให้ ต่อไปนี้เราจะพิจารณารายละเอียดของการทำ HANDSHAKE ของ 8255 เพิ่มขึ้น

Block Diagram ที่แสดงในรูปที่ 28 กำหนดให้ Port A เป็น Port Output และ Port B เป็น Port Input (ซึ่งอาจจะอยู่ในลักษณะอื่น ๆ ก็ได้) เพื่อให้ 8255 ทำงานในลักษณะดังกล่าวจะต้องส่งคำสั่งควบคุมให้แก่ 8255 ดังนี้ 10100110B หรือ 00A3H

ในรูปนี้ข้อมูล Output จะอยู่บนขา PA0-PA7 ของ 8255, OUTPUT BUFFER FULL อยู่ที่ PC7, สัญญาณ ACKNOWLEDGE จากอุปกรณ์ภายนอกจะอยู่ที่ PC6 ,ข้อมูล Input อยู่ที่ PB0-PB7, INPUT BUFFER FULL อยู่ที่ PC1 และสัญญาณ STROBE เพื่อให้ 8255 รับเอาข้อมูลเข้าไปอยู่ที่ PC2 รูปที่ 29 นี้แสดงรายละเอียดของแต่ละ Bit ของ Port C ในขณะที่ 8255 ทำงานในโหมด 1

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้



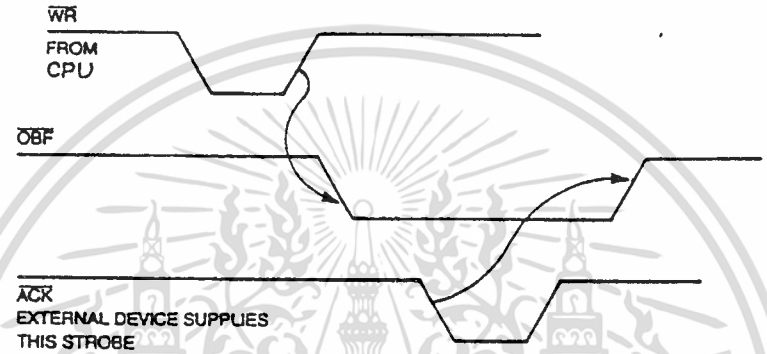
รูปที่ 28 แสดง Block Diagram ของ 8255 ที่ให้ Port A เป็น Port Output และPort Input ในลักษณะ HANDSHAKE

	OUT	IN
PC0	$INTR_B$	$INTR_B$
PC1	IBF_B	$\overline{OBF}_B$
PC2	$\overline{STB}_B$	$\overline{ACK}_B$
PC3	$INTR_A$	$INTR_B$
PC4	$\overline{STB}_A$	I/O
PC5	IBF_A	I/O
PC6	I/O	$\overline{ACK}_A$
PC7	I/O	$\overline{OBF}_A$

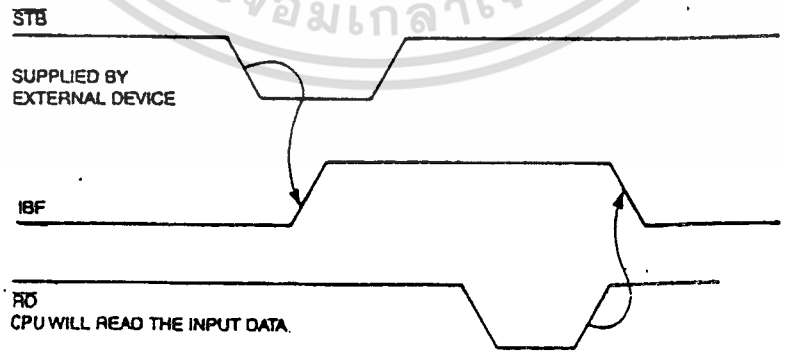
รูปที่ 29 แสดงรายละเอียดและหน้าที่ของแต่ละขาของ Port C ในโหมด 1

เอกสารนี้ถูกจัดทำขึ้นเพื่อการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ในเชิงพาณิชย์ การค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

ต่อไปเราจะพิจารณาซอฟท์แวร์ที่ทำให้ 8255 ทำงานในลักษณะดังกล่าวในวิธีการของเรานี้จะไม่ใช้การ Interrupt แต่จะให้ 8255 มาคอยตรวจสอบสถานะของ Port C แทน รูปที่ 30 (A) และรูปที่ 30 (B) นี้แสดง Digram เวลาที่เกิดขึ้นขณะการรับส่งในขบวนการ HANDSHAKE



รูปที่ 30 (A) แสดง Diagram เวลาของการส่งผ่านข้อมูลจาก Port A ไปยังอุปกรณ์ภายนอก

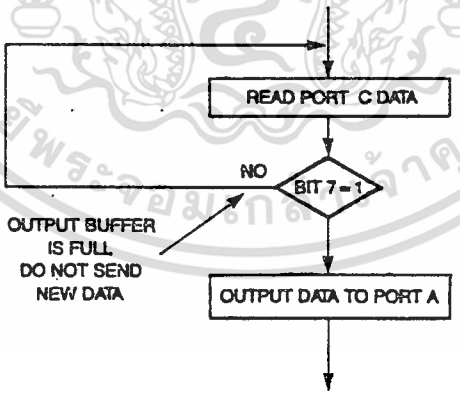


รูปที่ 30 (B) แสดง Diagram เวลาของการส่งผ่านข้อมูลจากอุปกรณ์ภายนอกเข้าสู่ CPU

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

ในการส่งข้อมูลออกไปยังอุปกรณ์ภายนอก 8255 จะต้องตรวจสอบ OUTPUT BUFFER FULL (PC7) ว่ามีค่าลอจิกเป็น "1" หรือไม่ ซึ่งทำได้โดยการอ่านข้อมูลเข้ามาทาง Port C ถ้า Bit D7 ของ Port C มีลอจิกเป็น "1" ก็จะแสดงว่าอุปกรณ์ภายนอกได้รับเอาข้อมูลจาก 8255 ไปแล้ว และ CPU จะสามารถส่งข้อมูลใหม่ให้ 8255 ได้ แต่ถ้า Bit D7 ของ Port C มีลอจิกเป็น "0" จะแสดงว่าข้อมูลที่อยู่ใน 8255 ยังไม่ถูกอุปกรณ์ภายนอกอ่านออกไป ,CPU จะไม่ส่งข้อมูลใหม่ออกไป จนกว่า Bit D7 ของ Port C เปลี่ยนจากลอจิก "0" เป็น "1"

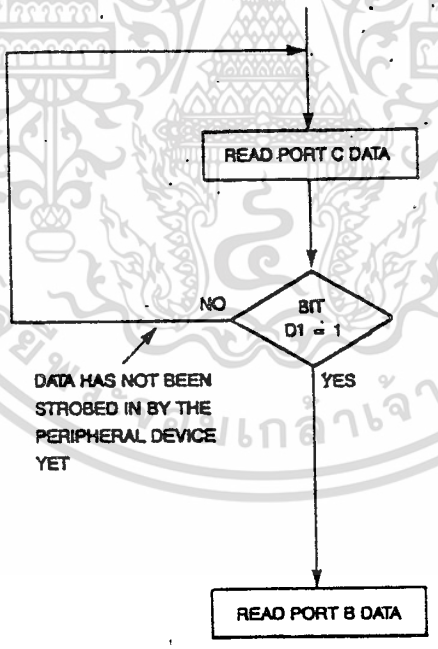
ในขณะที่ Bit D7 ของ Port C มีลอจิกเป็น "0" อุปกรณ์ภายนอกจะรับเอาข้อมูลจาก 8255 โดยการส่งสัญญาณ STROBE ACK ให้กับ Bit D6 ของ Port C และเมื่อมีการรับเอาข้อมูลออกไปแล้ว Bit D7 ของ Port C นี้ จะเปลี่ยนระดับลอจิกจาก "0" เป็น "1" ดังรูปที่ 31 นี้แสดงโฟลว์ชาร์ตของการทำงานดังกล่าว



รูปที่ 31 Flow Chart แสดงขบวนการการส่งข้อมูลออกไปยัง Port A และใช้ 4 Bitบน ของ Port C เป็นสัญญาณ HANDSHAKE

เอกสารนี้เป็นเอกสาร[illegible]ที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

ต่อไปเราจะศึกษาว่า CPU จะอ่านข้อมูลจากอุปกรณ์ภายนอกโดยขบวนการ HANDSHAKE ร่วมกับ 8255 ได้อย่างไร ก่อนที่ CPU จะรับเอาข้อมูลจาก Port Input เข้ามาได้นั้นจะต้องไปอ่านข้อมูลจาก Port C เพื่อตรวจสอบดูว่า Bit D1 มีลอจิกเป็นอะไร ถ้ามีลอจิกเป็น "1" แสดงว่าอุปกรณ์ภายนอกได้ส่งสัญญาณ STROBE ให้ 8255 รับเอาข้อมูลไปแล้ว โดยผ่านสายสัญญาณ STB (Bit D2 ของ Port C) ,CPU ก็จะสามารถอ่านข้อมูลจาก Port Input ได้ เวลานี้ INPUT BUFFER FULL (Bit D1 ของ Port C) จะเปลี่ยนลอกจิกเป็น "0" เพื่อบอกให้อุปกรณ์ภายนอกรู้ว่า 8255 พร้อมที่จะรับเอาข้อมูลใหม่แล้ว รูปที่ 32 นี้แสดงโฟลว์ชาร์ตของขบวนการต่าง ๆ ดังกล่าว

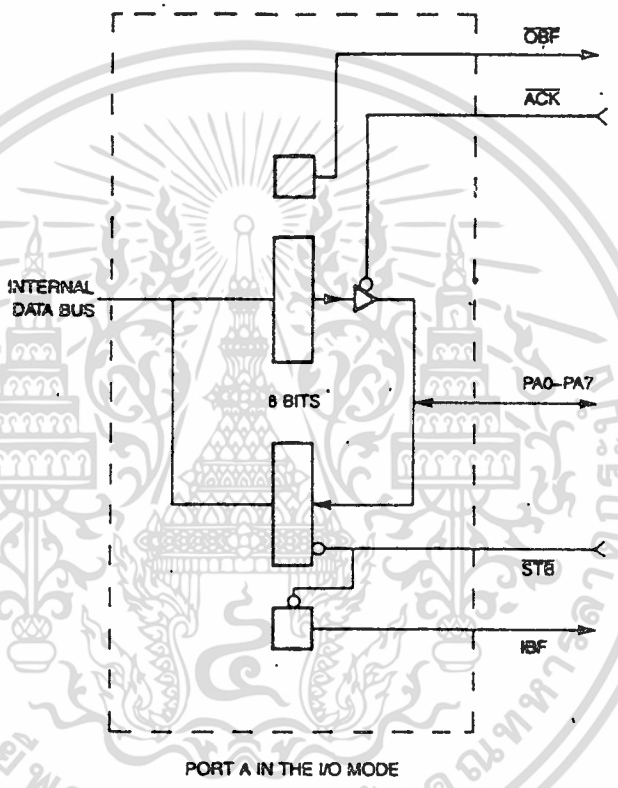


รูปที่ 32 โฟลว์ชาร์ตแสดงขบวนการการอ่านข้อมูลจากอุปกรณ์ภายนอกเข้ามายัง Port B โดยใช้ 4 Bitล่าง ของ Port C เป็น HANDSHAKE

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

การใช้งาน 8255 ในโหมด 2.

การทำงานของ 8255 ในโหมด 2 นี้จะเป็นการใช้งานในลักษณะที่ให้ Port A เป็น Port ข้อมูลแบบสองทิศทาง เมื่อ 8255 ถูกโปรแกรมให้ Port A อยู่ในโหมด 2 นี้แล้ว Port A จะมีลักษณการทำงานตาม Block Diagram ดังรูปที่ 33



รูปที่ 33 แสดง Block Diagram การทำงานของ 8255 ในโหมด 2

การทำงานในโหดนี้ก็คือ การใช้ Port A เป็น Input และOutput Latch Output Latch หมายถึง การเก็บเอาข้อมูลไว้เพื่อรออุปกรณ์ภายนอกรับเอาข้อมูลออกไป ส่วน Input Latch หมายถึงการเก็บข้อมูลที่อุปกรณ์ภายนอกส่งเข้ามาเพื่อรอให้ CPU อ่านเข้าไป ต่อไปเราจะพิจารณาการทำงานของ Block Diagram ว่าจะมีการรับหรือส่งข้อมูลกับอุปกรณ์ภายนอกได้อย่างไร

ในการส่งข้อมูลให้อุปกรณ์ภายนอกนั้น ขั้นตอนแรกที่เกิดขึ้นก็คือ CPU จะต้องส่งข้อมูลไป Latch ไว้ใน Port A ซึ่งจะทำให้ OUTPUT BUFFER FULL (OBF) ถูก Set ให้เป็นจริง สัญญาณนี้จะบอกให้อุปกรณ์ภายนอกรู้ว่าขณะนี้ Port A

เอกสารนี้เป็นเอกสารที่สงวนลิขสิทธิ์ ห้ามนำไปใช้ในการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

มีข้อมูลอยู่และบอกให้ CPU รู้ว่าข้อมูลของ Port A ที่ส่งไปให้นั้นยังไม่ถูกอุปกรณ์ภายนอกอ่านออกไป อุปกรณ์ภายนอกจะต้องส่งสัญญาณ ACK (ACKNOWLEDGE) ให้กับ 8255 สัญญาณนี้จะเป็นการ ENABLE ให้ข้อมูลที่อยู่ใน Port A ส่งออกไปยัง PA0-PA7 และเป็นการรีเซ็ท OBF เพื่อเป็นการบ่งบอกให้ CPU รู้ว่าข้อมูลที่อยู่ภายใน Port A ถูกอุปกรณ์ภายนอกอ่านออกไปแล้ว . และสามารถส่งข้อมูลใหม่ไปให้ Port A ได้อีก

ในการรับข้อมูลจากอุปกรณ์ภายนอกนั้น อุปกรณ์ภายนอกต้องตรวจสอบสถานะของ INPUT BUFFER FULL (IBF) เสียก่อน ถ้า IBF มีลอจิกเป็น "1" แสดงว่าขณะนี้ Port A มีข้อมูลอยู่ CPU ยังไม่ได้อ่านข้อมูลเข้าไป แต่ถ้ามีลอจิกเป็น "0" แสดงว่าไม่มีข้อมูลอยู่ใน Port A อุปกรณ์ภายนอกจะส่งข้อมูลและสัญญาณ STROBE (STB) ให้ 8255 สัญญาณนี้จะสั่งให้มีการนำข้อมูลไป Latch ไว้ใน Port A และSEt ให้ IBF มีสถานะเป็นจริง และ CPU ตรวจสอบสถานะของIBFได้โดยการอ่านข้อมูลจากPortCเมื่อ CPU อ่านข้อมูลออกไปแล้ว IBF จะมีสถานะลอจิกเป็น "0" และอุปกรณ์ภายนอกสามารถส่งข้อมูลใหม่เข้ามาได้

ในการใช้งานในโหมด 2 นี้ Port C จะเป็นตัวแสดงสภาวะของสัญญาณดังกล่าว และรายละเอียดของแต่ละ Bit จะแสดงดังรูปที่ 34

PORT C LINE	DEFINITION
PC0	I/O
PC1	I/O
PC2	I/O
PC3	$INTR_A$
PC4	$\overline{STB}_A$
PC5	IBF_A
PC6	$\overline{ACK}_A$
PC7	$\overline{OBF}_A$

รูปที่ 34 แสดงรายละเอียดของแต่ละขาของ Port C ในการใช้งานในโหมด 2

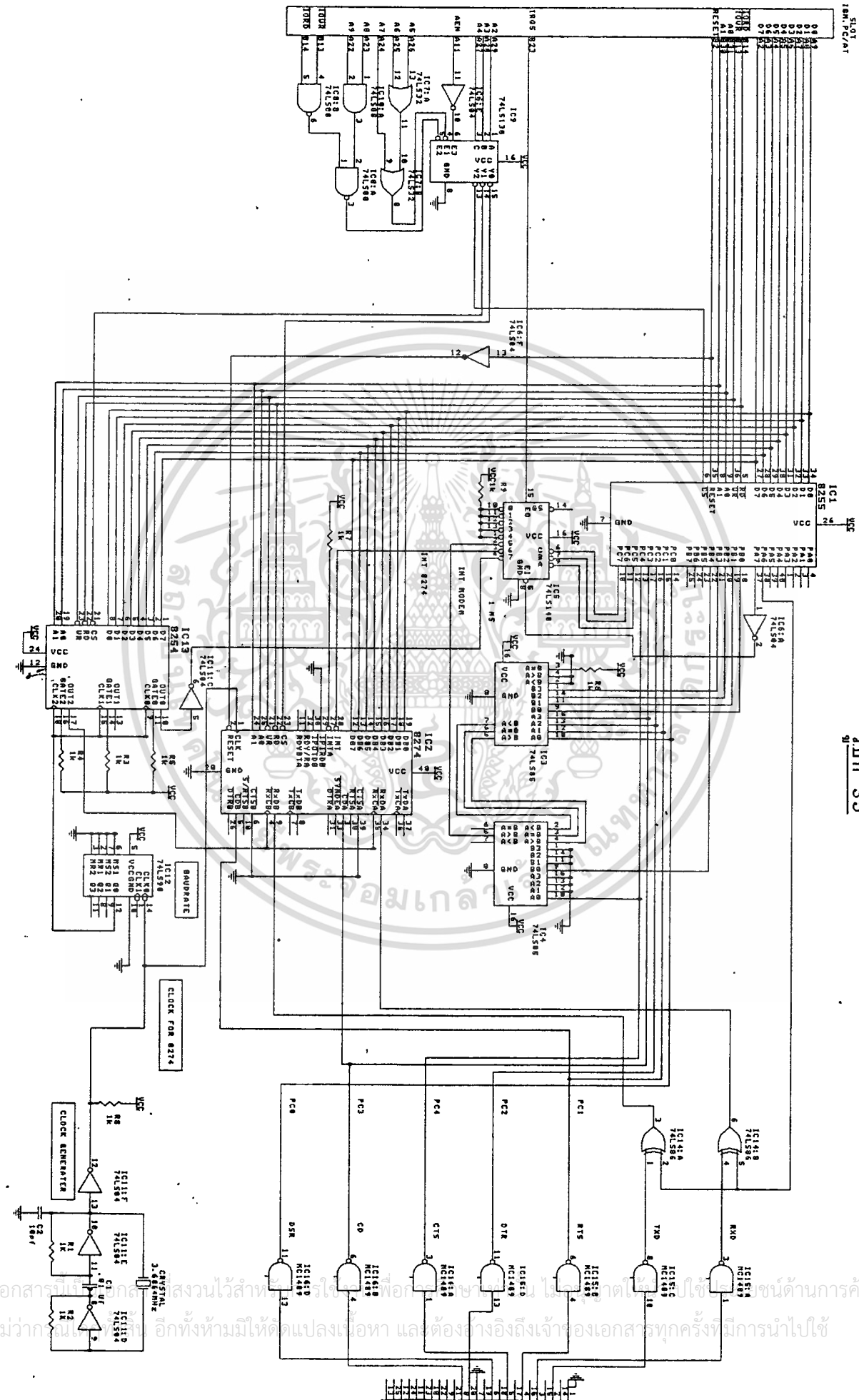
เอกสารนี้เป็นเอกสารที่สงวนลิขสิทธิ์ ห้ามนำไปใช้ในการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

การออกแบบและโครงสร้างทางฮาร์ดแวร์

การออกแบบวงจร MONITOR RS-232C ทำได้โดยการนำเอาไอซี มัลติโปรโตคอล ซีเรียลคอมมิวนิเคชั่น (Multi Protocol Serial Communication) เบอร์ 8274 ทำหน้าที่เป็นตัวจัดขบวนการต่าง ๆ สำหรับการสื่อสารข้อมูลแบบอนุกรม (Serial Communication) โดยทำการติดต่อระหว่างหน่วยประมวลผลกลาง (Central Processing Unit) กับการเชื่อมต่อสัญญาณสื่อสารภายนอก (Communication Line) โดยผ่านมาตรฐานการอินเตอร์เฟสการสื่อสารข้อมูลอนุกรมแบบ RS-232C ดังแสดงในรูปที่ 35 วงจรสนับสนุนการทำงานเพื่อประกอบเป็นวงจรสมบูรณ์แบบประกอบด้วย

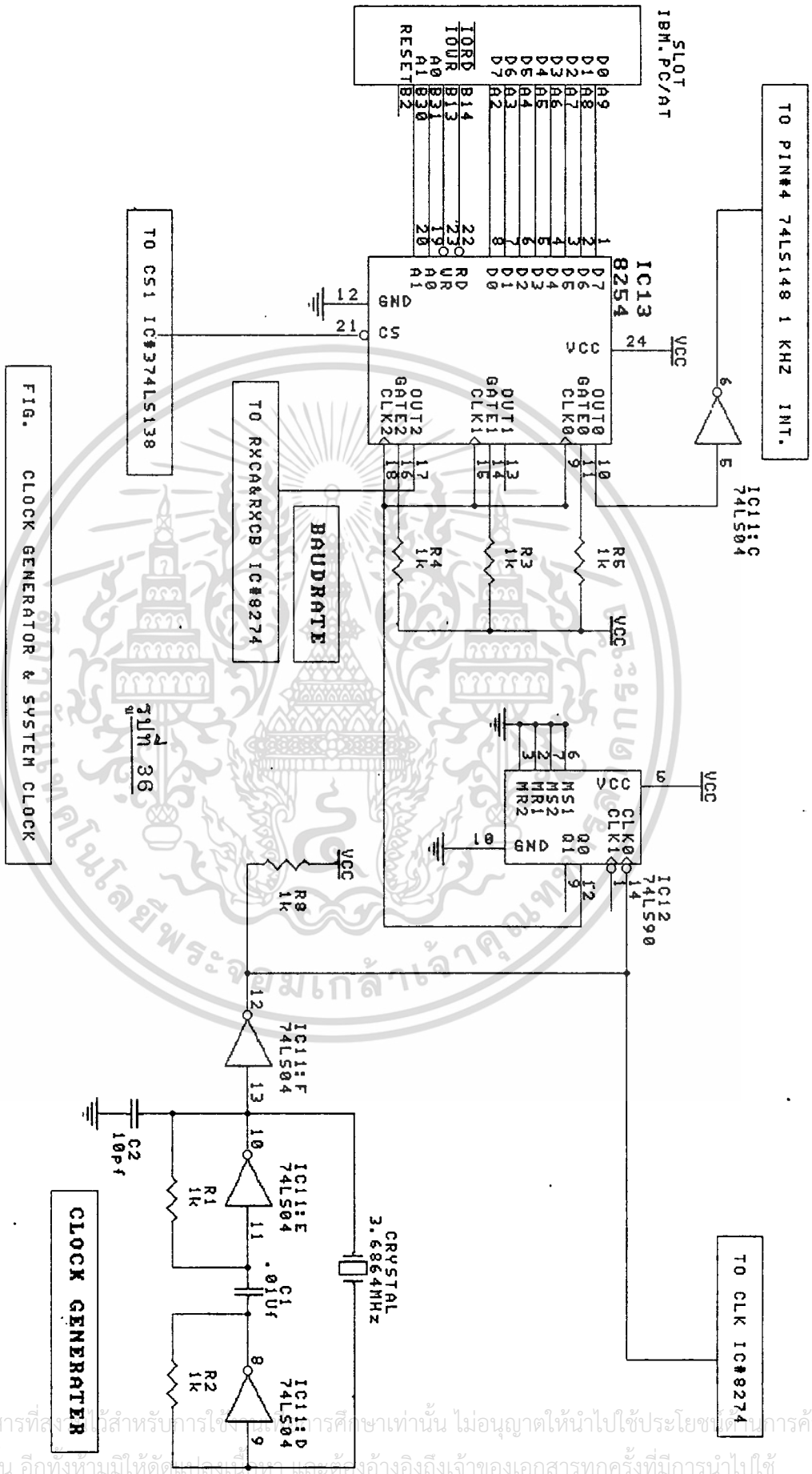
1. วงจรกำเนิดสัญญาณนาฬิกา และการสร้างสัญญาณนาฬิกาให้กับระบบการทำงาน (Clock Generator and System Clock Circuit) รูปที่ 36 ออกแบบโดยการใช้Inverter Gate แบบ TTL. เบอร์74LS04 โดยใช้ผลึกแร่ (Crystal) ที่ให้ความถี่ 3.6864 MHz. อุปกรณ์ที่ทำหน้าที่เป็นวงจรกำเนิดสัญญาณ คือ IC11:E, IC:11:D โดยมี ค.ต.ท R_1 ค่า 1KR, R_2 ค่า 1KR,C1 ค่า 0.1uF เป็นอุปกรณ์ต่อร่วม C_2 10pF ทำหน้าที่ Filter ให้กับชุดกำเนิดความถี่ R_s ค่า 1KR และ IC11:F ทำหน้าที่เป็นตัวช่วยให้สัญญาณความถี่มีรูปร่างดีขึ้น ความถี่ที่ได้จากวงจรกำเนิดสัญญาณนาฬิกาจะถูกหารโดย IC12 #74LS90 โดยความถี่นี้จะถูกหาร 2 ทำให้ความถี่ลดลงเป็น 1.8432 MH_z เพื่อเป็นสัญญาณนาฬิกาให้กับ IC13 #8254 ซึ่งเป็น Programmable Interval timmer ที่มีชุด Programmable Counter อยู่ 3 ชุดด้วยกัน ในที่นี้เราจะใช้เพียงแค่ 2 ชุดเท่านั้นคือ ชุดCounter 0 และ Counter 2 โดย Counter 2 จะทำหน้าที่หารความถี่เพื่อเป็นตัวกำหนดความถี่สัญญาณนาฬิกาให้กับการรับส่งข้อมูล (Transmitted and Received Clock) หรืออัตราบอร์ด (Baud Rate) การเปลี่ยนแปลงอัตราบอร์ดทำได้โดยการกำหนดค่าตัวหารให้กับ IC13 #8254 สัญญาณ BAUDRATE ที่ได้นี้จะส่งไปเข้าที่ขา RXCA และRXCB ของ IC2# 8274 เพื่อเป็นตัวกำหนดความถี่ในการรับส่งข้อมูลให้กับIC2 #8274 อีกทีหนึ่ง ส่วน Counter 0 จะถูกสั่งให้หารให้ได้ความถี่ที่ Output 0 มีค่าเท่ากับ 1KH_z ซึ่งความถี่ 1KH_z นี้

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้



รูปที่ 35

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับใช้เพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้



เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

จะเป็นสัญญาณขัดจังหวะการทำงานทุกๆ 1ms. (Interput) เพื่อให้หน่วยประมวลผลกลาง(Central Processing Unit,CPU) ในเครื่องคอมพิวเตอร์มาทำการอ่านข้อมูลจาก IC2 #8274 ไปเพื่อแสดงผลต่อไป

2. วงจรอ้างตำแหน่งพอร์ท (Port addressing) เนื่องจากได้ออกแบบให้การอ้างตำแหน่งของพอร์ทอยู่ในช่วง 0300H-03FFFH ซึ่งเป็นตำแหน่งของพอร์ทที่ไม่ได้นำออกไปใช้งานภายในเครื่อง IBM/PC การอ้างตำแหน่งของพอร์ทจึงใช้ขา Address จาก A0-A9 และ AEN ซึ่งเป็น Address Enable เพราะในช่วงที่หน่วยประมวลผลกลางถูกตัดออกจากระบบบัส คือสภาวะ Direct Memory Access สัญญาณบนขา AEN จะถูกทำให้เป็น High ดังนั้นเพื่อป้องกันการอ้างตำแหน่งของอุปกรณ์อินพุทเอาท์พุทไปทับกับตำแหน่งที่ถูก DMA จึงนำเอาสัญญาณ AEN มาทำการห้าม (Disable) การอ้างตำแหน่งของพอร์ทในสภาวะนี้

ตำแหน่งของพอร์ทที่ได้จากการออกแบบวงจรคือพอร์ทหมายเลข 0300H-0303H สำหรับ 8274 พอร์ทหมายเลข 0304H-0307H สำหรับ 8254 และพอร์ทหมายเลข 0308H-030BH สำหรับ 8255 วงจรที่ได้จากการออกแบบแสดงดังรูปที่ 37

3. วงจรสร้างสัญญาณขัดจังหวะการทำงาน (Interupt Signal Generator) การขอขัดจังหวะการทำงาน 8086 เกินขึ้นได้ 3 กรณี เมื่อเรียงตามลำดับความสำคัญคือ

1. การจัดจังหวะการทำงานทุก ๆ หนึ่งมิลลิวินาที เพื่อเป็นฐานเวลาให้กับ Timmer ภายในโปรแกรมทดสอบการทำงาน

2. การขัดจังหวะการทำงานจาก 8274 ซึ่งจะมีได้เมื่อมีเหตุการณ์เหล่านี้ เกิดขึ้นคือ

ก. เมื่อมีการส่งข้อมูล ซึ่งหมายถึง บัฟเฟอร์ของตัวส่งว่าง (Tx Buffer Empty)

ข. เมื่อมีการรับข้อมูล ซึ่งหมายถึง บัฟเฟอร์รับมีข้อมูลเข้ามา (Rx Buffer Available)

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

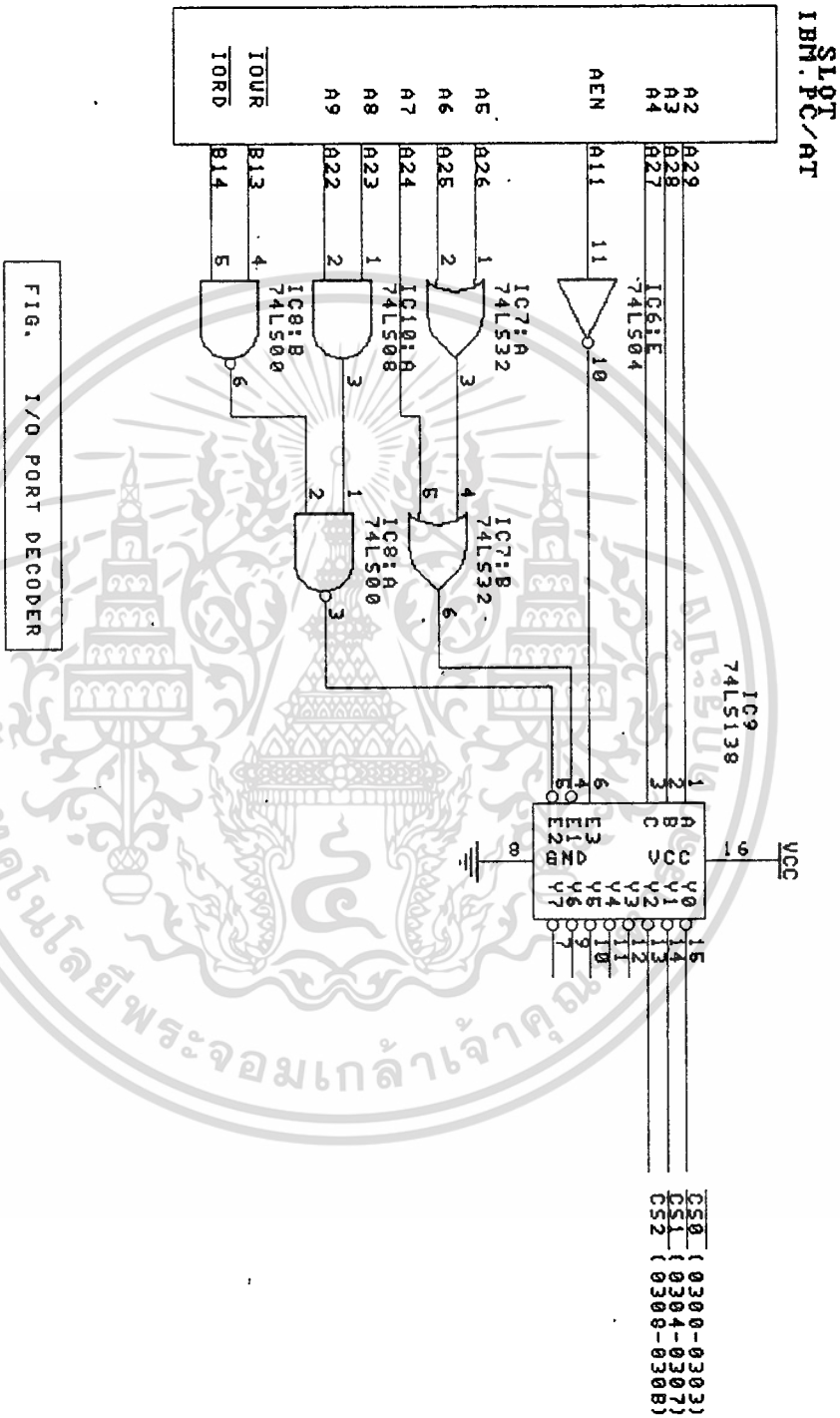


FIG. I/O PORT DECODER

รูปที่ 37

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

ค. เมื่อมีการผิดพลาดเกิดขึ้นจากสถานะภายใน 8274 คือ

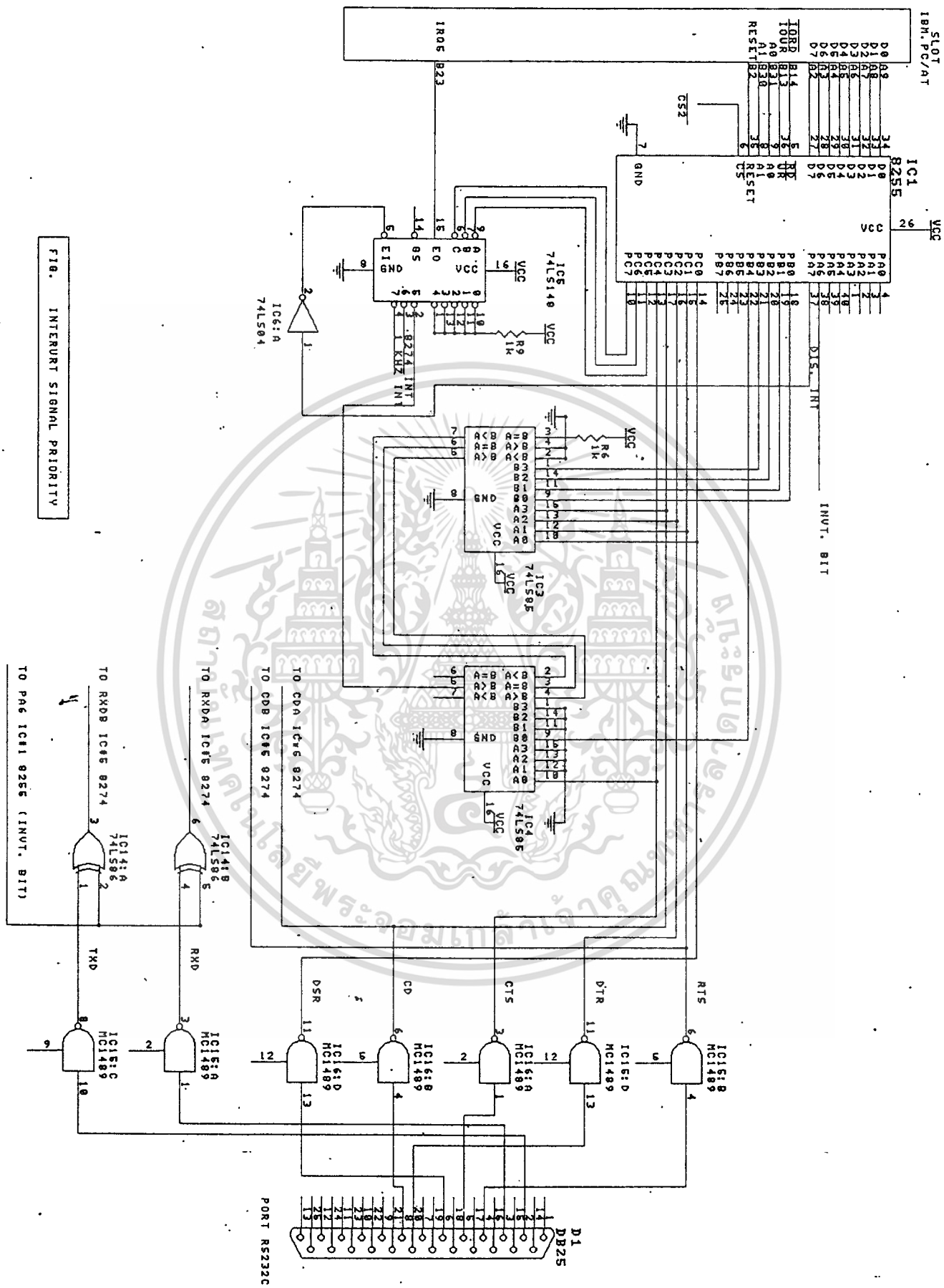
- Overrun
- Framing Error
- Break/Abort
- Parity Error

4. การขัดจังหวะการทำงานเนื่องจากการเปลี่ยนสถานะของสัญญาณควบคุมโมเด็มสัญญาณเหล่านี้คือ RTS CTS DTR DSR และ CD

การตรวจจับการเปลี่ยนแปลงสถานะของสัญญาณควบคุม Moderm เราจะใช้หลักการ Magnitude Comparater โดยใช้ IC #74LS85 (IC#3,IC#4) มาเป็นตัวเปรียบเทียบสัญญาณที่มาจาก Moderm กับสัญญาณที่ได้มาจาก Port B ของ IC#1 8255 โดย IC#3 74LS85 จะเปรียบเทียบขา DSR กับ P_{B0}, RTS กับ P_{B1}, CD กับ P_{B3} ถ้าหากเกิดการเปลี่ยนแปลงสถานะของสัญญาณ Moderm ขึ้นจะทำให้ที่ขา 5,6,7 ของIC#3 74LS85 เกิดการเปลี่ยนแปลงด้วยซึ่งขา 5,6,7 ของ IC#3 74LS85 นี้จะถูกต่อกับขา 4,3,2 ของ IC#4 74LS85 ตามลำดับ ซึ่งที่ IC#4 74LS85 นี้จะทำการเปรียบเทียบผลที่ได้มาจาก IC#3 74LS85 กับผลที่ได้จากการเปรียบเทียบระหว่างขา CTS กับขาของ P_{B4} ของ IC#1 8255 ถ้าผลที่ได้คือมีการเปลี่ยนแปลงเกิดขึ้นที่ขา 6 ของ IC#4 74LS85 ก็จะ Active คือมีสถานะเป็น "0" แล้วส่งผลไปให้ที่ขา 2 ของ IC# 74LS148 ที่ IC#5 74LS148 ซึ่งเป็น IC Encoder จะทำหน้าที่จัดลำดับความสำคัญของการ Interput ต่าง ๆ ที่ส่งมาเข้าที่ขา 4,3,2 เมื่อเกิดการ Interput เกิดขึ้นที่ขา 4,3,2 จะทำให้ EO ของ IC#5 74LS148 เกิดสัญญาณขึ้นเพื่อไป Interput การทำงานของตัว CPU 8086 ที่ IRQ5 ซึ่ง CPU จะไป Run Program เพื่อมาอ่านดูว่าเป็นการ Interput มาจากที่ไหน ขา A0,A1,A2 ของ IC#5 74LS148 ซึ่งจะดูได้จากที่ Encoder โดยอาศัย Port C ของ IC#1 8255 มาเป็นตัวช่วยรับข้อมูลแล้วส่งให้ CPU 8086 อีกทีหนึ่ง

การห้ามการขัดจังหวะการทำงาน(Disable Interput) ทำได้โดยการส่งข้อมูลมาที่ Port A ของ IC#1 8255 ที่ขา PA7 โดยขา PA7 นี้จะถูกต่อ

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้



รูปที่ 38

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

กับขา 1 ของ IC# 74LS04 Inverter เพื่อเปลี่ยนสถานะให้เป็นตรงกันข้าม แล้วต่อไปยังขา EI ของ IC#5 74LS148 เพื่อห้ามการขัดจังหวะที่จะเกิดขึ้นต่อไป

5. วงจรขับ และรับแรงดันสำหรับมาตรฐานการอินเตอร์เฟสแบบ RS-232C บนตำแหน่งขาสัญญาณเป็นแรงดัน 3 ระดับ คือ +12 โวลต์ -12 โวลต์และ 0 โวลต์ ดังนั้นจึงต้องมีการเปลี่ยนระดับของแรงดันที่เป็นมาตรฐานของ TTL ให้เป็นแรงดันมาตรฐานของ RS-232C โดยการใช้ไอซี RS-232C Line ReceiverMC1489แปลงระดับแรงดันขาเข้าซึ่งเป็นระดับแรงดันของ RS-232C มาเป็นระดับแรงกันของ TTL วงจรแปลงระดับของสัญญาณทั้งสอง แสดงดังรูปที่ 38

ที่ขา R_{xD} และ T_{xD} จะถูกต่อผ่าน IC14:A และ IC14:B ซึ่งเป็น Excusive-or gate เพื่อทำหน้าที่ Inverse สัญญาณที่รับเข้ามาให้มีลักษณะตรงกันข้ามกับ Output ที่ออกไป โปรแกรมการควบคุมการ Inverse นี้จะกระทำโดยอาศัยขา PA6 ของ Port A และ IC#1 8255 เป็นตัวส่งสัญญาณควบคุมให้อีกทีหนึ่ง

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

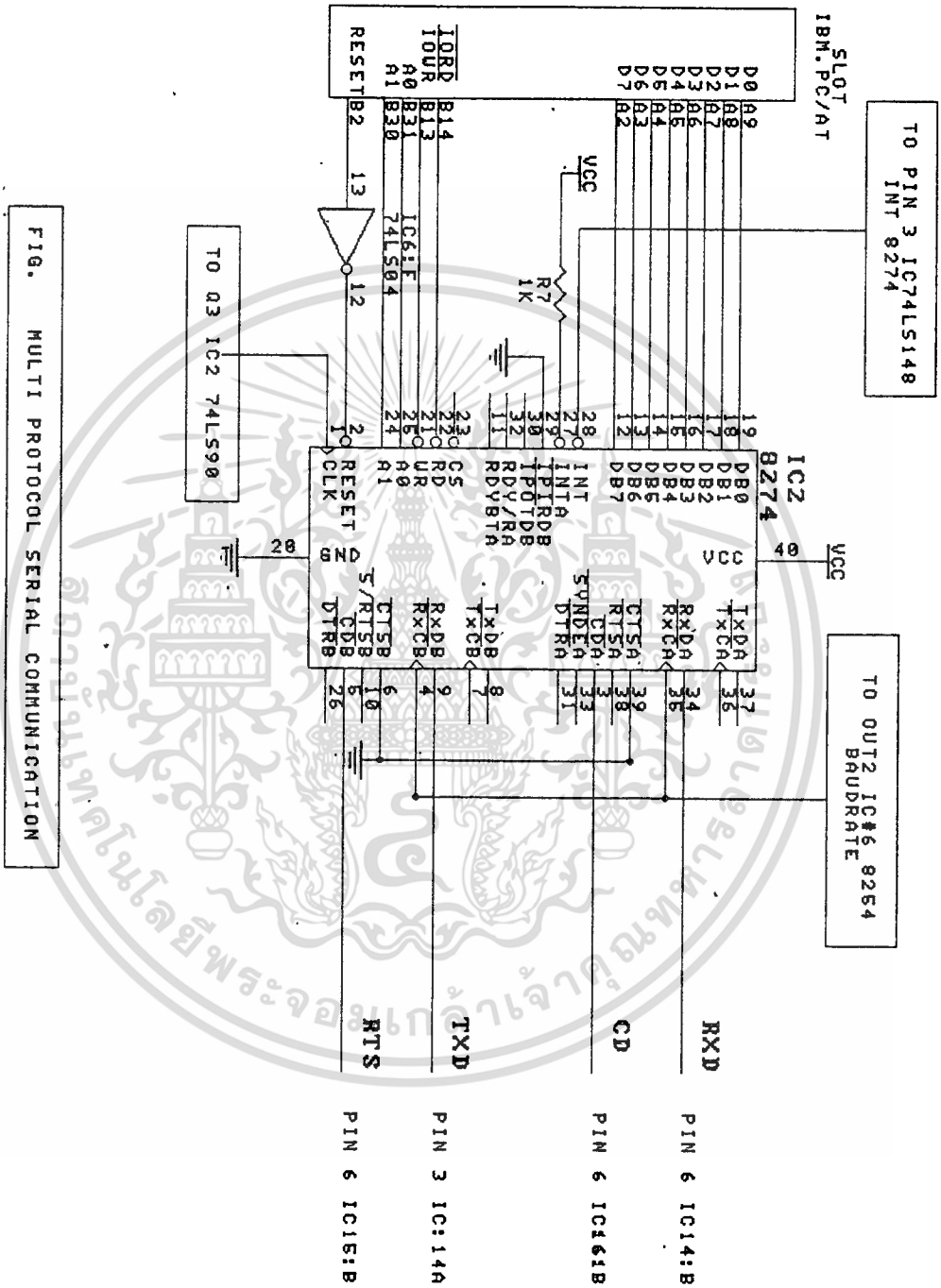


FIG. MULTI PROTOCOL SERIAL COMMUNICATION

รูป 39

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

บทที่ 5

โปรแกรม RS-232C Monitor

เป็นโปรแกรมที่เขียนขึ้นมาเพื่อควบคุมการทำงานของวงจรที่กล่าวมาทั้งหมด โปรแกรมควบคุมการทำงานทั้งหมดสามารถเขียนออกมาในลักษณะของภาษาการโปรแกรม ลอจิก ดีไซน์ (Program Logic Design หรือ PLD) ได้ดังนี้

โปรแกรมหลัก

```
Procedure  main
    เลือกการแสดงผลเป็น colour text
    ตรวจสอบการ์ดแสดงผล
    if การ์ดแสดงผลไม่ใช่ EGA หรือ VGA
       จบโปรแกรม
    endif
    if  พบแฟ้มข้อมูลของตัวอักษรสำหรับการแสดงผล
       ติดตั้งตัวอักษรสำหรับการแสดงผล
    else จบโปรแกรม
    endif
    จองเนื้อที่ในหน่วยความจำให้กับตัวแปรต่าง ๆ
    if เนื้อที่ในหน่วยความจำไม่พอ
          จบโปรแกรม
    endif
  โปรแกรมค่าให้กับอุปกรณ์ในการ์ดโปรโตคอล
    เลือกฟังค์ชั่น
    คืนหน่วยความจำให้ระบบ
    กลับไปใช้ตัวอักษรแบบเดิม
    จบโปรแกรม
```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```
endproc.
```

การติดตั้งตัวอักษรสำหรับการแสดงผล

```
Procedure  Init-font
     จองเนื้อที่หน่วยความจำสำหรับเก็บรูปแบบตัวอักษร
     อ่านข้อมูลจากแฟ้มข้อมูล
     ติดตั้งตัวอักษรใหม่
endproc.
```

โปรแกรมค่าให้กับอุปกรณ์

```
Procedure  Init-hardware
     ให้ Counter 2 ของ 8254 เป็นตัวกำเนิดความถี่ 1000 Hz.
     ให้ Port A และB ของ 8255 ที่ Output Port
     ให้ Port C ของ 8255 เป็น Input Port
     ยอมให้มีการอินเตอร์รัพท์  ผ่านขา IRQ3,IRQ5 ของ 8259
endproc.
```

เลือกฟังค์ชั่น

```
Procedure  Get-function
declare  fc := 0
enddeclare
     ลบจอภาพทั้งหมด
     สร้างหน้าต่างหมายเลข 1 ชื่อ "RS-232C MONITOR"
```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```
        สร้างเมนู
          fc := 0
          do
          case of  (การกดคีย์บอร์ด)
             case  (เปลี่ยนโปรโตคอล)
                fc := 1
             case  (ตรวจจับข้อมูล)
                fc := 2
             case  (ทดสอบความผิดพลาดของการส่งข้อมูล)
                fc := 3
             case  (ทดสอบตัวเอง)
                fc := 4
             case  (จบโปรแกรม)
                fc := 5
             case  (เกี่ยวกับโปรแกรม)
                fc := 6
          endcase
          เลือกฟังค์ชั่นย่อย (fc)
      until จบโปรแกรม
endproc.
```

เลือกฟังค์ชั่นย่อย

```
Procedure  Pull-down (fc)
declare  fn := 0
enddeclare
        สร้างหน้าต่างของฟังค์ชั่นย่อย
```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```
do
  case of  (การกดคีย์บอร์ด)
     case  (up arrow)
     case  (down arrow)
          เลื่อนแถบสว่างไปตามการกดคีย์
          เพิ่มหรือลดค่า fn ตามการกดคีย์
     case  ("Enter")
          กระทำฟังค์ชั่นย่อย (fc, fn)
  endcase
until  กด "Escape" หรือ Hot Key
ปิดหน้าต่างฟังค์ชั่นย่อย
endproc.
```

<u>กระทำฟังค์ชั่นย่อย</u>

```
Procedure Do-function (fc, fn)
case of  (fc)
     case  (fc = เปลี่ยนโปรโตคอล)
       case of  (fn)
          case  (fn = 1)
                เลือกโปรโตคอล
          case  (fn = 2)
                อ่านโปรแกรมจากแผ่นจานแม่เหล็ก
          case  (fn = 3)
                เก็บโปรแกรมลงแผ่นจานแม่เหล็ก
       endcase
```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```
case  (fc = ตรวจจับข้อมูล)
      ตรวจจับข้อมูล

case  (fc = ทดสอบความผิดพลาดของการส่งข้อมูล)
  case of  (fn)
     case  (fn = 1)
           เลือกลักษณะการทดสอบ Bit Error Rate
    case  (fn = 2)
           เริ่มทดสอบ Bit Error Rate
  endcase

case  (fn = ทดสอบตัวเอง)
  case of  (fn)
     case  (fn = 1)
           แสดงตัวอักษร
  endcase

case  (fc = จบโปรแกรม)
  case of  (fn)
     case  (fn = 1)
           ออกจากโปรแกรม
     case  (fn = 2)
           กลับไปทำอีก
  endcase

case  (fc = เกี่ยวกับโปรแกรม)
  case of  (fn)
     case  (fn = 1)
```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

แสดงเกี่ยวกับโปรแกรม

```
        endcase
      endcase
  endproc.
```

<u>แก้ไขการกำหนดโปรโตคอล</u>

```
  Procedure  Edit-setup
declare prot-setup:record
      level 01 proc                    :unsigned character
      level 01 max(MAX-PROC)           :unsigned character
      level 01 mn(MAX-SETUP)           :pointer to pointer
    level 01 st(MAX-PROC)            :array of pointer
        level 02 val                 : character
        level 02 sel(MAX-SPEED) : array of character
  enddeclare
      สร้างหน้าต่าง หมายเลข 3 ชื่อ "Setup Protocal"
      แสดงหัวข้อทั้งหมด บนหน้าต่างหมายเลข 3
      do
          case of  (การกดคีย์บอร์ด)
             case  (กด Space Bar)
                 เปลี่ยนตัวเลือกในหัวข้อนั้น ๆ
             case  (กด Right Arrow)
             case  (กด Left Arrow)
             case  (กด Up Arrow)
             case  (กด Down Arrow)
                 เปลี่ยนหัวข้อ
```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```
                case  (กด Enter)
                    เปลี่ยนค่าตัวเลือกของหัวข้อนั้น ๆ ให้เป็นตามที่เลือกไว้
        "       endcase
        until กด "Escape"
        ปิดหน้าต่างหมายเลข 3
    endproc.
```

ตรวจจับข้อมูล

```
    Procedure  Run-program
    declare
    DATA-STRUC : Record
        level 01 misc     : unsigned character
        level 01 data     : unsigned character
        level 01 lead     : unsigned character
    end record
    enddeclare
        สร้างหน้าต่างหมายเลข 3
        เตรียมค่าตัวแปรต่าง ๆ
        เขียนหน้าจอ เพื่อเตรียมการแสดงข้อมูล
        เตรียมค่าเวกเตอร์ของการอินเตอร์รัพท์
        โปรแกรมค่าของอุปกรณ์บนการ์ดโปรโตคอล
      do
            แสดงข้อมูลบนจอภาพ
            เขียนข้อมูลลงบน Hard Disk
            if ยังไม่จบโปรแกรม
                แสดงข้อมูลต่อไป
        "   endif
```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```
        until  มีการกดคีย์ เพื่อออกจากการทำงาน
        หยุดการเขียนข้อมูลลงบนจอภาพ
        หยุดการเขียนข้อมูลลงบนแผ่นจานแม่เหล็ก
        คืนค่าเวกเตอร์ของการอินเตอร์รัพท์
        ปิดหน้าต่างหมายเลข 3
    endproc.
```

<u>อ่านโปรแกรมจากแผ่นจานแม่เหล็ก</u>

```
    Procedure load-data
        รับชื่อแฟ้มข้อมูล
        if เปิดแฟ้มข้อมูลได้
            ตรวจสอบรหัสของแฟ้มข้อมูล
          if รหัสถูกต้อง
              อ่านค่าตัวแปรของโปรโตคอล
             do อ่านข้อมูลเข้ามาไว้ที่หน่วยความจำ
                if หมดข้อมูล
                    ปิดแฟ้มข้อมูล
                    break
                endif
             until หน่วยความจำเต็ม
          endif
        endif
    endproc.
```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

เก็บโปรแกรมลงแผ่นจานแม่เหล็ก

```
Procedure Save-data
     รับชื่อแฟ้มข้อมูล
     if พบแฟ้มข้อมูลนี้อยู่ในแผ่นจานแม่เหล็กแล้ว
          แสดงข้อความ "Overide ......(Y/N)"
          if ตอบ "N" หรือ กด "Escape"
             return
          endif
          if เปิดแฟ้มข้อมูลไม่ได้
             แสดงข้อความ "Can not open or creat file....."
             return
          endif
     endif
     เก็บรหัสลักษณะแฟ้มข้อมูล
     เก็บค่าตัวแปรของโปรโตคอล
     เก็บโปรแกรมทั้งหมด
endproc.
```

เลือกลักษณะการทดสอบ Bit Error Rate

```
Procedure Edit-BERT
declare
     Bert-Setup  : record
       level 01 Sa[MAX_BERT]        : array of record
          level 02 Val : character
```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```
            level 02 Sel[MAX_SPEED]  : ARRAY OF CHAR.
enddeclare
     สร้างหน้าต่างหมายเลข 3 ชื่อ "Bit Parameter Set Up"
     แสดงหัวข้อทั้งหมด  ลงในหน้าต่างหมายเลข 3
     do
          case of (การกดคีย์บอร์ด)
             case  (Space Bar)
               เปลี่ยนตัวเลือกให้หัวข้อนั้น ๆ
             case  (กด Right Arrow)
             case  (กด Left Arrow)
             case  (กด Up Arrow)
             case  (กด Down Arrow)
                   เปลี่ยนหัวข้อ
             case  (กด Enter)
                 เปลี่ยนค่าตัวเลือกของหัวข้อนั้น ๆ ให้เป็นตามที่เลือกไว้
          endcase
     until กด "Escape"
     ปิดหน้าต่างหมายเลข 3
endrroc.
```

<u>การทดสอบ Bit Error Rate</u>

```
Procedure Run-BERT
     เปิดหน้าต่างหมายเลข 3 ชื่อ "Bit Error Rate Test"
     เขียนหน้าจอเตรียมแสดงผล
     เตรียมค่าเวกเตอร์ของการอินเตอร์รัพท์
     โปรแกรมค่าของอุปกรณ์บนการ์ดโปรโตคอล
```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```
        do
              แสดงผลการทดสอบ
              case of รับคีย์บอร์ด
                 case (F4)
                    แทรกบิทที่ผิดลงไป 1 บิท
                 case (F5)
                    แทรกบิทที่ผิดลงไป 10 บิท
                 case (F9)
                    เริ่มทำงานใหม่
                 case (F10)
                    หยุดการทำงาน
              endcase
        until  กด "Escape"
        คืนค่าเวกเตอร์ของการอินเตอร์รัพท์
        ปิดหน้าต่างหมายเลข 3
    endproc.
```

<u>การแสดงตัวอักษร</u>

```
    Procedure Disp-Scrn
        เปิดหน้าต่างหมายเลข 3 ชื่อ "Display Fonts"
        do
            แสดงตัวอักษร ASCII
            แสดงตัวอักษร HEX
        until  มีการกดคีย์ เพื่อออกจากการแสดง
        ปิดหน้าต่างหมายเลข 3
    endproc.
```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

การออกจากโปรแกรม

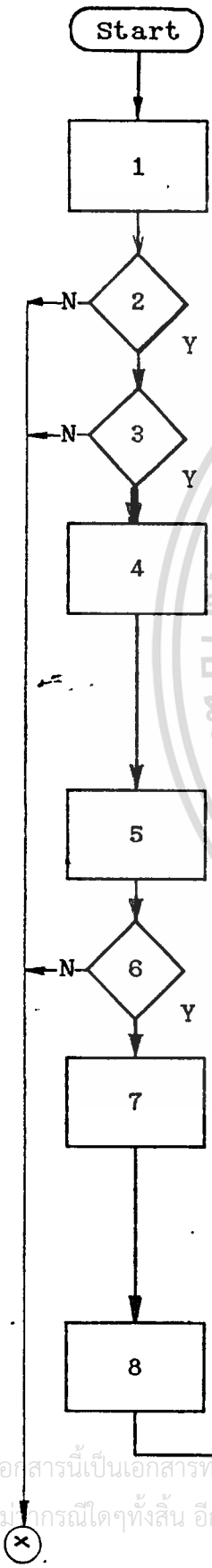
```
Procedure Quitelse
        คืนหน่วยความจำทั้งหมดให้กับระบบ
        if ยังไม่ได้ปิดแฟ้มข้อมูล
            ปิดแฟ้มข้อมูล
        endif
        ปิดหน้าต่างหมายเลข 1
endproc.
```

การแสดงเกี่ยวกับโปรแกรม

```
Procedure About
        เปิดหน้าต่างหมายเลข 3 ชื่อ "About"
        do
            แสดงเกี่ยวกับโปรแกรม
        until  มีการกดคีย์ เพื่อออกจากการแสดง
      ปิดหน้าต่างหมายเลข 3
endproc.
```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

Flow chart แสดงการทำงานของโปรแกรม MONITOR RS-232C



เลือกการแสดงผลเป็น colour text

ตรวจสอบการ์ดแสดงผลว่าเป็น EGA หรือ VGA หรือไม่

พบแฟ้มข้อมูลของตัวอักษรสำหรับการแสดงผล

ติดตั้งตัวอักษรสำหรับการแสดงผล

1. จองเนื้อที่หน่วยความจำสำหรับการเก็บรูปแบบตัวอักษร
2. อ่านข้อมูลจากแฟ้มข้อมูล
3. ติดตั้งตัวอักษร

จองเนื้อที่ในหน่วยความจำให้กับตัวแปรต่าง ๆ

เนื้อที่ในหน่วยความจำพอหรือไม่

โปรแกรมค่าให้กับอุปกรณ์ในการ์ดโปรโตคอล

ให้ Counter 2 ของ 8254 เป็นตัวกำเนิดความถี่ 1000 Hz.

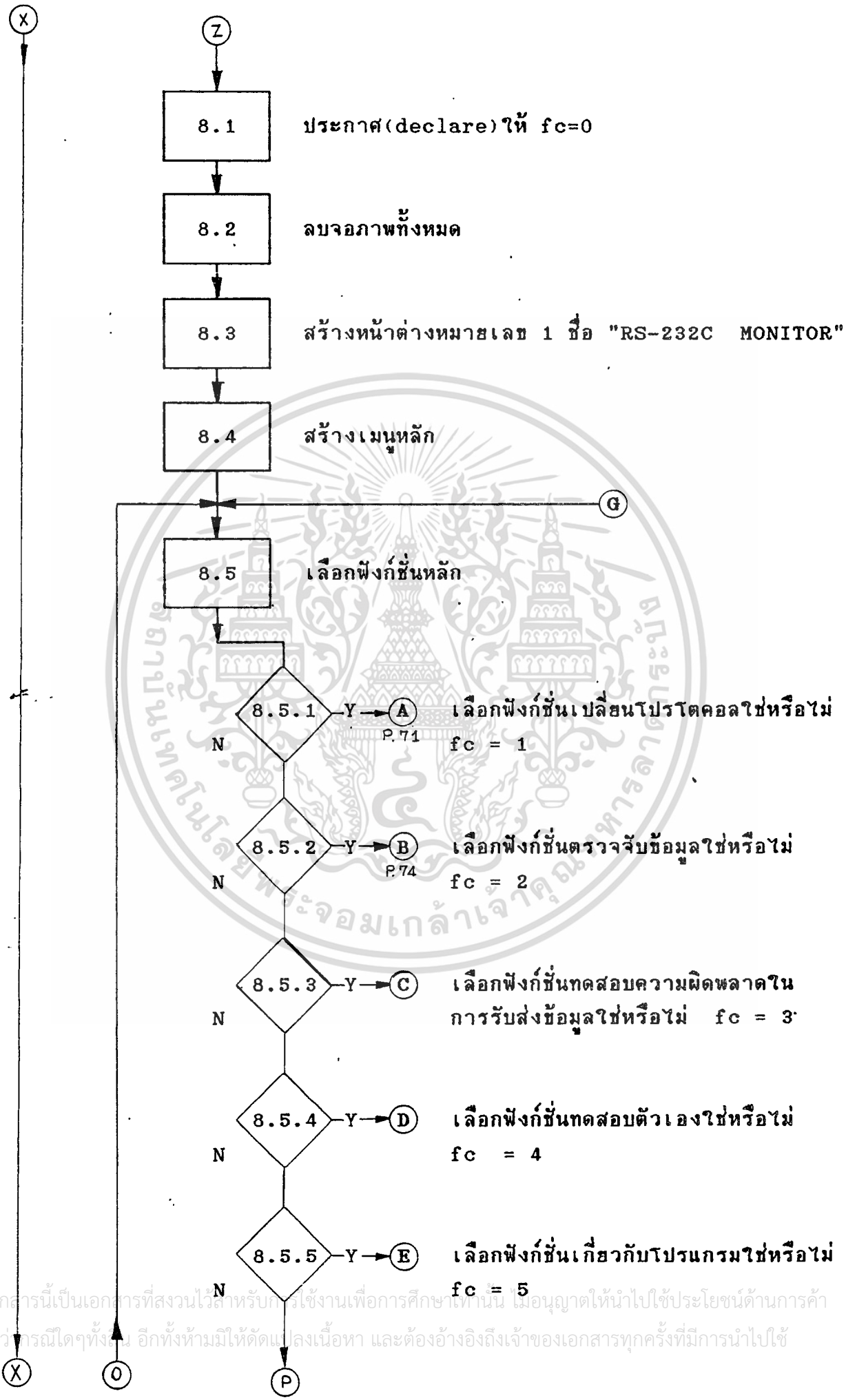
ให้ Port A และB ของ 8255 ที่ Output Port

ให้ Port C ของ 8255 เป็น Input Port

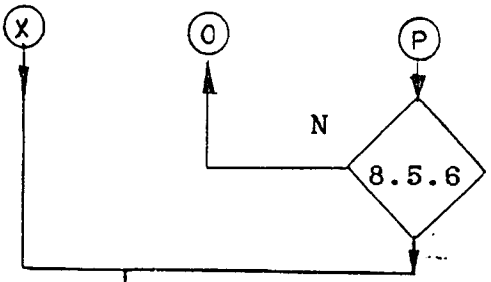
ยอมให้มีการอินเตอร์รัพท์ ผ่านขา IRQ3,IRQ5 ของ 8259

เลือกฟังก์ชั่น

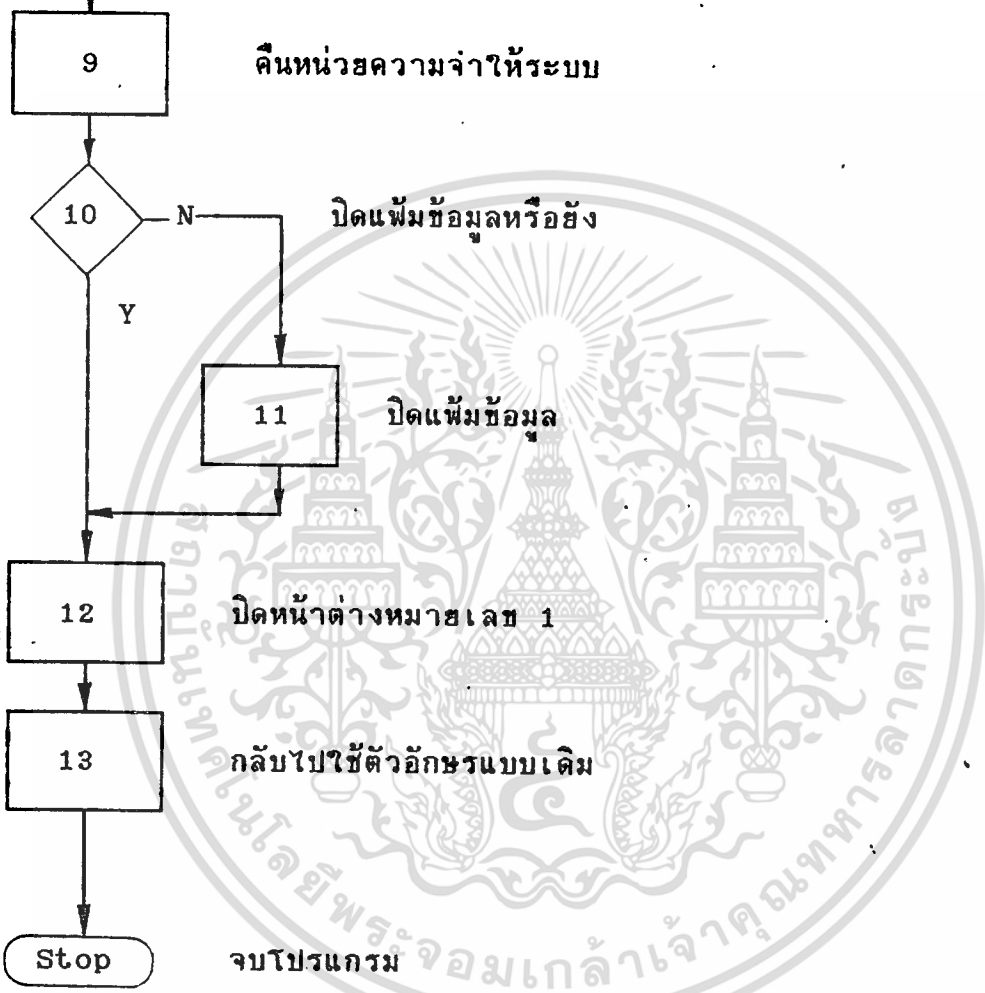
เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้



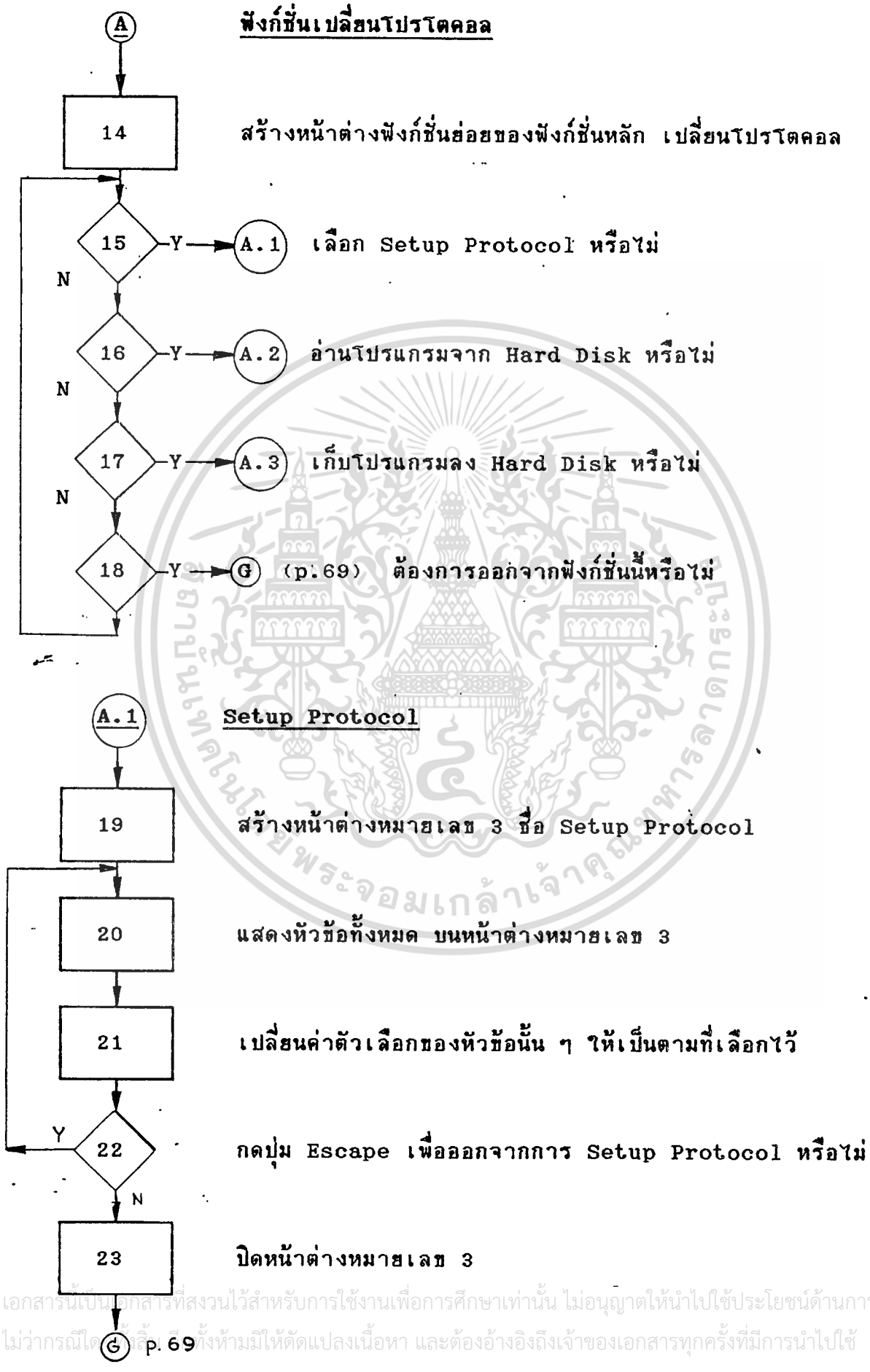
เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้



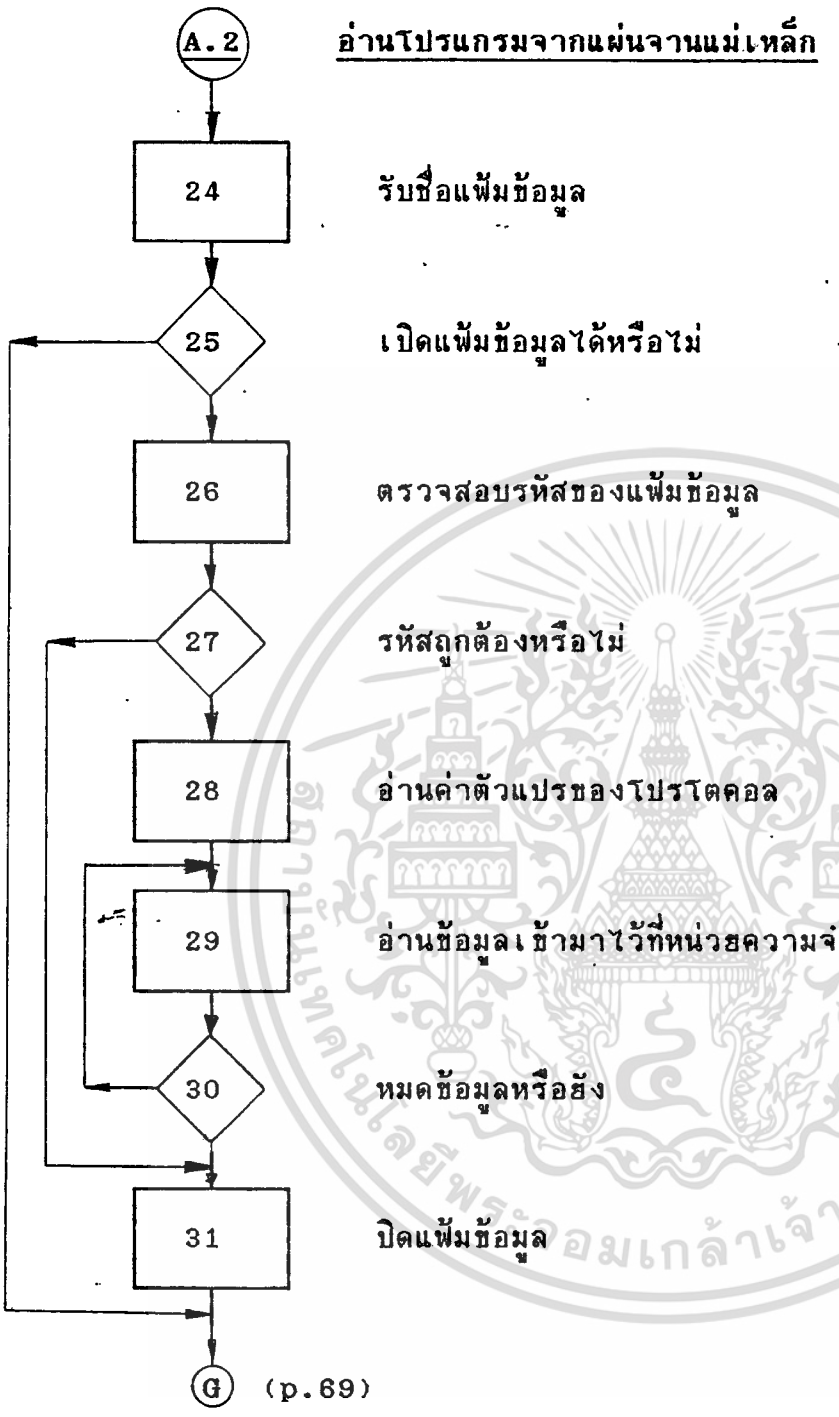
เลือกฟังก์ชั่นออกจากโปรแกรมใช่หรือไม่
fc = 6



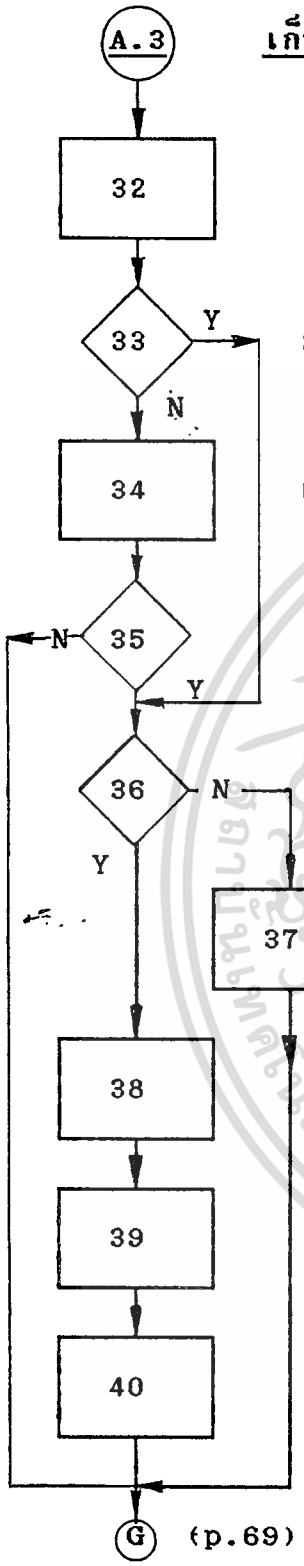
เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้



เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้



เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้



เก็บโปรแกรมลงแผ่นจานแม่เหล็ก

รับชื่อแฟ้มข้อมูล

มีแฟ้มข้อมูลนี้อยู่ในแผ่นจานแม่เหล็กหรือไม่

แสดงข้อความ "Overide .....(Y/N)"

ต้องการเขียนข้อมูลใหม่ทับข้อมูลเดิมหรือไม่

แฟ้มข้อมูลเปิดได้หรือไม่

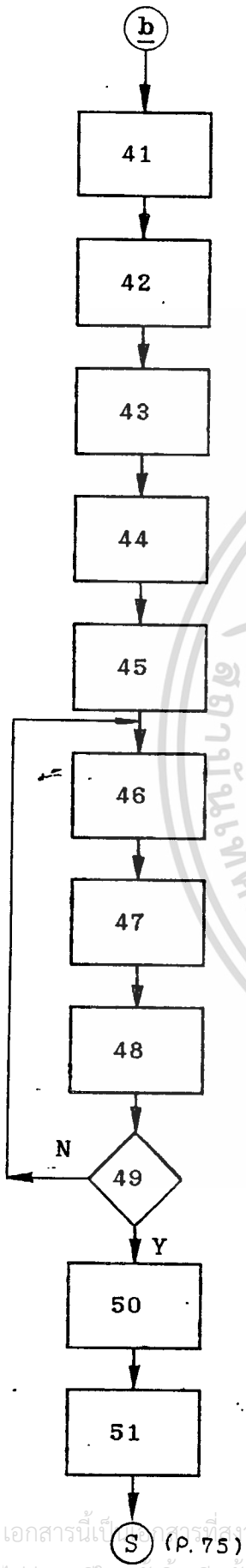
แสดงข้อความ "Can not open or creat file....."

เก็บรหัสลักษณะแฟ้มข้อมูล

เก็บค่าตัวแปรของโปรโตคอล

เก็บโปรแกรมทั้งหมด

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้



ตรวจจับข้อมูล

41 สร้างหน้าต่างของฟังก์ชั่นย่อยของฟังก์ชั่นหลัก ตรวจจับข้อมูล(Capture)

42 เตรียมค่าตัวแปรต่าง ๆ

43 เขียนหน้าจอ เพื่อเตรียมการแสดงข้อมูล

44 เตรียมค่าเวกเตอร์ของการอินเตอร์รัพท์

45 โปรแกรมค่าของอุปกรณ์บนการ์ดโปรโตคอล

46 อ่านข้อมูลจาก Bus RS-232c

47 แสดงข้อมูลบนจอภาพ

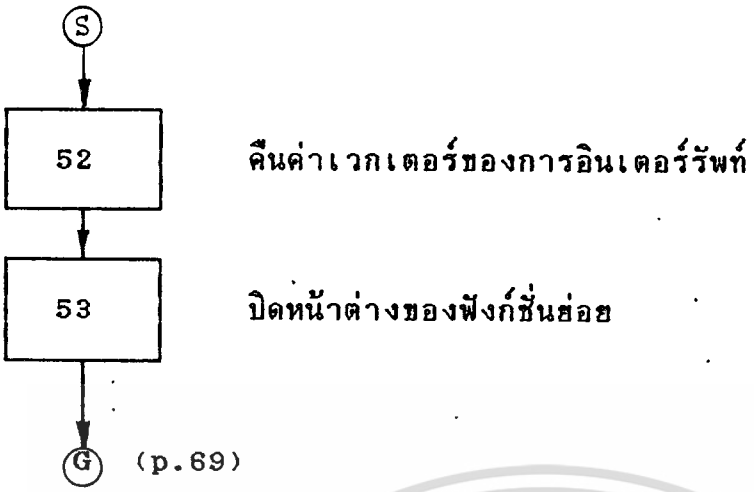
48 เขียนข้อมูลลงบน Hard Disk

49 ต้องการออกจากการทำงานหรือไม่

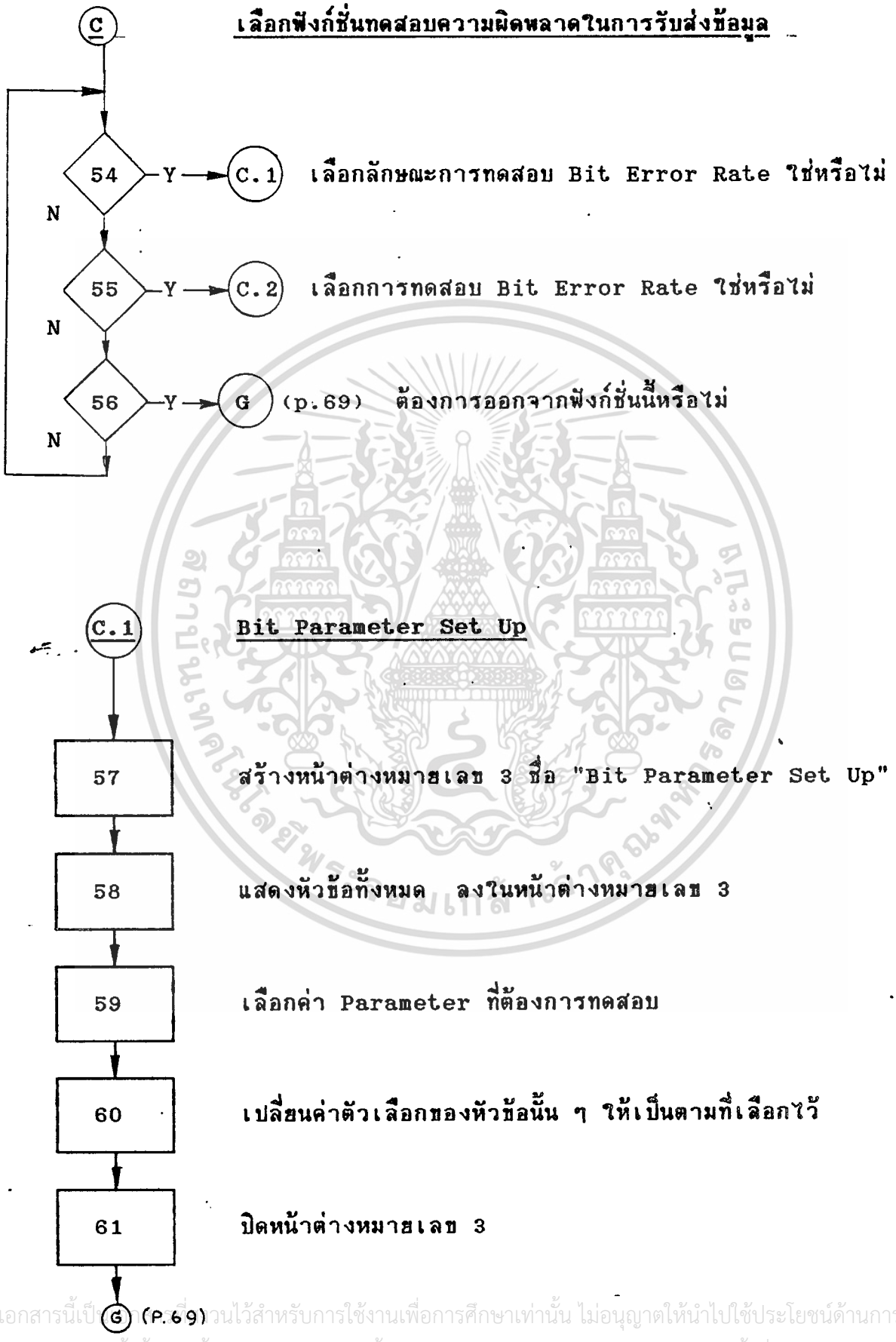
50 หยุดการแสดงข้อมูลบนจอภาพ

51 หยุดการเขียนข้อมูลลงบน Hard Disk

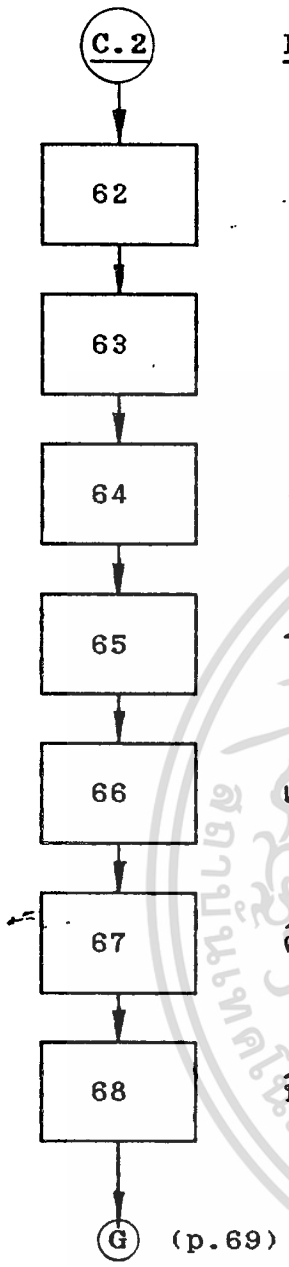
เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้



เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้



เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้



Bit Error Rate Test

62 เปิดหน้าต่างหมายเลข 3 ชื่อ "Bit Error Rate Test"

63 เขียนหน้าจอเตรียมแสดงผล

64 เตรียมค่าเวกเตอร์ของการอินเตอร์รัพท์

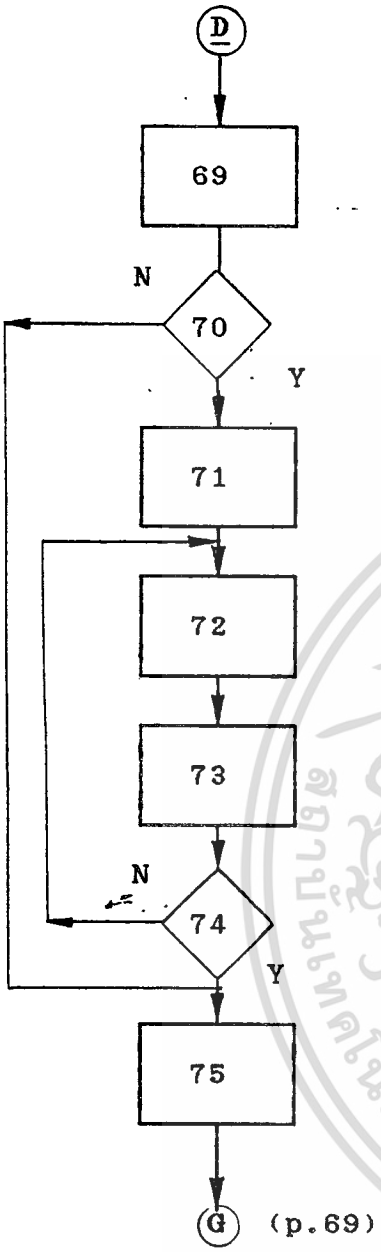
65 โปรแกรมค่าของอุปกรณ์บนการ์ดโปรโตคอล

66 แสดงผลการทดสอบ

67 คืนค่าเวกเตอร์ของการอินเตอร์รัพท์

68 ปิดหน้าต่างหมายเลข 3

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้



เลือกฟังก์ชั่นทดสอบตัวเอง

69 สร้างหน้าต่างฟังก์ย่อยของฟังก์ชั่นหลัก ทดสอบตัวเอง

70 ต้องการแสดงตัวอักษรหรือไม่

71 เปิดหน้าต่างหมายเลข 3 ชื่อ "Display Fonts"

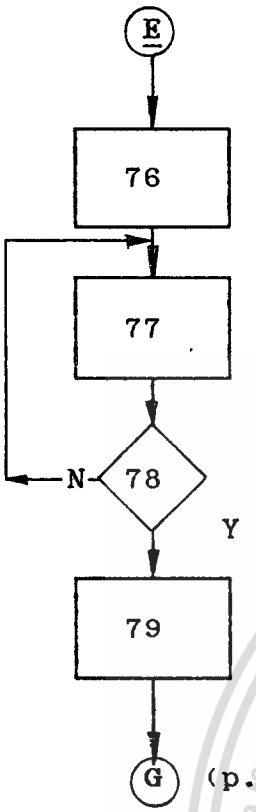
72 แสดงตัวอักษร ASCII

73 แสดงตัวอักษร HEX

74 มีการกดคีย์ เพื่อออกจากการแสดงใช่หรือไม่

75 ปิดหน้าต่างหมายเลข 3

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้



การแสดงเกี่ยวกับโปรแกรม

เปิดหน้าต่างหมายเลข 3 ชื่อ "About"

แสดงเกี่ยวกับโปรแกรม

มีการกดคีย์ เพื่อออกจากการแสดงหรือไม่

ปิดหน้าต่างหมายเลข 3

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

เอกสารอ้างอิง

1. William Schweber. *Electronic Communication Systems*, Prentice-Hall, 1991.
2. Douglas V. Hall. *Microprocessors and Interfancing Programming and Handware*, Singapore, McGraw-HILL, 1986.
3. Telebyte. *Data Communications Standards Library*, New York, Telebyte Technology Inc.,1985.
4. Joe Campbell. *C"Programmer's Guide to Serial Communications*, Indiana, Howard W.Sams & Company,1988.
5. George Sutty And Steve Blair. *Advanced Programmer's Guide to the EGA/VGA*, New York, Brady, 1988.
6. Lee Atkinson & Mark Atkinson. *Using Borlandc++ 3*, 2nd Edition, QUE Corporation, 1992.
7. Borland. *Borlandc++ Version 3.0 Programmer's Guide*, CA, Borland International,Inc, 1992.
8. Borland. *Borlandc++ Version 3.0 Library Reference*, CA, Borland International,Inc, 1992.
10. Intel. *Microcommunications Handbook* , CA , Intel Literature Sales, 1989.
11. Intel.*Microprocessor and peripheral Handbook Volume 2*, CA,Intel Literature Sales,1989.
12. ยืน ภู่วรวรรณ. *เทคโนโลยีฮาร์ดแวร์ IBM PC*, บริษัท ซีเอ็ดยูเคชั่น จำกัด
13. วีระศักดิ์ กาญจนรจิต. *ฟอนต์ภาษาไทยกับการ์ด EGA/VGA*, Computor Review ฉบับที่ 58 เดือนมิถุนายน 2532.

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

14. Larry Jordan and Bruce Churchill. *Communications and Networking for the IBM PC and Compatibles*, 4th Edition, New York, Brady Publishing, 1992.

15. Robert Jourdain. *Programmer's Problem Solver*,2nd Edition, New York, Brady Publishing, 1992.

16. Michael Holmes and Bob Flanders. *C++ Communications Utilities*, California, Ziff-Davis Prees, 1993.

17. Robert L. Hummel. *Programmers's Technical Reference: Data and Fax Communications*, California, Ziff-Davis Prees,1993.

18. อ.นรินทร์ เนาวประทีป. *การใช้งาน Z80* ,ศูนย์ภาษาคอมพิวเตอร์

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

ภาคผนวก ก.

PROGRAM MONITOR RS-232C

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```
/* ------------------------- */
/*   File :  CONST.INC       */
/* ------------------------- */

#define   END           79
#define   HOME          71
#define   PG_UP         73
#define   PG_DN         81
#define   ALT_B         48
#define   ALT_C         46
#define   ALT_D         32
#define   ALT_E         18
#define   ALT_H         35
#define   ALT_P         25
#define   ENTER         13      // keyboard return code
#define   SPACE         32
#define   ESCAPE        27
#define   DELETE        83
#define   BCKSPC        8
#define   INSERT        82
#define   ARW_UP        72
#define   ARW_LF        75
#define   ARW_RG        77
#define   ARW_DN        80
#define   FNAME_LENGTH  50

#define   WIN_WIDTH     37
#define   NORMTEXT      0x07
#define   FUNC_BAR      0x70

// chanal of 8274
#define   CHNAL_A       0x00    // channal A of 8274
#define   CHNAL_B       0x01    // channal B of 8274

// 8274 addressing
#define   DATA_8274_A   0x0300  // data on channal A of 8274
#define   DATA_8274_B   0x0301  // data on channal B of 8274
#define   COMM_8274_A   0x0302  // command & status port A
#define   COMM_8274_B   0x0303  // command & status port B

// 8254 addressing
#define   COUNT0_8254   0x0304  // counter 0
#define   COUNT1_8254   0x0305  // counter 1
#define   COUNT2_8254   0x0306  // counter 2
#define   CONTROL_8254  0x0307  // control word

// 8255 addressing
#define   PORT_A_8255   0x0308  // port a
#define   PORT_B_8255   0x0309  // port b
#define   PORT_C_8255   0x030A  // port c
#define   CONTROL_8255  0x030B  // control word

// 8259 operating command word
#define   OCW1_8259     0x0021  // ocw 1
#define   OCW2_8259     0x0020  // ocw 2

// all bit in all write register of 8274
#define   WR0_ABORT     0x08    // send abort
#define   WR0_RSTEXT    0x10    // reset ext/status interrupt
#define   WR0_CHNRST    0x18    // channal reset
#define   WR0_NXTRX     0x20    // enable interrupt on next Rx char.
#define   WR0_INTPND    0x28    // reset Tx interrupt or DMA pending
```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```
#define   WR0_ERRRST    0x30    // error reset
#define   WR0_ENDINT    0x38    // end of interrupt
#define   WR0_RxCRC     0x40    // reset Rx CRC checker
#define   WR0_TxGEN     0x80    // reset Tx CRC generator
#define   WR0_TxEOM     0xC0    // reset tx underrun/EOM latch

#define   WR3_RxENA     0x01    // Rx enable
#define   WR3_SYNLD     0x02    // sync char. load inhibit
#define   WR3_ADDR      0x04    // address search mode (in SDLC)
#define   WR3_RxCRC     0x08    // Rx CRC enable
#define   WR3_HUNT      0x10    // hunt mode
#define   WR3_AUTO      0x20    // auto enable

#define   WR4_PTYENA    0x01    // odd parity
#define   WR4_EVEN      0x02    // even parity
#define   WR4_8SYNC     0x00    // 8 bits sync. char.
#define   WR4_16SYNC    0x10    // 16 bits sync. char.
#define   WR4_HDLCMD    0x20    // HDLC or SDLC mode
#define   WR4_EXSYNC    0x30    // external sync. mode

#define   WR5_TxCRC     0x01    // Tx CRC enable
#define   WR5_RTS       0x02    // RTS lead
#define   WR5_CRC16     0x04    // 0=CCITT-CRC,1=CRC 16
#define   WR5_TxENA     0x08    // Tx enable
#define   WR5_SDBRK     0x10    // send break
#define   WR5_DTR       0x80    // DTR lead

#define   RR0_CD        0x08    // CD status
#define   RR0_SYNC      0x10    // hunt/sync detect
#define   RR0_CTS       0x20    // CTS status
#define   RR0_TxUDR     0x40    // Tx unoerrun
#define   RR0_BREAK     0x80    // break detected

#define   RR1_PARIRY    0x10    // parity error
#define   RR1_CRCFRM    0x20    // frame or CRC error
#define   RR1_EOF       0x80    // end of frame

// send flag
#define   FLAG_CRC_CAL  0x01    // start calculate CRC
#define   FLAG_STP_CRC  0x02    // stop calculate CRC
#define   FLAG_STR_CRC  0x04    // CRC cal. already
#define   FLAG_SND_INV  0x20    // send invert data
#define   FLAG_ENA_CRC  0x40    // enable calculate CRC

// receive flag
#define   FLAG_CAL_CRC  0x01    // start calculate CRC
#define   FLAG_STP_CRC  0x02    // stop calculate CRC
#define   FLAG_STR_CRC  0x04    // CRC cal. already
#define   FLAG_ENA_CRC  0x40    // enable calculate CRC

// miscellenous flag
#define   MISC_DTE      0x01    // data from DCE
#define   MISC_LEAD     0x02    // lead or data change
#define   MISC_TYPE     0x0C    // mask of type of data
#define   MISC_PARITY   0x10    // parity error
#define   MISC_HILGHT   0x20    // hilight on data or status
#define   MISC_FRAME    0x40    // frame or CRC error
#define   MISC_BREAK    0x08    // break/abort detected
#define   MISC_DATA     0x00    // lead status change
#define   MISC_CRC      0x04    // CRC charactor
```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```
// lead mapping
#define   LEAD_DTR      0x04    // WR5 bit D7
#define   LEAD_DSR      0x01
#define   LEAD_CTS      0x10    // RR0 bit D5
#define   LEAD_CD       0x08    // RR0 bit D3
#define   LEAD_RTS      0x02    // WR5 bit D1

// lead intrrupt
#define   L_DSR         0x01
#define   L_RTS         0x02
#define   L_DTR         0x04
#define   L_CD          0x08
#define   L_CTS         0x10

// intrrupt type
#define   INT_EXT       0x40
#define   INT_8274      0x20
#define   INT_1ms       0x00

#define   CRC_ON_CHAR   2       // maximum charactors checking to start calc. CRC
#define   CRC_OFF_CHAR  3       // maximum charactors checking to stop calc. CRC
#define   DSK_BUF_SIZE  15      // number of receive data stroed before kept to disk
#define   MAX_PRG_BLCK  32      // maximum block in program
#define   PRG_BUF_SIZE  1000    // size of buffer size
#define   DAT_BUF_SIZE  1500    // size of dat buffer
#define   LOD_DAT_BLCK  750     // maximum data block

enum {
        REGISTER_0,             // register number
        REGISTER_1,
        REGISTER_2,
        REGISTER_3,
        REGISTER_4,
        REGISTER_5,
        REGISTER_6,
        REGISTER_7,
};

enum {
        KEY_F1=59,              // function key return code of keyboard
        KEY_F2,
        KEY_F3,
        KEY_F4,
        KEY_F5,
        KEY_F6,
        KEY_F7,
        KEY_F8,
        KEY_F9,
        KEY_F10,
};
enum {                          // window name
        WINDOW1,
        WINDOW2,
        WINDOW3,
        WINDOW4,
        WINDOW5,
        WINDOW6,
        MAX_WINDOW,
};
```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```
enum {                          // main function name
        FUNC_SETUP,
        FUNC_RUN,
        FUNC_BERT,
        FUNC_DIAG,
        FUNC_QUIT,
        FUNC_HELP,
        MAX_FUNC,
};

enum {                          // function of setup protocol parameter
        SET_SETUP,
        SET_LOAD,
        SET_SAVE,
        MAX_SET,
};

enum {                          // BERT function
        BERT_SETUP,
        BERT_RUN,
        MAX_BERT,
};

enum {                          // diagnostic function
        DIAG_SCREEN,
        MAX_DIAG,
};

enum {                          // attribute of each window
        MAIN_ATTR,
        PULL_ATTR,
        EDIT_ATTR,
        SIML_ATTR,
        COMM_ATTR,
        VERF_ATTR,
        EROR_ATTR,
        ABUT_ATTR,
        MAX_ATTR,
};

enum {                          // style of window frame
        STYLE1,
        STYLE2,
        STYLE3,
        MAX_STYLE,
};

enum {
        CONV_EBCDIC,            // code conversion
        CONV_TRANSC,
        MAX_CONV_CODE,
};

// -------------------- PARAMETER OF PROTOCOL ---------------------
enum {
        SETUP_PROC,
        SETUP_CODE,
        SETUP_MODE,
        SETUP_PRTY,
        SETUP_STOP,
        SETUP_ERROR,
        SETUP_CLOCK,
```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น ห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```
        SETUP_SPEED,
        SETUP_DISP,
        SETUP_ORDER,
        SETUP_SENSE,
        SETUP_ADDR,
        SETUP_SYNC,
        SETUP_CRCON,
        SETUP_CRCOFF,
        MAX_SETUP,
};

enum {
        PROC_ASYNC,
        PROC_SYNC,
        PROC_BSC,
        PROC_HDLC,
        PROC_SDLC,
        MAX_PROC,
};

enum {
        CODE_EBCDIC,
        CODE_ASCII8,
        CODE_ASCII7,
        MAX_CODE,
};

enum {
        MODE_NRZ,
        MODE_NRZI,
        MAX_MODE,
};

enum {
        PARITY_NONE,
        PARITY_ODD,
        PARITY_EVEN,
        MAX_PARITY,
};

enum {
        STOP_10,
        STOP_15,
        STOP_20,
        MAX_STOPBIT,            // async stop bit
};

enum {
        ERROR_CCITT,
        ERROR_CRC16,
        ERROR_NONE,
        MAX_ERROR,
};

enum {
        CLK_INTERNAL,
        CLK_EXTERNAL,
        MAX_CLKSCR,             // clock source
};

enum {
        SPEED_50,
```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```
        SPEED_75,
        SPEED_110,
        SPEED_134,
        SPEED_150,
        SPEED_200,
        SPEED_300,
        SPEED_600,
        SPEED_1200,
        SPEED_1800,
        SPEED_2000,
        SPEED_2400,
        SPEED_3200,
        SPEED_3600,
        SPEED_4800,
        SPEED_7200,
        SPEED_9600,
        SPEED_12000,
        SPEED_14400,
        SPEED_16000,
        SPEED_19200,
        MAX_SPEED,              // speed selectable
};

enum {
        DISP_DTE,               // DTE only
        DISP_DCE,               // DCE only
        DISP_DATA,              // data only
        DISP_STAT,              // data & status
        MAX_DISPMD,
};

enum {
        BITODR_MSB,
        BITODR_LSB,
        MAX_BITORD,
};

enum {
        BITSND_NORMAL,
        BITSND_INVERT,
        MAX_BITSND,
};

typedef struct {
                char    border;         // border attribute
                char    header;         // header attribute
                char    norm_tx;        // normal text attribute
                char    hili_tx;        // hilight text attribute
                char    hot_key;        // hotkey attribute
        } WIN_ATTR;

typedef struct {
                char    lft_top;        // left top conner
                char    lft_bot;        // left bottom conner
                char    rgh_top;        // right top conner
                char    rgh_bot;        // right bottom conner
                char    lft_ver;        // left vertical line
                char    rgh_ver;        // right vertical line
                char    top_hor;        // left horizontal line
                char    bot_hor;        // right horizontal line
        } WIN_STYLE;

typedef struct {
```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```
                char    top;            // top of window
                char    left;           // left side of window
                char    right;          // right side of window
                char    bottom;         // bottom of winoow
                char    *head;          // pointer to header of window
                char    *scr;           // buffer to store last screen
                WIN_ATTR *clr;          // color of window
                WIN_STYLE *sty;         // border style of window
        } WIN_STRUC;

typedef struct {
                unsigned  char  val;                    // selected value
                signed    char  sel[MAX_SPEED+1];       // 1:selectable, 0:unselectable
        } SETUP_ARRAY;

typedef struct {
                unsigned  char  proc;                   // selected protocol
                unsigned  char  max[MAX_PROC];          // maximum setup parameter
                signed    char  **nm[MAX_SETUP+1];      // array of pointer to choice name
                SETUP_ARRAY     *st[MAX_PROC];          // protocol parameter
        } SETUP;

typedef struct {
                char    x;
                char    y;
                char    *name;
        } MNBAR_STRUC;

typedef struct {
                char    count;
                char    xbar;
                char    width,act;
                char    *list[5];
                char    *item;
        } MENU_STRUC;

typedef struct {
                unsigned char   ascii;  // to ascii code
                unsigned char   attr;   // attribute to display
        } ASCII_TABLE;

/*  Send and Receive data buffer
                misc                    lead status
        bit 0: 0=DCE,1=DTE              bit 0: DSR
        bit 1: 0=data,1=lead            bit 1: RTS
        bit2,3: 00=data,01=CRC          bit 2: DTR
                10=start,11=stop frame  bit 3: CD
        bit 4: 1=parity error           bit 4: CTS
        bit 5: 1=hilight                bit 5:  -
        bit 6: 1=CRC/frame error        bit 6:  -
        bit 7: 1=break/abort            bit 7:  - */


typedef struct {
                unsigned char   misc;
                unsigned char   data;
                unsigned char   lead;
                unsigned int    timer[3];
                unsigned int    counter[3];
        } DATA_STRUC;
typedef struct {
```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```
                char    drv[3];         // drive name
                char    dir[66];        // directory name
                char    fil[9];         // filename
                char    ext[5];         // file extended name
        } PATH_STRUC;

struct  list_table {
                long                    ptr;    // start block position
                unsigned int            len;    // number of valid data in block
                struct list_table       *next;  // points to next block
                struct list_table       *prior; // points to prior block
        };
typedef struct list_table       LIST_STRUC;     // file list table for analysis
```



เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```
/*-------------------------*/
/*  File :  MONITOR.INC    */
/*-------------------------*/

#ifndef __MEDIUM__
#error  This program supports medium memory model only.
#endif

MENU_STRUC  Menu[MAX_FUNC] = {

//" Protocol   Cature   BERT   Diagnostic   Exit       Help "

                MAX_SET,
                6,9,0,
                " Setup  ",  " Load   ",
                " Save   ",  "",
                "",
                "",
                "Protocol",

                0,19,8,0,
                "",          "",
                "",          "",
                "",          "",
                "",
                "Capture",

                MAX_BERT,
                30,8,0,
                " Setup ",   " Run   ",
                "",          "",
                "",
                "",
                "BERT",

                MAX_DIAG,
                39,9,0,
                " Fonts  ",  "",
                "",          "",
                "",
                "",
                "Diagnostic",

                2,53,6,0,
                " Yes ",     " No ",
                "",          "",
                "",
                "",
                "Exit",

                1,70,8,0,
                " About ",   "",
                "",          "",
                "",
                "",
                "Help",
        };

int     Fh;                             // file handle
unsigned char   Disp_mode;              // current display mode
char    *Font_name[2][2] = {            // font file
                { "Master14.fnt", "Hex14.fnt" },
                { "Master16.fnt", "Hex16.fnt" }
        };

WIN_ATTR  Win_attr[MAX_ATTR] = {                // window attribute
        0x03,   0x13,   0x07,   0x17,   0x74,   // main window
        0x70,   0x70,   0x70,   0x17,   0x74,   // pulldown
```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น ทั้งนี้ห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```
        0x73,   0x73,   0x70,   0x17,   0x73,   // edit program
        0x17,   0x71,   0x07,   0x70,   0x03,   // run program
        0x17,   0x17,   0x17,   0x70,   0x10,   // command line
        0x70,   0x70,   0x70,   0x17,   0x74,   // verify block
        0x47,   0x47,   0x40,   0x00,   0x47,   // error block
        0x73,   0x73,   0x70,   0x72,   0x25,   // about
    //border, header, norm_tx, hili_tx;hot_key
    };

WIN_STYLE  Win_style[MAX_STYLE] = {
        218,192,191,217,179,179,196,196,        // all single lines
        213,192,184,217,179,179,205,196,        // double at top hor.only
        201,200,187,188,186,186,205,205,        // all double lines
};

DATA_STRUC      *Data_buff;                     // receive data buffer
unsigned char   *Prog_buff;                     // program buffer pointer
LIST_STRUC      *Str_list_tbl;                  // start list table of data file
LIST_STRUC      *Cur_list_tbl;                  // current list table of data file
LIST_STRUC      *End_list_tbl;                  // last list table of data file
WIN_STRUC       Win[MAX_WINDOW];

extern  char far        *Vid_buff;              // video buffer
extern  unsigned        _heaplen = 0;
extern  unsigned        _stklen = 10000;        //stack size

/* -------------------- Monitor.c --------------------*/

void    Quit(void);
void    Write_func(char);
void    Restore_all(char);
void    Hiligh_func(char);
void    Get_function(void);
char    Do_func(char,char);
void    Write_header(char);
void    Init_hardware(void);
char    Init_variable(void);
char    Pulldown(char,char*);
void    Unpulldown(char,char);
char    Init_font(char*,char);
void    Make_pulldown(char,char*);
void    Write_pulldown(char,char);
void    Hiligh_pulldown(char,char);
char    Load_char_gen(char*,char*,char,cnar);
void    Make_window(char,char,char,char,char*,WIN_ATTR*,WIN_STYLE*,char*,char);
void    Write_head(char);
void    Make_win(char,char,char,char,char*,WIN_ATTR*,WIN_STYLE*,char*,char);
void    about(void);

void    interrupt (*Old_int0D)();

extern  char    Cursor_on(void);
extern  char    Cursor_off(void);
extern  char    Getch(void);
extern  void    Ungetch(char);
extern  void    Proc_setup(void);
extern  void    Cputch(char,char,char,char);

extern  void    Load(char,char);
extern  void    Save(char,char);
extern  void    Print(char,char);
```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งไม่อนุญาตให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```
extern  void    Run_program(void);
extern  void    Edit_setup(char);

extern  void    Brt_Setup(void);
extern  void    Brt_run(void);

extern  void    Diag_screen(void);

extern  void    interrupt New_int0D(void);

// assembly routine

extern  SETUP   Prot_setup;
extern  char    Check_EGA_VGA(void);
extern  void    Set_mode(char,char);
extern  void    EGA_512_char(char,char);
extern  void    Restore_default_font(char);
extern  void    Load_default_char(char,char);
extern  void    Load_custom_char(char*,char,char);
extern  void    Delete_list_table(LIST_STRUC*);
extern  void    Gen_sound(unsigned int,unsigned int);
extern  void    Scroll_up(char,char,char,char,char,char);
```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```
/* ------------------------- */
/*   File :  MONITOR.C       */
/* ------------------------- */

#include  <dos.h>
#include  <io.h>
#include  <conio.h>
#include  <ctype.h>
#include  <alloc.h>
#include  <stdio.h>
#include  <stdlib.h>
#include  <string.h>
#include  "constant.inc"
#include  "monitor.inc"

void main(char argc,char *argv[])
{
        char    n;
        char    path[50];

        if (argc>1) {
            strcpy(path,argv[1]);       // get path
            n = strlen(path);
            if (path[n] != '\\' || path [n] != ':')
                strcat(path,"\\");
        }
        else path[0] = 0;

        if ((n=Check_EGA_VGA()) < 2) {
            Vid_buff = (char far*) 0xB6000000L;
            Set_mode(3,0);                      // set color text moce
            if (!Init_font(path,n))
                exit(3);
        }
        else {
                Vid_buff = (char far*) 0xB0000000L;
                cprintf("\n\rDisplay adapter card is not EGA or VGA.\n\r");
        }
        if (!Init_variable())
            exit(4);
        Init_hardware();
        Gen_sound(20000,5);
        Cursor_off();
        Get_function();
        Restore_all(n);
        Cursor_on();
        exit(0);
}

char Init_font(char *path,char card_id)
{
        char    *Font_buff;

        textattr(NORMTEXT);
        if ((Font_buff=(char*) malloc (16*256)) == NULL) {
            cprintf("Sufficeint enough memory\n\r");
            return 0;
        }
        if (!Load_char_gen(Font_buff,path,card_id,0))   // load master font
            return 0;

        if (!Load_char_gen(Font_buff,path,card_id,1))   // load hex font
```

เอกสารนี้เป็นเอกสารที่สงวนลิขสิทธิ์ ไม่อนุญาตให้ผู้ใดนำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```
            return 0;

        free(Font_buff);
        EGA_512_char(0,1);
        return 1;
}

char Init_variable(void)
{
        Prog_buff = (unsigned char*) calloc (1,PRG_BUF_SIZE);
        Data_buff = (DATA_STRUC*) calloc (1,sizeof(DATA_STRUC)*DAT_BUF_SIZE);

        if (!Prog_buff || !Data_buff) {
            cprintf("Sufficeint enough memory\n\r");
            return 0;
        }
        Fh = 0;                 // file does not open yet
        Str_list_tbl = 0;                       // start of data on file
        End_list_tbl = 0;                       // current file pointer
        Disp_mode = (Prot_setup.st[Prot_setup.proc]+SETUP_DISP)->val;
        cprintf("Memory avaliable %u %ubyte(s)\n\r",coreleft(),farcoreleft());
        return 1;
}

void Init_hardware(void)
{
        disable();
        outportb(OCW1_8259,inportb(OCW1_8259) | 0x28);  // disable IQR3,5 of 8259
        Old_int0D = getvect(0x0D);
        setvect(0x0D,New_int0D);
        enable();

              // initial 8254: baud rate generate & timer trig
        outportb(CONTROL_8254,0x36);            // counter 0: mode 3 square wave
        outportb(COUNT0_8254,0xC6);             // devide by 1990
        outportb(COUNT0_8254,0x07);             // for 1mS timer trig

              // initial 8255: control
        outportb(CONTROL_8255,0x89);            // A, B outport, C input
        outportb(PORT_A_8255,0);                // open all
        outportb(PORT_B_8255,0);
}

void Restore_all(char n)
{
        disable();
        setvect(0x0D,Old_int0D);
        outportb(OCW1_8259,inportb(OCW1_8259) | 0x28);  // disable IQR3,5 of 8259
        enable();

        Restore_default_font(n);        // return to normal display mode
        EGA_512_char(0,0);              // reset to normal
        if (Fh) {
            _close(Fh);
            Delete_list_table(Str_list_tbl);
        }
}

char Load_char_gen(char *buff,char *path,char card_id,char num)
{
        FILE    *fp;
        char    temp[10];
```

เอกสารนี้เป็นสิ่งที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```
        char    Fpath[50];
        register unsigned  i,high;

        strcpy(Fpath,path);
        strcat(Fpath,Font_name[card_id][num]);          // path and file name
        if ((fp=fopen(Fpath,"rb+"))==NULL) {
            cprintf("Attempt to load %s.\n\r",Fpath);
            cprintf("Type: monitor [path of font]\n\r");
            return 0;
        }
        fgets(temp,10,fp);                      // Get key word
        high = fgetc(fp);                       // get charactor high
        if (!strcmp(temp,"EGA. Font")) {
            for (i=0 ;i<high*256;i++)
                *(buff+i) = getc(fp);
            Load_custom_char(buff,high,num);
            fclose(fp);
            return 1;
        }
        cprintf("file %s is not found.",path);  // font file error
        fclose(fp);
        return 0;
}

void Make_window(char left,char top,char right,char bottom,char *head,
                 WIN_ATTR *clr,WIN_STYLE *sty,char *scr,char num)
{
        register char   i;

        Win[num].sty = sty;             // border style
        Win[num].clr = clr;             // window color
        Win[num].scr = scr;             // pointer to screen buffer
        Win[num].top = top;
        Win[num].head = head;
        Win[num].left = left;
        Win[num].right = right;
        Win[num].bottom = bottom;       // window size

        if (num)
            gettext(left,top,right,bottom,scr); // save used screen
        Scroll_up(left,top,right,bottom,clr->norm_tx,0);
        top = Win[num].top;
        left = Win[num].left;
        right = Win[num].right;
        bottom = Win[num].bottom;
        textattr(Win[num].clr->border);

        for (i=left+1;i<right;i++) {
            gotoxy(i,bottom);
            cprintf("%c",Win[num].sty->bot_hor);
        }
        for (i=top+1;i<bottom;i++) {
            gotoxy(left,i); cprintf("%c",Win[num].sty->lft_ver);
            gotoxy(right,i); cprintf("%c",Win[num].sty->rgh_ver);
        }
        gotoxy(left,top); cprintf("%c",Win[num].sty->lft_top);
        gotoxy(right,top); cprintf("%c",Win[num].sty->rgh_top);
        gotoxy(left,bottom); cprintf("%c",Win[num].sty->lft_bot);
        gotoxy(right,bottom); cprintf("%c",Win[num].sty->rgh_bot);
        Write_header(num);
}
```

เอกสารนี้เป็นเอกสาร ให้นำไปใช้ประโยชน์ด้านการค้า
ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```
void Write_header(char num)
{
        register unsigned char  i;

        for (i=Win[num].left+1;i<Win[num].right;i++)
            Cputch(i,Win[num].top,Win[num].clr->border,Win[num].sty->top_hor);
        i = (Win[num].right-Win[num].left-strlen(Win[num].head)) >> 1;
        textattr(Win[num].clr->header);
        gotoxy(Win[num].left+i,Win[num].top);
        cprintf("%s",Win[num].head);
}

void Get_function(void)
{
        char            *scrn,fc;
        register char   i,of,pop;

        textattr(NORMTEXT);
        clrscr();
        Make_window(1,2,80,24," RS232 MONITOR ",&Win_attr[MAIN_ATTR],
                                &Win_style[STYLE3],"",WINDOW1);
        textattr(FUNC_BAR);
        gotoxy(1,1);
        for (i=1;i<=80;i++)
            cprintf("%c",0x20);                 // menu bar
        for (i=0;i<MAX_FUNC;i++)
            Write_func(i);

        scrn = (char*)malloc(160);
        textattr(Win[WINDOW1].clr->hili_tx);
        gotoxy(Menu[FUNC_SETUP].xbar,1);
        cprintf("%s",Menu[FUNC_SETUP].item);
        about();

        for (fc=of=pop=0;;) {
            of=fc;
            switch(toupper(Getch())) {
                case 0:
                    switch(Getch()) {
                        case ALT_B:
                            Ungetch('B');
                            pop = 0;
                            break;

                        case ALT_D:
                            Ungetch('D');
                            pop = 0;
                            break;

                        case ALT_E:
                            Ungetch('E');
                            pop = 0;
                            break;

                        case ALT_H:
                            Ungetch('H');
                            pop = 0;
                            break;

                        case ALT_P:
                            Ungetch('P');
                            pop = 0;
```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```
            break;

        case ALT_C:
            Ungetch('C');
            pop = 0;
            break;

        case ARW_LF:
            if (!fc)
                fc = MAX_FUNC-1;
            else --fc;
            break;

        case ARW_RG:
            if (fc==MAX_FUNC-1)
                fc=0;
            else ++fc;
            break;

        case ARW_DN:
            pop=1;
            break;
    }
    break;

case ENTER:
    if (fc == 1){
      Run_program();
    }
    pop=1;
    break;

case 'P':                       // setup parameter
    fc=0;
    pop=1;
    break;

case 'C':                       // capture
    Run_program();
    fc=1;
    pop=1;
    break;

case 'B':                       // BERT
    fc=2;
    pop=1;
    break;

case 'D':                       // diagnostics
    fc=3;
    pop=1;
    break;

case 'E':                       // exit program
    fc=4;
    pop=1;
    break;

case 'H':                       // show help
    fc=5;
    pop=1;
    break;
```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```
            }
            if (fc!=of) {
                Write_func(of);
                Hiligh_func(fc);
            }
            if (pop) {
                if ((pop=Pulldown(fc,scrn)) == -1)
                return;
            }
        }
}

void Write_func(char fc)
{
        gotoxy(Menu[fc].xbar,1);
        textattr(Win[WINDOW1].clr->hot_key);
        cprintf("%1c",*Menu[fc].item);
        textattr(FUNC_BAR);
        cprintf("%s",Menu[fc].item+1);
}

void Hiligh_func(char fc)
{
        textattr(Win[WINDOW1].clr->nili_tx);
        gotoxy(Menu[fc].xbar,1);
        cprintf("%s",Menu[fc].item);
}

char Pulldown(char fc,char *scrn)
{
        register char   c,i,fn,ofn,sel;

        if (!Menu[fc].count)
          return 1;

        Make_pulldown(fc,scrn);
        fn = Menu[fc].act;              // previous selected function
        for (;;) {
            sel = 0;
            ofn = fn;
            switch((c=Getch())) {
                case 0:
                    switch(c=Getch()) {
                        case ARW_UP:
                            if (Menu[fc].count) {
                                if (!fn)
                                    fn = Menu[fc].count-1;
                                else --fn;
                            }
                            break;

                        case ARW_DN:
                            if (Menu[fc].count) {
                                if (fn == Menu[fc].count-1)
                                    fn=0;
                                else ++fn;
                            }
                            break;

                        case ALT_B:
                            Unpulldown(fc,fn);
                            Ungetch('B');
```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```
                return 1;

            case ALT_C:
                Unpulldown(fc,fn);
                Ungetch('C');
                return 1;

            case ALT_D:
                Unpulldown(fc,fn);
                Ungetch('D');
                return 1;

            case ALT_E:
                Unpulldown(fc,fn);
                Ungetch('E');
                return 1;

            case ALT_H:
                Unpulldown(fc,fn);
                Ungetch('H');
                return 1;

            case ALT_P:
                Unpulldown(fc,fn);
                Ungetch('P');
                return 1;

            case ARW_LF:
            case ARW_RG:
                Unpulldown(fc,fn);
                Ungetch(c);
                Ungetch(0);
                return 1;           // to next pulldown fuction
            }
            break;

        case ESCAPE:
            Unpulldown(fc,fn);
            return 0;

        case ENTER:
            if ((c=Do_func(fc,fn)) != 1)
                return c;
            break;

        default:
            c = toupper(c);
            for(i=0;i<Menu[fc].count;i++) {
                if (c == *(Menu[fc].list[i]+1)) {
                    sel = 1;
                    fn = i;
                    break;
                }
            }
            break;
    }
    if (fn != ofn) {
        Write_pulldown(fc,ofn);
        Hiligh_pulldown(fc,fn);
    }
    if (sel) {
        if ((c=Do_func(fc,fn)) != 1)
```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```
                return c;
            }
/       }
}

void Write_pulldown(char fc,char fn)
{
        gotoxy(Win[WINDOW2].left+1,3+fn);
        textattr(Win[WINDOW2].clr->norm_tx);
        cprintf("%s",Menu[fc].list[fn]);              // normal
        gotoxy(Win[WINDOW2].left+2,3+fn);
        textattr(Win[WINDOW2].clr->hot_key);
        cprintf("%1c",*(Menu[fc].list[fn]+1));        // hotkey
}

void Hiligh_pulldown(char fc,char fn)
{
        textattr(Win[WINDOW2].clr->hili_tx);
        gotoxy(Win[WINDOW2].left+1,3+fn);
        cprintf("%s",Menu[fc].list[fn]);
}

void Make_pulldown(char fc,char *scrn)
{
        register char   i,fn,lft,rgh,bot;

        fn = Menu[fc].act;
        lft = Menu[fc].xbar-1;
        bot = Menu[fc].count+3;
        rgh = Menu[fc].width+lft;
        Make_window(lft,2,rgh,bot,"",&Win_attr[PULL_ATTR],
                &Win_style[STYLE1],scrn,WINDOW2);
        for (i=0;i<Menu[fc].count;i++)
            Write_pulldown(fc,i);
        Hiligh_pulldown(fc,fn);
}

void Unpulldown(char fc,char fn)
{
      Menu[fc].act = fn;              // save previous selected function
        puttext(Win[WINDOW2].left,Win[WINDOW2].top,
                Win[WINDOW2].right,Win[WINDOW2].bottom,Win[WINDOW2].scr);
}

char Do_func(char fc,char fn)
{
        switch (fc) {
            case FUNC_SETUP:
                switch (fn) {
                    case SET_SETUP:
                        Edit_setup(0);
                        break;

                    case SET_LOAD:
                        Load(fc,fn);
                        break;

                    case SET_SAVE:
                        Save(fc,fn);
                        break;
                }
                break;
```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```
            case FUNC_BERT:
                switch (fn) {
                    case BERT_SETUP:
                        break;

                    case BERT_RUN:
                        break;
                }
                break;

            case FUNC_DIAG:
                switch (fn) {
                    case DIAG_SCREEN:
                        break;
                }
                break;

            case FUNC_QUIT:
                if (!fn) {
                    Quit();                     // yes
                    return -1;
                }
                else {
                    Unpulldown(fc,fn);          // no
                    return 0;                   // unpulldown
                }
            case FUNC_HELP:
                    about();
                    break;
        }
        return 1;                               // pulldown continue
}

void Quit(void)
{
        free(Prog_buff);
        free(Win[WINDOW2].scr);
        textattr(NORMTEXT);
        clrscr();
}

void about(void)                        //Show about on screen
{
        char left,top,right,bottom;
        char *scr;
        int  x,y;

        scr = (char*) malloc (1600);
        if (!scr) {
            cprintf("Not enough memory.");
            return;
        }
        Make_win(21,4,59,22," about ",&Win_attr[ABUT_ATTR],
                                    &Win_style[STYLE3],scr,WINDOW6);
        textattr(Win[WINDOW6].clr->norm_tx);
        gotoxy(33, 6); cprintf("RS-232C MONITORS");
        gotoxy(32, 7); cprintf("Copyright (c),1993");
        gotoxy(26, 9); cprintf("Develope by : ");
        gotoxy(30,10); cprintf("Mr. Ronruechai Jamfoong");
        gotoxy(30,11); cprintf("Mr. Krailerd Chungmarisakul");
        gotoxy(26,13); cprintf("Adviser :");
```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```
        gotoxy(30,14); cprintf("Mr. Paisan Sithiyopasakul  ");
        gotoxy(26,17); cprintf("King Mongkut's Institute of");
        gotoxy(26,18); cprintf("   Technology Ladkrabang   ");
        gotoxy(44,20); cprintf("%c",220);
        textattr(Win[WINDOW6].clr->hili_tx);
        for(x=36;x<44;x++) {
            gotoxy(x,20);
            cprintf("%c",219);
        }
        textattr(Win[WINDOW6].clr->hot_key);
        gotoxy(39,20); cprintf("O");
        gotoxy(40,20); cprintf("K");
        textattr(0x70);
        for(x=37;x<45;x++) {                    //background hili
            gotoxy(x,21);
            cprintf("%c",223);
        }
        x = getch();
        if(!x)
           getch();
        puttext(Win[WINDOW6].left,Win[WINDOW6].top,Win[WINDOW6].right+2,
               Win[WINDOW6].bottom+1,scr);     // restore used screen

}
void Make_win(char left,char top,char right,char bottom,char *head,
              WIN_ATTR *clr,WIN_STYLE *sty,char *scr,char num)
{
        register char   i, x, y;

        Win[num].sty = sty;             // border style
        Win[num].clr = clr;             // window color
        Win[num].scr = scr;             // pointer to screen buffer
        Win[num].top = top;
        Win[num].head = head;
        Win[num].left = left;
        Win[num].right = right;
        Win[num].bottom = bottom;       // windcw size

        if (num)
            gettext(left,top,right+2,bottom+1,scr);     // save used screen
        Scroll_up(left,top,right,bottom,clr->norm_tx,0);

        textattr(Win[num].clr->border);
        for (i=left+1;i<right;i++) {
            gotoxy(i,bottom);
            cprintf("%c",Win[num].sty->bot_nor);
        }
        for (i=top+1;i<bottom;i++) {
            gotoxy(left,i); cprintf("%c",Win[num].sty->lft_ver);
            gotoxy(right,i); cprintf("%c",Win[num].sty->rgh_ver);
        }
        gotoxy(left,top); cprintf("%c",Win[num].sty->lft_top);
        gotoxy(right,top); cprintf("%c",Win[num].sty->rgh_top);
        gotoxy(left,bottom); cprintf("%c",Win[num].sty->lft_bot);
        gotoxy(right,bottom); cprintf("%c",Win[num].sty->rgh_bot);

        textcolor(BLACK);
        for(x=left+2;x<=right+2;x++) {                  // bottom background
            gotoxy(x,bottom+1);
            cprintf("%c",219);                          //put char
        }
        for(y=top+1;y<=bottom;y++) {                    // right background
```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งไม่อนุญาตให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```
            gotoxy(right+1,y);
            cprintf("%c",219);                          //put char
            gotoxy(right+2,y);
            cprintf("%c",219);
        }
        Write_head(num);
}
void Write_head(char num)
{
        register unsigned char  i;

        for (i=Win[num].left+1;i<Win[num].right;i++)
            Cputch(i,Win[num].top,Win[num].clr->border,Win[num].sty->top_hor);
        textattr(Win[num].clr->header);
        gotoxy(Win[num].left+2,Win[num].top);
        cprintf("%c",'[');
        gotoxy(Win[num].left+4,Win[num].top);
        cprintf("%c",']');
        i = (Win[num].right-Win[num].left-strlen(Win[num].head)) >> 1;
        gotoxy(Win[num].left+i,Win[num].top);
        cprintf("%s",Win[num].head);
        textattr(Win[num].clr->hili_tx);
        gotoxy(Win[num].left+3,Win[num].top);
        cprintf("%c",'■');
}
```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```
/* ------------------------- */
/*   File :  SETUP.INC       */
/* ------------------------- */

/* --------------------- Protocol --------------------------- */

char    *Pro_setup_name[MAX_SETUP-4] = {        // item of choices
                "Protocol",
                "Code",
                "Mode",
                "Parity",
                "Stop bit",
                "Err check",
                "Clock source",
                "Bits/sec",
                "Disp mode",
                "Bit order",
                "Bit sense",
        };

char    *Protocol_name[MAX_PROC-4] = {          // choice of protocol
                "ASYNC"
        };

char    *Code_name[MAX_CODE] = {                // choice of code
                "EBCDIC",
                "ASCII8",
                "ASCII7",
        };

char    *Mode_name[MAX_MODE] = {                // choice of SDLC mode
                "NRZ ",
                "NRZI",
        };
char    *Parity_name[MAX_PARITY] = {            // choice of CHAR parity
                "None",
                "Odd ",
                "Even",
        };

char    *Stop_name[MAX_STOPBIT] = {             // choice of ASYNC mode
                "1  stop bit ",
                "1½ stop bits",
                "2  stop bits",
        };

char    *Error_name[MAX_ERROR] = {              // choice of error checking
                "CCITT ",
                "CRC 16",
                "None  ",
        };

char    *DTE_clk_name[MAX_CLKSCR] = {           // choice of DTE clock
                "Internal",
                "External",
        };

char    *Speed_name[MAX_SPEED] = {              // choice of speed
                "50    ",       "75    ",
                "110   ",       "134.5",
                "150   ",       "200   ",
                "300   ",       "600   ",
```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```
                "1200 ",        "1800 ",
                "2000 ",        "2400 ",
                "3200 ",        "3600 ",
                "4800 ",        "7200 ",
                "9600 ",        "12000",
                "14400",        "16000",
                "19200",
        };

char    *Disp_mode_name[MAX_DISPMD] = {         // choice of display mode
                "DTE only     ",
                "DCE only     ",
                "DTE & DCE    ",
                "Data & Status",
        };

char    *Bit_order_name[MAX_BITORD] = {         // choice of bit order
                "MSB first",
                "LSB first",
        };

char    *Bit_sense_name[MAX_BITSND] = {         // choice of bit sense
                "Normal  ",
                "Inverted",
        };

SETUP_ARRAY  ASYNC_array[MAX_SETUP] = {
                0,      { 1,-2 },               // protocol
                1,      { 1,1,1,-2 },           // code
                0,      {-3 },                  // mode
                0,      { 1,1,1,-2 },           // parity
                0,      { 1,1,1,-2 },           // stop bit
                0,      {-3 },                  // err check
                0,      { 1,1,-2 },             // clock source
                11,     { 1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,-2 },
                3,      { 1,1,1,1,-2 },         // disp mode
                0,      { 1,1,-2 },             // bit order
                0,      { 1,1,-2 },             // bit sense
        };

SETUP Prot_setup = {
                PROC_ASYNC,                     // default function
                {                               // maximum function
                        MAX_SETUP-4,
                },{                             // choice name
                        Protocol_name,
                        Code_name,
                        Mode_name,
                        Parity_name,
                        Stop_name,
                        Error_name,
                        DTE_clk_name,
                        Speed_name,
                        Disp_mode_name,
                        Bit_order_name,
                        Bit_sense_name,
                },{
                        ASYNC_array,            // setup array
                }
        };

unsigned char   Start_CRC_char[CRC_ON_CHAR] = { 0xFF,0xFF };
```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```
unsigned char   Stop_CRC_char[CRC_OFF_CHAR] = { 0xFF,0xFF,0xFF };

char    Choc_line[50];                  // line number of choice menu
void    Edit_setup(char);
void    Write_setup(void);
void    Hili_setup(char,char,char,char,char);

extern  WIN_STYLE       Win_style[MAX_STYLE];
extern  WIN_ATTR        Win_attr[MAX_ATTR];     // window attribute
extern  WIN_STRUC       Win[MAX_WINDOW];

extern  void    Error(char*,...);
extern  char    Getch(void);
extern  void    Ungetch(char);
extern  void    Cputch(char,char,char,char);
extern  void    Scroll_up(char,char,char,char,char,char);
extern  void    Make_window(char,char,char,char,char*,WIN_ATTR*,WIN_STYLE*,char*,char);
```



เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```
/* ------------------------- */
/*   File :  SETUP.C         */
/* ------------------------- */

#include  <alloc.h>
#include  <conio.h>
#include  <ctype.h>
#include  <string.h>
#include  "constant.inc"
#include  "setup.inc"

static  SETUP           *Sp;
static  SETUP_ARRAY     *Sa;
static  char            **Item;
static  unsigned char   Max;

static char *Header[1] = {
                  " Setup Protocol ",
              };

void Edit_setup(char fc)                        // fc = 0,  Proctcol setup
{
        char            *hd,*scr1;
        unsigned char   ln,oln,x,y,oy;
        register char   c,val,oval;

        scr1 = (char*) malloc (2652);
        if (!scr1 ;| !Choc_line) {
            Error("Not enough memory.");
            if (scr1) free(scr1);
            return;
        }
        if (!fc) {
            Item = Pro_setup_name;
            hd = Header[0];
            Sp = &Prot_setup;
            val = Sp->proc;
            Max = Sp->max[val];
            Sa = Sp->st[val];           // protocol setup
        }
        Make_window(15,4,50,16,hd,&Win_attr[EDIT_ATTR],
                            &Win_style[STYLE3],scr1,WINDOW3);

        x = Win[WINDOW3].left+21;
        y = Win[WINDOW3].top+2;
        Write_setup();
        gotoxy(x,y);
        val = oval = Sa->val;                   // get val
        Hili_setup(1,x,y,0,val);
        for (ln=oln=0,oy=y;;) {
            switch(Getch()) {
                case 0:
                    switch(c=Getch()) {
                        case ALT_B:
                        case ALT_C:
                        case ALT_D:
                        case ALT_E:
                        case ALT_H:
                            Ungetch(c);
                            Ungetch(0);
                            Ungetch(ESCAPE);
                            break;
```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```
            case ARW_UP:
                if (ln>0) {
                    do --ln;
                    while ((Sa+ln)->sel[0] == -3);
                    oval=val=(Sa+ln)->val;
                    oy = y--;
                }
                break;

            case ARW_DN:
                if (ln < Max-1) {
                    do ++ln;
                    while ((Sa+ln)->sel[0] == -3);
                    oval=val=(Sa+ln)->val;
                    oy = y++;
                }
                break;
        }
        break;

    case SPACE:                     // go forward the choices
        if ((Sa+ln)->sel[val+1] != -2) {
            do ++val;
            while ((Sa+ln)->sel[val] == 0);
        }
        else {
            val = 0;
        }
        Hili_setup(0,x,y,ln,oval);
        Hili_setup(1,x,y,ln,val);
        break;

    case ENTER:
        if (!ln && !fc) {
            if (Sp->proc != val) {
                oln = 0;
                Sp->proc = val;
                Sa = Sp->st[val];
                Max = Sp->max[val];
                Write_setup();
                Hili_setup(1,x,y,ln,val);
            }
        }
        else {
            if (val != (Sa+ln)->val) {
                register char len;

                (Sa+ln)->val = val;                 // new val
                Write_setup();
                }
                Hili_setup(0,x,y,ln,oval);
                Hili_setup(1,x,y,ln,val);
            }
        }
        break;

    case ESCAPE:
        puttext(15,4,50,16,scr1);
        free (scr1);
        return;
}
```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```
            if (ln != oln) {
                Hili_setup(0,x,oy,oln,(Sa+oln)->val);
                Hili_setup(1,x,y,ln,val);
                oln=ln;
            }
            if (oval!=val)
                oval=val;
        }
    }

    void Write_setup(void)
    {
        register char           i,n,m,x,y;
        unsigned char           attr1,attr2;

        x = Win[WINDOW3].left+4;
        y = Win[WINDOW3].top+2;
        attr2 = Win[WINDOW3].clr->hot_key;
        attr1 = Win[WINDOW3].clr->norm_tx | 0x08;       // hex. value
        Scroll_up(Win[WINDOW3].left+1,Win[WINDOW3].top+1,
            Win[WINDOW3].right-1,Win[WINDOW3].bottom-1,
            Win[WINDOW3].clr->norm_tx,0);
        for (i=0;i<Max;i++) {
            if ((Sa+i)->sel[0] != -3) {                 // is selectable?
                textattr(attr2);
                gotoxy(x,y);
                cprintf("%-15s  ",*(Item+i));
                if (i >= SETUP_ADDR) {
                    if ((Sa+i)->sel[0] == -2) {
                        if (i == SETUP_CRCOFF || i == SETUP_CRCON) {
                            unsigned char *p;

                            if (i == SETUP_CRCOFF) {
                                p = Stop_CRC_char;
                                m = CRC_OFF_CHAR;
                            }
                            else {
                                p = Start_CRC_char;
                                m = CRC_ON_CHAR;
                            }
                            for (n=0;n<m;n++)
                                Cputch(x+n+17,y,attr1,*(p+n));
                        }
                        else Cputch(x+17,y,attr1,(Sa+i)->val);
                        y++;
                    }
                }
                else {
                textattr(Win[WINDOW3].clr->norm_tx);
                cprintf("%s",Sp->nm[i][(Sa+i)->val]);
                y++;
                }
            }
        }
    }

    void Hili_setup(char i,char x,char y,char ln,char val)
    {
        register char   n,m,attr;

        gotoxy(x,y);
        attr = Win[WINDOW3].clr->norm_tx | 0x08;
```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```
        if (i) {
            if (ln < SETUP_ADDR) {
                textattr(Win[WINDOW3].clr->hili_tx);
                cprintf("%s",Sp->nm[ln][val]);
            }
            else Cputch(x,y,Win[WINDOW3].clr->header,0x08,val);
        }
        else {
            if (ln >= SETUP_ADDR) {
                if (ln == SETUP_CRCOFF || ln == SETUP_CRCON) {
                    unsigned char *p;

                    if (ln == SETUP_CRCOFF) {
                        p = Stop_CRC_char;
                        m = CRC_OFF_CHAR;
                    }
                    else {
                        p = Start_CRC_char;
                        m = CRC_ON_CHAR;
                    }
                    for (n=0;n<m;n++)
                        Cputch(x+n,y,attr,*(p+n));
                }
                else Cputch(x,y,attr,(Sa+ln)->val);
                y++;
            }
            else {
                textattr(Win[WINDOW3].clr->norm_tx);
                cprintf("%-14s",Sp->nm[ln][val]);
            }
        }
    }
}
```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```
/* ------------------------- */
/*   File :  INTERP.INC      */
/* ------------------------- */

unsigned int    Recv_flag;      // receiving flag
unsigned char   Oper_mode;      // operating mode:0=DTE,1=DCE
unsigned char   Bit_length;     // number of bits length of data
unsigned char   Time_enable;    // '1' on bit 0-3 for enable timer
unsigned char   Lead_status;    // current modem lead status
unsigned int    Str_dta_inx;    // index to starting point data in data buffer
unsigned int    End_dta_inx;    // index to ending point data in data buffer
unsigned int    Str_dsp_inx;    // index to starting point of written data
unsigned int    End_dsp_inx;    // index to ending point of written data
unsigned int    Display_row;    // row to display
unsigned char   Display_col;    // column to display
unsigned char   X_code;         // table number of xfer code
unsigned char   Code_xfer[MAX_CODE] = { 0,3,3 };// transfer code table
unsigned char   Codelen[MAX_CODE] = { 8,8,7 };// bits per byte

unsigned int    Timer[4];       // timer tick counter,in every 1mS
unsigned int    Counter[3];     // counter,used in simulated program
unsigned char   Write_register[2][8];   // 8274 write only register
unsigned char   Lead_order[] = { 0,3,1,4,2 };//virtual modem lead register order
unsigned char   Clock_source[2] = { 0x01,0x02 };//internal or external clock
unsigned char   Bits_sense[2] = { 0x00,0x40 };// normal or inverted bit
unsigned char   Register_seq[8] = { 4,3,5,2,1,6,7 };// for 8274 programming
signed char     *Lead_name[7] = { "DTE","DCE","RTS","CTS","DCD","DTR","DSR" };
unsigned char   Lead_mask[] = { LEAD_RTS, LEAD_DTR, LEAD_CTS, LEAD_DSR, LEAD_CD };

unsigned int    Clock_divisor[MAX_SPEED] =
                        {               // at clock = 1.8432 MHz
        /* 50 - 150*/           0x9000,0x6000,0x4174,0x3586,0x3000,
        /* 200 - 1800*/         0x2400,0x1800,0x0C00,0x0600,0x0400,
        /* 2000 - 4800*/        0x0399,0x0300,0x0240,0x0200,0x0180,
        /* 7200 - 19200*/       0x0100,0x00C0,0x0099,0x0080,0x0073,0x0060
                        };

void    Start_disp(void);
void    Start_timer(void);
void    Start_file(void);

void    Stop_disp(void);
void    Stop_timer(void);
void    Stop_file(void);
void    Stop_test(void);

void    Run_program(void);
void    Disp_screen(char*);
void    Program_8274_card(void);
void    interrupt New_int0B(void);
void    Write_todsk(void);
char    Write_data_DTE(char,unsigned int);
char    Write_data_DCE(char,unsigned int);
char    Write_data_twoline(char,unsigned int);
char    Write_data_state(char,unsigned int);
char    (*Write_current_data)(char,unsigned int);

char    (*Write_data_pointer[MAX_DISPMD]) (char,unsigned int) = {
                Write_data_DTE,         // write data to screen mode
                Write_data_DCE,
                Write_data_twoline,
                Write_data_state,
```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งไม่อนุญาตให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```
};

extern  int             Fh;             // file handle
extern  char            Stack_ptr;      // buffer stack pointer
extern  char            Fname[2][80];    / filename
extern  signed  char    *File_extend[ ]; // file extended name
extern  signed  char    *File_keyword[3]; / keyword of file

extern  SETUP           Prot_setup;
extern  unsigned char   Disp_mode;      // current display mode
extern  DATA_STRUC      *Data_buff;     // receive data buffer
extern  unsigned char   *Prog_buff;     '/ program buffer pointer
extern  WIN_STRUC       Win[MAX_WINDOW];
extern  WIN_ATTR        Win_attr[MAX_ATTR]; // window attribute
extern  WIN_STYLE       Win_style[MAX_STYLE];
extern  ASCII_TABLE     To_ASCII_tbl[MAX_CONV_CODE][256];// conversion table (to ASCII)

extern  void    Rx_async_A(void);       // receive async mode (chanal A)
extern  void    Rx_async_B(void);       // receive async mode (chanal B)

extern  void    Ext_int_A(void);        // external status (chanal A)
extern  void    Ext_int_B(void);        // external status (chanal B)

extern  void    Special_A(void);        // special status (chanal A)
extern  void    Special_B(void);        // special status (chanal B)

extern  void    Init_ISR_func(void);    // initial current function of ISR
extern  void    Save_prot_setup(void);  // save all setup parameters of protocol
extern  void    Cputch(char,char,char,char);// put one char. with attribute
extern  void    Error(char*,...);
extern  char    Getch(void);
extern  void    Ungetch(char);

extern  char    Swap_bit(unsigned char,unsigned char);// swap data MSB to LSB
extern  void    Scroll_up(char,char,char,char,char,char);// scroll screen up

extern  void    Make_window(char,char,char,char,char*,WIN_ATTR*,WIN_STYLE*,char*,char);

ASCII_TABLE  To_ASCII_tbl[MAX_CONV_CODE][256] = {// conversion table (to ASCII)
             {
/* EBCDIC*/        {0x00,0}, {0x01,0}, {0x02,0}, {0x03,0}, {0x80,0},
                  {0x09,0}, {0x81,0}, {0x7F,0}, {0x08,1}, {0x82,0},
                  {0x83,0}, {0x0B,0}, {0x0C,0}, {0x0D,0}, {0x0E,0},
                  {0x0F,0}, {0x10,0}, {0x11,0}, {0x12,0}, {0x13,0},
                  {0x84,0}, {0x85,0}, {0x08,0}, {0x86,0}, {0x18,0},
/* 25*/             {0x19,0}, {0x87,0}, {0x88,0}, {0x1C,0}, {0x1D,0},
                  {0x1E,0}, {0x1F,0}, {0x89,0}, {0x8A,0}, {0x1C,0},
                  {0x23,1}, {0x8B,0}, {0x0A,0}, {0x17,0}, {0x1B,0},
                  {0x28,1}, {0x29,1}, {0x83,0}, {0x8C,0}, {0x2C,1},
                  {0x05,0}, {0x06,0}, {0x07,0}, {0x30,1}, {0x31,1},
/* 50 */            {0x16,0}, {0x33,1}, {0x8F,0}, {0x1E,0}, {0x8E,0},
                  {0x04,0}, {0x38,1}, {0x39,1}, {0x3A,1}, {0x8D,0},
                  {0x14,0}, {0x15,0}, {0x3E,1}, {0x1A,0}, {0x20,0},
                  {0x41,1}, {0x42,1}, {0x43,1}, {0x44,1}, {0x45,1},
                  {0x46,1}, {0x47,1}, {0x48,1}, {0x49,1}, {0x90,1},
/* 75 */            {0x2E,0}, {0x3C,0}, {0x28,0}, {0x2B,0}, {0x60,0},
                  {0x26,0}, {0x51,1}, {0x52,1}, {0x53,1}, {0x54,1},
                  {0x55,1}, {0x56,1}, {0x57,1}, {0x58,1}, {0x59,1},
                  {0x21,0}, {0x24,0}, {0x2A,0}, {0x29,0}, {0x3B,0},
                  {0x91,0}, {0x2D,0}, {0x2F,0}, {0x62,1}, {0x63,1},
/* 100 */           {0x64,1}, {0x65,1}, {0x66,1}, {0x67,1}, {0x68,1},
                  {0x69,1}, {0x7C,0}, {0x2C,0}, {0x25,0}, {0x5F,0},
```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```
                {0x3E,0}, {0x3F,0}, {0x70,1}, {0x71,1}, {0x72,1},
                {0x73,1}, {0x74,1}, {0x75,1}, {0x76,1}, {0x77,1},
                {0x78,1}, {0x27,0}, {0x3A,0}, {0x23,0}, {0x40,0},
/* 125 */       {0x27,0}, {0x3D,0}, {0x22,0}, {0x80,1}, {0x61,0},
                {0x62,0}, {0x63,0}, {0x64,0}, {0x65,0}, {0x66,0},
                {0x67,0}, {0x68,0}, {0x69,0}, {0x8A,1}, {0x8B,1},
                {0x8C,1}, {0x8D,1}, {0x8E,1}, {0x8F,1}, {0x90,1},
                {0x6A,0}, {0x6B,0}, {0x6C,0}, {0x6D,0}, {0x6E,0},
/* 150 */       {0x6F,0}, {0x70,0}, {0x71,0}, {0x72,0}, {0x9A,1},
                {0x9B,1}, {0x9C,1}, {0x9D,1}, {0x9E,1}, {0x9F,1},
                {0xA0,0}, {0x7E,0}, {0x73,0}, {0x74,0}, {0x75,0},
                {0x76,0}, {0x77,0}, {0x78,0}, {0x79,0}, {0x7A,0},
                {0xAA,1}, {0xAB,1}, {0xAC,1}, {0xAD,1}, {0xAE,1},
/* 175 */       {0xAF,1}, {0xB0,1}, {0xB1,1}, {0xB2,1}, {0xB3,1},
                {0xB4,1}, {0xB5,1}, {0xB6,1}, {0xB7,1}, {0xB8,1},
                {0xB9,1}, {0xBA,1}, {0xBB,1}, {0xBC,1}, {0xBD,1},
                {0xBE,1}, {0xBF,1}, {0x7B,0}, {0x41,0}, {0x42,0},
                {0x43,0}, {0x44,0}, {0x45,0}, {0x46,0}, {0x47,0},
/* 200 */       {0x48,0}, {0x49,0}, {0xCA,1}, {0xCB,1}, {0x92,0},
                {0xCD,1}, {0x93,0}, {0xCF,1}, {0x7D,0}, {0x4A,0},
                {0x4B,0}, {0x4C,0}, {0x4D,0}, {0x4E,0}, {0x4F,0},
                {0x50,0}, {0x51,0}, {0x52,0}, {0xDA,1}, {0xDB,1},
                {0xDC,1}, {0xDD,1}, {0xDE,1}, {0xDF,1}, {0x5C,0},
/* 225 */       {0xE1,1}, {0x53,0}, {0x54,0}, {0x55,0}, {0x56,0},
                {0x57,0}, {0x58,0}, {0x59,0}, {0x5A,0}, {0xEA,1},
                {0xEB,1}, {0x94,0}, {0xED,1}, {0xEE,1}, {0xEF,1},
                {0x30,0}, {0x31,0}, {0x32,0}, {0x33,0}, {0x34,0},
                {0x35,0}, {0x36,0}, {0x37,0}, {0x38,0}, {0x39,0},
/* 250 */       {0x6C,0}, {0xFB,1}, {0xFC,1}, {0xFD,1}, {0xFE,1},
                {0xFF,1}
        },{
/* Transcode */ {0x01,0}, {0x41,0}, {0x42,0}, {0x43,0}, {0x44,0},
                {0x45,0}, {0x46,0}, {0x47,0}, {0x48,0}, {0x49,0},
                {0x02,0}, {0x2E,0}, {0x3C,0}, {0x07,0}, {0x1B,0},
                {0x17,0}, {0x26,0}, {0x4A,0}, {0x4B,0}, {0x4C,0},
                {0x4D,0}, {0x4E,0}, {0x4F,0}, {0x50,0}, {0x51,0},
/* 25 */        {0x52,0}, {0x20,0}, {0x24,0}, {0x2A,0}, {0x1F,0},
                {0x04,0}, {0x10,0}, {0x2D,0}, {0x2F,0}, {0x53,0},
                {0x54,0}, {0x55,0}, {0x56,0}, {0x57,0}, {0x58,0},
                {0x59,0}, {0x5A,0}, {0x1B,0}, {0x2C,0}, {0x25,0},
                {0x05,0}, {0x03,0}, {0x09,0}, {0x30,0}, {0x31,0},
/* 50 */        {0x32,0}, {0x33,0}, {0x34,0}, {0x35,0}, {0x36,0},
                {0x37,0}, {0x38,0}, {0x39,0}, {0x16,0}, {0x23,0},
                {0x40,0}, {0x15,0}, {0x19,0}, {0x7F,0}, {0x00,0},
                {0x00,0}, {0x00,0}, {0x00,0}, {0x00,0}, {0x00,0},
                {0x00,0}, {0x00,0}, {0x00,0}, {0x00,0}, {0x00,0},
/* 75 */        {0x00,0}, {0x00,0}, {0x00,0}, {0x00,0}, {0x00,0},
                {0x00,0}, {0x00,0}, {0x00,0}, {0x00,0}, {0x00,0},
                {0x00,0}, {0x00,0}, {0x00,0}, {0x00,0}, {0x00,0},
                {0x00,0}, {0x00,0}, {0x00,0}, {0x00,0}, {0x00,0},
                {0x00,0}, {0x00,0}, {0x00,0}, {0x00,0}, {0x00,0},
/* 100 */       {0x00,0}, {0x00,0}, {0x00,0}, {0x00,0}, {0x00,0},
                {0x00,0}, {0x00,0}, {0x00,0}, {0x00,0}, {0x00,0},
                {0x00,0}, {0x00,0}, {0x00,0}, {0x00,0}, {0x00,0},
                {0x00,0}, {0x00,0}, {0x00,0}, {0x00,0}, {0x00,0},
                {0x00,0}, {0x00,0}, {0x00,0}, {0x00,0}, {0x00,0},
/* 125 */       {0x00,0}, {0x00,0}, {0x00,0}, {0x00,0}, {0x00,0},
                {0x00,0}, {0x00,0}, {0x00,0}, {0x00,0}, {0x00,0},
                {0x00,0}, {0x00,0}, {0x00,0}, {0x00,0}, {0x00,0},
                {0x00,0}, {0x00,0}, {0x00,0}, {0x00,0}, {0x00,0},
                {0x00,0}, {0x00,0}, {0x00,0}, {0x00,0}, {0x00,0},
/* 150 */       {0x00,0}, {0x00,0}, {0x00,0}, {0x00,0}, {0x00,0},
```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```
                {0x00,0}, {0x00,0}, {0x00,0}, {0x00,0}, {0x00,0},
                {0x00,0}, {0x00,0}, {0x00,0}, {0x00,0}, {0x00,0},
                {0x00,0}, {0x00,0}, {0x00,0}, {0x00,0}, {0x00,0},
                {0x00,0}, {0x00,0}, {0x00,0}, {0x00,0}, {0x00,0},
/* 175 */       {0x00,0}, {0x00,0}, {0x00,0}, {0x00,0}, {0x00,0},
                {0x00,0}, {0x00,0}, {0x00,0}, {0x00,0}, {0x00,0},
                {0x00,0}, {0x00,0}, {0x00,0}, {0x00,0}, {0x00,0},
                {0x00,0}, {0x00,0}, {0x00,0}, {0x00,0}, {0x00,0},
                {0x00,0}, {0x00,0}, {0x00,0}, {0x00,0}, {0x00,0},
/* 200 */       {0x00,0}, {0x00,0}, {0x00,0}, {0x00,0}, {0x00,0},
                {0x00,0}, {0x00,0}, {0x00,0}, {0x00,0}, {0x00,0},
                {0x00,0}, {0x00,0}, {0x00,0}, {0x00,0}, {0x00,0},
                {0x00,0}, {0x00,0}, {0x00,0}, {0x00,0}, {0x00,0},
                {0x00,0}, {0x00,0}, {0x00,0}, {0x00,0}, {0x00,0},
/* 225 */       {0x00,0}, {0x00,0}, {0x00,0}, {0x00,0}, {0x00,0},
                {0x00,0}, {0x00,0}, {0x00,0}, {0x00,0}, {0x00,0},
                {0x00,0}, {0x00,0}, {0x00,0}, {0x00,0}, {0x00,0},
                {0x00,0}, {0x00,0}, {0x00,0}, {0x00,0}, {0x00,0},
                {0x00,0}, {0x00,0}, {0x00,0}, {0x00,0}, {0x00,0},
/* 250 */       {0x00,0}, {0x00,0}, {0x00,0}, {0x00,0}, {0x00,0},
                {0x00,0},
            }
        };

unsigned char ASCII_to_tbi[MAX_CONV_CODE][128] = {// conversion table (from ASCII)
            {
/* to EBCDIC */     0x00,0x01,0x02,0x03,0x37,0x2D,0x2E,0x2F,0x16,0x05,
                    0x25,0x0B,0x0C,0x0D,0x0E,0x0F,0x10,0x11,0x12,0x13,
                    0x3C,0x3D,0x32,0x26,0x18,0x19,0x3F,0x27,0x1C,0x1D,
                    0x1E,0x1F,0x40,0x5A,0x7F,0x7B,0x5B,0x6C,0x50,0x79,
                    0x4D,0x5D,0x5C,0x4E,0x6B,0x60,0x4B,0x61,0xF0,0xF1,
/* 50 */            0xF2,0xF3,0xF4,0xF5,0xF6,0xF7,0xF8,0xF9,0x7A,0x5E,
                    0x4C,0x7e,0x6E,0x6F,0x7C,0xC1,0xC2,0xC3,0xC4,0xC5,
                    0xC6,0xC7,0xC8,0xC9,0xD1,0xD2,0xD3,0xD4,0xD5,0xD6,
                    0xD7,0xD8,0xD9,0xE2,0xE3,0xE4,0xE5,0xE6,0xE7,0xE8,
                    0xE9,0x4A,0xD0,0x4F,0x5F,0x6d,0x4F,0x81,0x82,0x83,
/* 100 */           0x84,0x85,0x86,0x87,0x88,0x89,0x91,0x92,0x93,0x94,
                    0x95,0x96,0x97,0x98,0x99,0xA2,0xA3,0xA4,0xA5,0xA6,
                    0xA7,0xA8,0xA9,0xC0,0x6A,0xD0,0xA1,0x07
            },{
/* to transcode */  0xFF,0x0C,0x0A,0x2E,0x1E,0x2D,0xFF,0x0D,0xFF,0x2F,
                    0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,
                    0xFF,0x3D,0x3A,0x0F,0xFF,0x3E,0xFF,0x2A,0xFF,0xFF,
                    0xFF,0x1D,0x1A,0xFF,0xFF,0x3B,0x1B,0x2C,0x10,0xFF,
                    0xFF,0xFF,0x1C,0xFF,0x2B,0x20,0x0B,0x21,0x30,0x31,
                    0x32,0x33,0x34,0x35,0x36,0x37,0x38,0x39,0xFF,0xFF,
                    0x0C,0xFF,0xFF,0xFF,0x3C,0x01,0x02,0x03,0x04,0x05,
                    0x06,0x07,0x08,0x09,0x11,0x12,0x13,0x14,0x15,0x16,
                    0x17,0x18,0x19,0x22,0x23,0x24,0x25,0x26,0x27,0x28,
                    0x29,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0x01,0x02,0x03,
                    0x04,0x05,0x06,0x07,0x08,0x09,0x11,0x12,0x13,0x14,
                    0x15,0x16,0x17,0x18,0x19,0x22,0x23,0x24,0x25,0x26,
                    0x27,0x28,0x29,0xFF,0xFF,0xFF,0xFF,0xFF,
            }
        };
```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```
/* ------------------------- */
/*   File : INTERP.C         */
/* ------------------------- */

#include <io.h>
#include <dos.h>
#include <fcntl.h>
#include <stdlib.h>
#include <string.h>
#include <dir.h>
#include <mem.h>
#include <conio.h>
#include "constant.inc"
#include "interp.inc"

static  unsigned int    Disk_inx;       // index of buffer for write to disk
static  unsigned char   *Prg_ptr;       // program pointer
static  unsigned char   **Bck_ptr;      // array of address of start of each block
static  unsigned char   Disp_enable;    // 1 for enable dispay data on screen
static  char    *Head[] = { " Monitor ", " Monitor DTE ", " Monitor DCE " };

void Run_program(void)
{
        register unsigned char  n;

        if ((Bck_ptr=(unsigned char**) malloc (sizeof(char*)*MAX_PRG_BLCK)) == '\0') {
            Error("Not enough memory.");
            return;
        }
        Disk_inx = 0;           // begin of file
        Disp_enable = 1;        // display data on screen enabled
        Str_dta_inx = 0;        // index to begin of data buffer
        End_dta_inx = 0;        // no data entry
        Str_dsp_inx = 0;        // no data displayed
        End_dsp_inx = 0;        // no data displayed
        Lead_status = 0;        // no lead activate
        Time_enable = 0;        // disable all timer
        memset(Timer,0,4);      // initial all timer
        memset(Counter,0,3);    // clear counter

        Disp_mode = (Prot_setup.st[Prot_setup.proc]+SETUP_DISP)->val;          // display mode
        n = (Prot_setup.st[Prot_setup.proc]+SETUP_CODE)->val;                  // code of data
        Bit_length  = Codelen[n];                                              // bit length of data
        X_code = Code_xfer[n];
        Oper_mode = 0;          // Monitor mode

        Init_ISR_func();
        Prg_ptr = *(Bck_ptr+1);         // start at first block
        Disp_screen(Head[Oper_mode]);
        Program_8274_card();            // initial all hardware
        outportb(OCW1_8259,inportb(OCW1_8259) & 0xD7);//enable IQR3,5 of 8259
        Scroll_up(1,25,80,25,BLACK,1);

        for (;;) {
            if (kbhit() || Stack_ptr) {
                if ((n=Getch()) == 0) {
                    switch(n=Getch()) {
                        case ALT_B:
                        case ALT_D:
                        case ALT_E:
                        case ALT_P:
                        case ALT_H:
```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```
                    Ungetch(n);
                    Ungetch(0);
                    Ungetch(ESCAPE);
                    break;
                }
            }
            else {
                if (n == ESCAPE)
                    break;
            }
        }
        Lead_status = inportb(PORT_C_8255);
        gotoxy(1,25);
        cprintf("%02X %02X",inportb(COMM_8274_A),inportb(COMM_8274_B));
        if (End_dsp_inx != End_dta_inx && Disp_enable) {
            (*Write_current_data)(1,End_dsp_inx);
            if (++End_dsp_inx >= DAT_BUF_SIZE)// advance to next position
                End_dsp_inx = 0;
        }
    }
    Stop_test();
    Stop_file();

    free(Bck_ptr);
    free(Win[WINDOW3].scr);
    puttext(2,3,79,23,Win[WINDOW3].scr);
}

void Disp_screen(char *header)
{
    register char   i,n;
    register char   c,attr;
    unsigned char   *scrn;

    scrn = (unsigned char*) mailoc (3432);
    if (!scrn) {
        Error("Not enough memory for data screen.");
        return;
    }
    Make_window(2,3,79,23,header,&Win_attr[SIML_ATTR],&Win_style[STYLE2],scrn,WINDOW3);
    textattr(attr=Win[WINDOW3].clr->border);
    c = (Prot_setup.st[Prot_setup.proc]+SETUP_DISP)->val;
    Write_current_data = Write_data_pointer[c];//pointer to function of write data

    switch (c) {
        case DISP_DTE:
        case DISP_DCE:
        case DISP_DATA:
            for (n=5;n<19;n+=2)
                Scroll_up(3,n,78,n,Win[WINDOW3].clr->hili_tx,1);
            Display_row = 5;        // top row
            Display_col = 3;        // left most column
            break;

        case DISP_STAT:
            attr = Win[WINDOW3].clr->hili_tx;
            Scroll_up(6,5,78,5,attr,1);
            Scroll_up(6,13,78,13,attr,1);
            textattr(Win[WINDOW3].clr->header);
            for (n=0,i=5;i<12;i++,n++) {
                gotoxy(3,i); cprintf("%s",Lead_name[n]);
                gotoxy(3,i+8); cprintf("%s",Lead_name[n]);
```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```
                }
                Display_row = 5;        // top row
                Display_col = 6;        // left most column
                break;
        }
        Data_buff->misc = 0;            // set start status
}

void Program_8274_card(void)
{
        register unsigned char  *t;
        register unsigned char  c,n;

        c = Prot_setup.proc;
        Write_register[CHNAL_A][REGISTER_5] = 0;        //unused
        Write_register[CHNAL_A][REGISTER_3] = (Bit_length-5)<<6 | WR3_RxENA;

        if ((n=(Prot_setup.st[c]+SETUP_PRTY)->val) > 0)
            Write_register[CHNAL_A][REGISTER_4] =  n | WR4_PTYENA;
        else Write_register[CHNAL_A][REGISTER_4] =  0;// parity none

        if (c == PROC_ASYNC) {
                Write_register[CHNAL_A][REGISTER_4] |= (((Prot_setup.st[c]+SETUP_STOP)->val+1)<<2);
                Write_register[CHNAL_A][REGISTER_6] = 0;
                Write_register[CHNAL_A][REGISTER_7] = 0;
        }

        Write_register[CHNAL_B][REGISTER_5] = 0;     // unused
        Write_register[CHNAL_B][REGISTER_2] = 0;     // interrupt vector
        Write_register[CHNAL_A][REGISTER_2] = 0x14;  // non vectored mode
        Write_register[CHNAL_A][REGISTER_1] = 0x10;  // type of interrupt
        Write_register[CHNAL_B][REGISTER_1] = 0x14;
        Write_register[CHNAL_B][REGISTER_3] = Write_register[CHNAL_A][REGISTER_3];
        Write_register[CHNAL_B][REGISTER_4] = Write_register[CHNAL_A][REGISTER_4];
        Write_register[CHNAL_B][REGISTER_6] = Write_register[CHNAL_A][REGISTER_6];
        Write_register[CHNAL_B][REGISTER_7] = Write_register[CHNAL_A][REGISTER_7];

        t = (unsigned char*)(Clock_divisor+(Prot_setup.st[Prot_setup.proc]+SETUP_SPEED)->val);
        outportb(CONTROL_8254,0x76);
        outportb(COUNT1_8254,0x04);
        outportb(COUNT1_8254,0x00);     // counter 1: divide by 4
        outportb(CONTROL_8254,0xB6);
        outportb(COUNT2_8254,*t++);     // counter 2: baud rate generator
        outportb(COUNT2_8254,*t);       // MSB of counter 2

        outportb(COMM_8274_B,WR0_CHNRST);       // chanal B reset
        for (n=0;n<7;n++) {
            outportb(COMM_8274_B,Register_seq[n] | WR0_RSTEXT);
            outportb(COMM_8274_B,Write_register[CHNAL_B][Register_seq[n]]);
        }
        outportb(COMM_8274_B,WR0_RSTEXT);
        outportb(COMM_8274_B,WR0_ERRRST);

        outportb(COMM_8274_A,WR0_CHNRST);       // chanal A reset
        for (n=0;n<7;n++) {
            outportb(COMM_8274_A,Register_seq[n] | WR0_RSTEXT);
            outportb(COMM_8274_A,Write_register[CHNAL_A][Register_seq[n]]);
        }
        outportb(COMM_8274_A,WR0_RSTEXT);       // reset Tx interrupt pending
        outportb(COMM_8274_A,WR0_ERRRST);       // error reset

        outportb(PORT_B_8255,0);
```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```
        outportb(PORT_A_8255,0x80 | Bits_sense[(Prot_setup.st[Prot_setup.proc]+SETUP_SENSE)->val]);
}

void Start_disp(void)
{
        Disp_enable = 1;
        End_dsp_inx = End_dta_inx;
        Prg_ptr += 3;
}

void Start_timer(void)
{
        Time_enable |= (0x01<<((*(Prg_ptr+3)-1));// enable timer
        Prg_ptr += 5;
}

void Start_file(void)
{
        PATH_STRUC      path;

        if (Fh > 0)  _close(Fh);
        fnsplit(Fname[2],path.drv,path.dir,path.fil,path.ext);
        strcpy(path.ext,File_extend[0]);        // force ext. name
        fnmerge(Fname[2],path.drv,path.dir,path.fil,path.ext);
        if ((Fh=_creat(Fname[2],FA_ARCH)) == -1) {
            Error("Can't creat file %s",Fname[1]);
            Prg_ptr += 4;
            return;
        }
        _write(Fh,File_keyword[0],strlen(File_keyword[0]));     // keyword
        _write(Fh,File_keyword[2],strlen(File_keyword[2]));
        Disk_inx = End_dta_inx;
        Save_prot_setup();
        Prg_ptr += 4;
}

void Stop_disp(void)
{
        Disp_enable = 0;
        Prg_ptr += 3;
}

void Stop_timer(void)
{
        Time_enable &= ~(0x01 << (*(Prg_ptr+3)-1));     // disable timer
        Prg_ptr += 5;
}

void Stop_file(void)
{
        register unsigned char  n;

        if (Fh > 0) {
            if (_close(Fh) == -1)
                Error("Can't close file %s.",Fname[2]);
            Fh = -1;
        }
        Prg_ptr += 3;
}

void Stop_test(void)
{
```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```
        outportb(PORT_A_8255,0);        // clear all
        outportb(PORT_B_8255,0);
        outportb(OCW1_8259,inportb(OCW1_8259) , 0x28);//disable IQR3,5 of 8259
        Prg_ptr += 3;
}

// return value: 0=normal,1=next page,2=no data

char Write_data_DTE(char cursor,unsigned int index)
{
        DATA_STRUC              *d_ptr;
        register unsigned int   i;
        register unsigned char  c,n,ret,attr;

        ret = 0;
        d_ptr = Data_buff+index;
        if (d_ptr->misc & MISC_HILGHT)
            attr = Win[WINDOW3].clr->border;
        else attr = Win[WINDOW3].clr->hili_tx;
        c = n = d_ptr->data;
        if (X_code < 3) {
            c = To_ASCII_tbl[X_code][n].ascii;
            if (To_ASCII_tbl[X_code][n].attr)
                attr |= 0x08;                   // display in HEX char.
        }
        else if (n > 0x80) attr |= 0x08;
        Cputch(Display_col,Display_row,attr,c);         // DTE
        Cputch(Display_col,Display_row+1,Win[WINDOW3].clr->norm_tx,0x20);

        if (Display_col>=78) {
            Display_row += 2;
            if (Display_row > 17) {
                Display_row = 5;
                ret++;
            }
            Display_col = 2;
        }
        Display_col++;
        if (cursor) {
            Cputch(Display_col,Display_row,Win[WINDOW3].clr->norm_tx|0x80,0xFF);         // DTE cursor
            Cputch(Display_col,Display_row+1,Win[WINDOW3].clr->hili_tx|0x80,0xFF);      // DCE cursor
        }
        return ret;
}

char Write_data_DCE(char cursor,unsigned int index)
{
        DATA_STRUC              *d_ptr;
        register unsigned int   i;
        register unsigned char  c,n,ret,attr;

        ret = 0;
        d_ptr = Data_buff+index;
        if (d_ptr->misc & MISC_HILGHT)
            attr = Win[WINDOW3].clr->border;
        else attr = Win[WINDOW3].clr->norm_tx;
        c = n = d_ptr->data;
        if (X_code < 3) {
            c = To_ASCII_tbl[X_code][n].ascii;
            if (To_ASCII_tbl[X_code][n].attr)
                attr |= 0x08;           // display in HEX char.
```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น การอ้างอิงถึงต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```
        }
        else if (n > 0x80) attr |= 0x08;
        Cputch(Display_col,Display_row+1,attr,c);   // DCE
        Cputch(Display_col,Display_row,Win[WINDOW3].clr->hili_tx,0xFF); // DTE

        if (Display_col>=78) {
            Display_row += 2;
            if (Display_row > 17) {
                Display_row = 5;
                ret++;
            }
            Display_col = 2;
        }
        Display_col++;
        if (cursor) {
            Cputch(Display_col,Display_row,Win[WINDOW3].clr->norm_tx|0x80,0xFF);        // DTE cursor
            Cputch(Display_col,Display_row+1,Win[WINDOW3].clr->hili_tx|0x80,0xFF);      // DCE cursor
        }
        return ret;
}

char Write_data_twoline(char cursor,unsigned int index)
{
        DATA_STRUC              *d_ptr;
        register unsigned int   i;
        register unsigned char  c,n,ret,attr1,attr2;

        ret = 0;
        d_ptr = Data_buff+index;
        if (d_ptr->misc & MISC_HILGHT) {
            attr1 = Win[WINDOW3].clr->border;
            attr2 = Win[WINDOW3].clr->border;
        }
        else {
            attr1 = Win[WINDOW3].clr->hili_tx;
            attr2 = Win[WINDOW3].clr->norm_tx;
        }
        c = n = d_ptr->data;
        if (X_code < 3) {
            c = To_ASCII_tbl[X_code][n].ascii;
            if (To_ASCII_tbl[X_code][n].attr) {
                attr1 |= 0x08;
                attr2 |= 0x08;          // display in HEX char.
            }
        }
        else if (n > 0x80) {
            attr1 |= 0x08;
            attr2 |= 0x08;
        }
        if (d_ptr->misc & MISC_DTE) {   // is DTE
            Cputch(Display_col,Display_row,attr1,c);    // DTE
            Cputch(Display_col,Display_row+1,Win[WINDOW3].clr->norm_tx,0x20);
        }
        else {
            Cputch(Display_col,Display_row+1,attr2,c);  // DCE
            Cputch(Display_col,Display_row,Win[WINDOW3].clr->hili_tx,0x20);
        }
        if (Display_col>=78) {
            Display_row += 2;
            if (Display_row > 17) {
                Display_row = 5;
                ret++;
```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งไม่อนุญาตให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```
            }
            Display_col = 2;
        }
        Display_col++;
        if (cursor) {
            Cputch(Display_col,Display_row,Win[WINDOW3].clr->norm_tx|0x80,0xFF);        // DTE cursor
            Cputch(Display_col,Display_row+1,Win[WINDOW3].clr->hili_tx|0x80,0xFF);      // DCE cursor
        }
        return ret;
}

char Write_data_state(char cursor,unsigned int index)
{
        DATA_STRUC              *d_ptr;
        register unsigned char  i,lead;
        register unsigned char  c,n,ret,attr1,attr2;

        ret = 0;
        d_ptr = Data_buff+index;
        if (d_ptr->misc & MISC_HILGHT) {
            attr1 = Win[WINDOW3].clr->border;
            attr2 = Win[WINDOW3].clr->border;
        }
        else {
            attr1 = Win[WINDOW3].clr->hili_tx;
            attr2 = Win[WINDOW3].clr->norm_tx;
        }
        c = n = d_ptr->data;
        if (!index)
            lead = (Data_buff+DAT_BUF_SIZE-1)->lead;
        else if (index == Str_dta_inx)
            lead = 0;
        else lead = (d_ptr-1)->lead;

        lead ^= d_ptr->lead;            // lead change state?
        Display_row += 2;
        for (i=0;i<5;i++) {
            if (lead & Lead_mask[i]) {
                if ((d_ptr-1)->lead & Lead_mask[i])     // previous state is low
                    Cputch(Display_col,Display_row+Lead_order[i],attr2,0x9E);   // fall down
                else Cputch(Display_col,Display_row+Lead_order[i],attr2,0x9F);// rise up
            }
            else {
                if (d_ptr->lead & Lead_mask[i])
                    Cputch(Display_col,Display_row+Lead_order[i],attr2,0x9D);   // hi state
                else Cputch(Display_col,Display_row+Lead_order[i],attr2,0x9C);// low state
            }
        }
        Display_row -= 2;
        if (X_code < 3) {
            c = To_ASCII_tbl[X_code][n].ascii;
            if (To_ASCII_tbl[X_code][n].attr) {
                attr1 |= 0x08;
                attr2 |= 0x08;          // display in HEX char.
            }
        }
        else if (n > 0x80) {
            attr1 |= 0x08;
            attr2 |= 0x08;
        }
        if (d_ptr->misc & MISC_DTE) {
            Cputch(Display_col,Display_row,attr1,c);    // DTE
```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```
            Cputch(Display_col,Display_row+1,Win[WINDOW3].clr->norm_tx,0x20);
        }
        else {
            Cputch(Display_col,Display_row+1,attr2,c);  // DCE
            Cputch(Display_col,Display_row,Win[WINDOW3].clr->hili_tx,0x20);
        }
        if (Display_col>=78) {
            Display_row += 8;
            if (Display_row > 17) {
                ret++;
                Display_row = 5;
            }
            Display_col = 5;
        }
        Display_col++;
        if (cursor) {
            Cputch(Display_col,Display_row,Win[WINDOW3].clr->norm_tx|0x80,0xFF);        // DTE cursor
            Cputch(Display_col,Display_row+1,Win[WINDOW3].clr->hili_tx|0x80,0xFF);     // DCE cursor
            Scroll_up(Display_col,Display_row+2,Display_col,Display_row+6,Win[WINDOW3].clr->norm_tx,5);
        }
        return ret;
}

void Write_todsk(void)                  // 1 mS time interval
{
        register char   i,n;

        for (n=0,i=0x01;n<4;n++,i<<=1) {
            if (i & Time_enable)
                Timer[n]++;             // increment if timer eanble
        }
        outportb(OCW2_8259,0x65);       // EOI command of 8259,IRQ3,IRQ5
        if (Disk_inx != End_dta_inx && Fh) {
            if (Disk_inx+DSK_BUF_SIZE > End_dta_inx) {
                i = _write(Fh,Data_buff+Disk_inx,DSK_BUF_SIZE*sizeof(DATA_STRUC));
                Disk_inx += DSK_BUF_SIZE;
            }
            else if (DAT_BUF_SIZE-Disk_inx+End_dta_inx > DSK_BUF_SIZE) {
                n = DAT_BUF_SIZE-Disk_inx;
                i = _write(Fh,Data_buff+Disk_inx,n*sizeof(DATA_STRUC));
                n = DSK_BUF_SIZE-n;
                i = _write(Fh,Data_buff,n*sizeof(DATA_STRUC));
                Disk_inx = n;
            }
            else i = 0;
            if (i == -1) {
                Error("Error in write data to disk.File was close.");
                Fh = _close(Fh);
            }
        }
}
```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```
/* ------------------------- */
/*   File :  ISR8274.INC     */
/* ------------------------- */

unsigned char   Rx_Direct_A[2] = { 0x00,MISC_DTE };// data from DCE and DTE
unsigned char   Rx_Direct_B[2] = { MISC_DTE,0x00 };// data from DCE and DTE

void    (**ISR_table) ();       // pointer to ISR table function
void    Null_func(void);
void    Store_time(void);
void    Init_ISR_func(void);

void    Rx_async_A(void);       // receive async mode (chanal A)
void    Rx_async_B(void);       // receive async mode (chanal B)

void    Ext_int_A(void);        // external status (chanal A)
void    Ext_int_B(void);        // external status (chanal B)

void    Special_A(void);        // special status (chanal A)
void    Special_B(void);        // special status  chanal B)
void    Write_todsk(void);
void    Write_status(void);

void    interrupt New_int0D(void);

void    (*ISR_async_func[]) () = {
                Null_func,
                Ext_int_B,
                Rx_async_B,
                Special_B,
                Null_func,      // transmit interrupt function
                Ext_int_A,      // external & status interrupt
                Rx_async_A,     // receive interrupt function
                Special_A,      // spacial condition interrupt
        };

extern  SETUP           Prot_setup;
extern  DATA_STRUC      *Data_buff;     // receive data buffer

extern  unsigned char   Disp_mode;      // current display mode
extern  unsigned int    Str_dta_inx;    // index to starting point data in data buffer
extern  unsigned int    End_dta_inx;    // index to ending point data in data buffer
extern  unsigned int    Str_dsp_inx;    // index to starting point of written data
extern  unsigned int    End_dsp_inx;    // index to ending point of written data
extern  unsigned int    Timer[4];       // timer tick counter,in every 1mS
extern  unsigned int    Counter[3];     // counter,used in simulated program
extern  unsigned int    Max_send;       // maximum of charactors for send
extern  unsigned char   Oper_mode;      // operating mode:0=DTE,1=DCE
extern  unsigned int    Send_flag;      // sending flag
extern  unsigned int    Recv_flag;      // receiving flag
extern  unsigned char   Bit_length;     // number of bits length of data
extern  unsigned char   Lead_status;    // current modem lead status
extern  unsigned char   Time_enable;    // '1' on bit 0-3 for enable timer
extern  unsigned char   Bits_sense[2];  // normal or inverted bit
extern  unsigned char   Lead_mask[5];
extern  unsigned char   Write_register[2][8];   // 8274 write only register
```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```
/* ------------------------- */
/*   File :  ISR8274.C       */
/* ------------------------- */

#include  <dos.h>
#include  "constant.inc"
#include  "isr8274.inc"

void interrupt New_int0D(void)
{
        register unsigned char  c;

        outportb(COMM_8274_B,02);               // read RR2
        c = inportb(COMM_8274_B);
        (*(ISR_table+c)) ();                    // interrupt function
        outportb(COMM_8274_A,WR0_ENDINT);       // EOI command of 8274
        outportb(OCW2_8259,0x65);               // EOI command of 8259,IRQ5
}

void Null_func(void)
{
}

void Rx_async_A(void)
{
        DATA_STRUC      *d_ptr;

        d_ptr = Data_buff+End_dta_inx;
        d_ptr->data = inportb(DATA_8274_A);
        d_ptr->lead = Lead_status;
        d_ptr->misc |= Rx_Direct_A[Oper_mode];          // data
        Store_time();
}

void Rx_async_B(void)
{
        DATA_STRUC      *d_ptr;

        d_ptr = Data_buff+End_dta_inx;
        d_ptr->data = inportb(DATA_8274_B);
        d_ptr->lead = Lead_status;
        d_ptr->misc |= Rx_Direct_B[Oper_mode];          // data
        Store_time();
}

void Ext_int_A(void)
{
}

void Write_status(void)
{
}

void Ext_int_B(void)
{
}

void Special_A(void)
{
        outportb(COMM_8274_A,01);               // read RR1
        (Data_buff+End_dta_inx)->misc |= (inportb(COMM_8274_A) & 0xF0);         // read status
        outportb(COMM_8274_A,WR0_ERRRST);       // error reset
```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```
}

void Special_B(void)
{
        outportb(COMM_8274_B,1);                    / read RR1
        (Data_buff+End_dta_inx)->misc |= (inportb(COMM_8274_B) & 0xF0);
        outportb(COMM_8274_B,WR0_ERRRST ;           :/ error reset
}

void Store_time(voic)
{
        DATA_STRUC      *d_ptr;
        register unsigned cnar  i;

        d_ptr = Data_buff+End_dta_inx;
        for (i=0;i<3;i++) {
            d_ptr->timer[i] = Timer[i];
            d_ptr->counter[i] = Counter[i];
        }
        if (++End_dta_inx >= DAT_BUF_SIZE)
            End_dta_inx = 0;                        // circular buffer
        (Data_buff+End_dta_inx)->misc = 0;          // init value
        if (Str_dta_inx == End_dta_inx) {
            if (++Str_dta_inx >= DAT_BUF_SIZE)
                Str_dta_inx = 0;
        }
}

void Init_ISR_func(void)
{
        if (Prot_setup.proc == PROC_ASYNC)
            ISR_table = ISR_async_func;
}
```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```
/* ------------------------- */
/*   File :  FUNC2.INC       */
/* ------------------------- */

char  *File_keyword[] = {
                        "RS-232C Monitor. Copyright (c), 1993. ,
                        "Setup",
                        "Receive Data",
                };

char   *Func_header[] = {  "",
                           " Load data  ,
                           " Save data ",
                         };

char    Load(char,char);
char    Save(char,char);

void    Load_prot_setup(void);
void    Save_prot_setup(void);

char    Load_data(char);
void    Start_file(void);
void    Brt_Setup(void);
void    Brt_run(void);

void    Diag_screen(void);

extern  int             fh;             // file handle
extern  unsigned int    Str_dta_inx;    //index of starting point data in data buffer
extern  unsigned int    End_dta_inx;    //index of ending point data in data buffer
extern  unsigned char   Disp_mode;      // current display mode
extern  SETUP           Prot_setup;     // parameter setup
extern  DATA_STRUC      *Data_buff;     // receive data buffer
extern  LIST_STRUC      *Str_list_tbl;  // start list table of data file
extern  LIST_STRUC      *Cur_list_tbl;  // current list table of data file
extern  LIST_STRUC      *End_list_tbl;  // last list table of data file
extern  WIN_STRUC       Win[MAX_WINDOW];
extern  MENU_STRUC      Menu[MAX_FUNC];

extern  void    Error(char*,...);
extern  char    Verify(char*,...);
extern  char    Append_list_table(long);
extern  char    True_charactor(unsigned char);
extern  void    Delete_list_table(LIST_STRUC*);
extern  char    Load_data_disk(unsigned int*,unsigned int*);
extern  char    *Get_filename(char,char,char,char,char*,char);
```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```
/* ------------------------------ *
/*   File :  FUNC2.C              *
/* ------------------------------ */

#include  <io.h>
#include  <conio.h>
#include  <stdio.h>
#include  <dir.h>
#include  <dos.h>
#include  <fcntl.h>
#include  <string.h>
#include  "constant.inc"
#include  "func2.inc"

char Load(char fc,char fn)        //fc=0 fr=
{
        char    *str;
        register unsigned char  n;
        register unsigned int   i,m;

        n = Menu[fc].xbar+2;
        m = Win[WINDOW2].top+5+fn;
        if ((str=Get_filename(n,m,n+WIN_WIDTH,m-2,Func_header[fn],fc)) == 0)
          return 0;

        if ((Fh=_open(str,O_RDONLY|O_BINARY))==-1) {
          Error("Can't open file %s.",str);
          return 0;
        }

        if (fc == FUNC_SETUP)
          n = Load_data(0);          // fc = Load data

        if (!n) Error("Invalid data in file %s.",str);
        return 1;
}

char Save(char fc,char fn)
{
        char    *str;
        struct  ffblk    ffblk;
        register signed   char  n;
        register unsigned int   i,m;

        n = Menu[fc].xbar+2;
        m = Win[WINDOW2].top+4+fn;
        if ((str=Get_filename(n,m,n+WIN_WIDTH,m-2,Func_header[fn],fc)) == 0)
            return 0;

        if ((findfirst(str,&ffblk,FA_ARCH | FA_RDONLY)) == '\0') {
            if (Verify("Overwrite %s [Y/N]?",str) < 1)
                return 0;
        }
        Start_file();
        return 1;
}

void Load_prot_setup(void)
{
        SETUP_ARRAY     *sa;
        register char   i;
```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```
        _read(Fh,&Prot_setup.proc,1);
        sa = Prot_setup.st[Prot_setup.proc];
        for (i=1;i<MAX_SETUP-4;i++)
            _read(Fh,&(sa+i)->val,1);          // load setup variable
}

void Save_prot_setup(void)
{
        SETUP_ARRAY     *sa;
        register char   i;

        _write(Fh,&Prot_setup.proc,1);         // type of protocol
        sa = Prot_setup.st[Prot_setup.proc];
        for (i=1;i<MAX_SETUP-4;i++)
            _write(Fh,&(sa+i)->val,1);         // save setup variable
}

char Load_data(char mode)
{
        DATA_STRUC      *d_ptr;
        char            temp1[40];
        char            temp2[15];
        signed   int    c;
        unsigned int    n,m;

        n = strlen(File_keyword[0]);
        _read(Fh,temp1,n);                     // get keyword
        temp1[n] = 0;
        n = strlen(File_keyword[2]);
        _read(Fh,temp2,strlen(File_keyword[2]));
        temp2[n] = 0;
        if (strcmp(temp1,File_keyword[0]) || strcmp(temp2,File_keyword[2]))
            return 0;                          // invalid file
        d_ptr = Data_buff;
        Load_prot_setup();
        Delete_list_table(Str_list_tbl);       // delete old table
        Str_list_tbl = 0;
        Append_list_table(tell(Fh));           // start of data section
        if (!mode)
            Disp_mode = (Prot_setup.st[Prot_setup.proc]+SETUP_DISP)->val;
        for (n=m=0;n<DAT_BUF_SIZE-1;) {
            _read(Fh,d_ptr,sizeof(DATA_STRUC));
            if (True_charactor((Data_buff+n)->misc)) {
                m++;
                d_ptr++;
                if (++n == (DAT_BUF_SIZE-1)/2) {
                    End_list_tbl->len = m;
                    if (!Append_list_table(tell(Fh)))
                        break;
                    m = 0;
                }
            }
            if ((c=eof(Fh)) != 0) {            // is end-of-file
                if (c == -1) {
                    Error("Error while reading data file.");
                    Fh = _close(Fh);
                }
                Delete_list_table(Str_list_tbl);
                Str_list_tbl = End_list_tbl = 0;
                break;
            }
        }
```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```
        Str_dta_inx = 0;
        End_dta_inx = n;
        Cur_list_tbl = Str_list_tbl;
        if (Fh > 0) {                   // scan data
            DATA_STRUC  p;

            End_list_tbl->len = m;
            for (m=0;;) {
                if ((c=_read(Fh,&p,sizeof(DATA_STRUC))) <= 0) {
                    if (c == -1)
                        Error("Error while reading data file.");
                    End_list_tbl->len = m;
                    break;
                }
                if (True_charactor(p.misc)) {
                    if (++m == (DAT_BUF_SIZE/2)) {
                        End_list_tbl->len = m;
                        if (!Append_list_table(tell(Fh)))
                            break;
                        m = 0;
                    }
                }
            }
        }
        return 1;
    }

    void    Brt_Setup(void)
    {
    }

    void    Brt_run(void)
    {
    }

    void    Diag_screen(void)
    {
    }
```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```
/* ------------------------- */
/*   File :  UTILITY.INC     */
/* ------------------------- */

char    Stack[5];                       // unget stack buffer
char    Stack_ptr=0;                    // unget stack pointer
char    Blank[] = "                :    // blank string
char    File_lst[13][100];              // listing of file-name
char    Fname[3][MAXPATH];              // 3 filename's buffer
char    *File_extend[1] = { "*.MON" };  // file extended name

char far*       Vid_buff;               // video buffer

char    Getch(void);
void    Ungetch(char);
void    Cursor_on(void);
void    Cursor_off(void);
char    *Get_name(char*);
void    Error(char*,...);
char    Verify(char*,...);
char    *File_select(char*);
void    File_list(char,char);
void    Putch(char,char,char);
void    Cputch(char,char,char,char);
void    Hili_filelist(char,int,char);
void    Gen_sound(unsigned,unsigned);
void    Write_block(char*,char,char,char,char,char);
char    *Get_filename(char,char,char,char,char*,char);

char    Append_list_table(long);
char    True_character(unsigned char);
void    Delete_list_table(LIST_STRUC*);

// ------------------------ external variable --------------------------

extern  unsigned char   Disp_mode;      // current display mode

extern  WIN_STYLE       Win_style[MAX_STYLE];
extern  WIN_ATTR        Win_attr[MAX_ATTR];     // window attribute
extern  WIN_STRUC       Win[MAX_WINDOW];
extern  MENU_STRUC      Menu[MAX_FUNC];
extern  LIST_STRUC      *Str_list_tbl;  // start list table of data file
extern  LIST_STRUC      *Cur_list_tbl;  // current list table of data file
extern  LIST_STRUC      *End_list_tbl;  // last list table of data file

extern  void    Write_header(char);
extern  void    Scroll_up(char,char,char,char,char,char);
extern  void    Make_window(char,char,char,char,char*,WIN_ATTR*,WIN_STYLE*,char*,char);
```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```
/* ------------------------- */
/*   File :  UTILITY.C       */
/* ------------------------- */

#include  <dir.h>
#include  <dos.h>
#include  <mem.h>
#include   <io.h>
#include  <alloc.h>
#include  <ctype.h>
#include  <conio.h>
#include  <stdio.h>
#include  <stdarg.h>
#include  <string.h>
#include  "constant.inc"
#include  "utility.inc"


// Get path & filename on window
// List filename if path name consited of '?' or '*' in DOS format
// past environment:block header,window location,border and text attribute
// return pointer to tempname[] if success or NULL if user escape

char *Get_filename(char left,char top,char right,char bottom,
                   char *header,char fc)
{
        char            done,row,
        char            *ptr,*str,*scrn;
        PATH_STRUC      path,toth;

        if ((scrn=(char *) malloc(1430)) == NULL) {
           Error("Not enough memory.");
           return NULL;
        }
        Make_window(left,top,right,bottom,header,
                    &Win_attr[PULL_ATTR],&Win_style[STY_EI],scrn,WINDOWS);
        done = fnsplit(Fname[0],path.drv,path.dir,path.fil,path.ext);
        strcpy(path.ext,File_extend[fc]);       // force ext. name
        if (!done)
           strcpy(path.fil,"*");
        fnmerge(Fname[0],path.drv,path.dir,path.fil,path.ext);
        if ((ptr=Get_name(Fname[0])) == NULL) { // file name stored in fname
           puttext(left,top,right,bottom,scrn);
           free(scrn);
           return (NULL);                       // cancel function
        }
        puttext(left,top,right,bottom,scrn);
        while(*(ptr) == ' ')
           ptr++;                               // skip blank

        ptr = strcpy(Fname[2],ptr);
        done = fnsplit(ptr,path.drv,path.dir,path.fil,path.ext);
        if (!(done & FILENAME))
           strcpy(path.fil,"*");
        strcpy(path.ext,File_extend[fc]);       // force ext. name

        if ((done & FILENAME) && !(done & WILDCARDS)) {
           free(scrn);
           fnmerge(Fname[0],path.drv,path.dir,path.fil,path.ext);
           return Fname[0];
        }
```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งไม่อนุญาตให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```
        if (!(done & DRIVE)) {
            path.drv[0] = 'A' + getdisk();
            strcpy(&path.drv[1],":");
        }

        if (*(path.dir) != '\\') {              // directory not start from root
            char temp1[50],temp2[50];

            getcurdir(path.drv[0]-'@',temp1);   // get current directory of spec. drive
            strcpy(temp2,path.dir);
            if (temp1[0])
                sprintf(path.dir,"\\%s\\%s",temp1,temp2);// make comolete path
            else sprintf(path.dir,"\\%s",temp2);
        }
        fnmerge(Fname[2],path.drv,path.dir,path.fil,path.ext);

        Make_window(8,11,72,21,Fname[2],&Win_attr[PULL_ATTR],
                                &Win_style[STYLE1],scrn,WINDOW5);
        ptr = File_select(Fname[2]);
        while (ptr && strchr(ptr,'\\')) {
            if ((str = strrchr(path.dir,'\\')) == 0)
                str = strcpy(path.dir,"\\");    // point to first block

            if (*ptr == '.')
                while (*(--str) != '\\' && strlen(path.dir)>1)//to mother directory
                    *str = NULL;
            else {
                while (*ptr != ' ' && *ptr != NULL) // to new subdir.
                    *(++str) = *ptr++;
                *(++str) = NULL;
            }
            fnmerge(Fname[2],path.drv,path.dir,path.fil,path.ext);
            ptr = File_select(Fname[2]);
        }
        if (ptr) {
            fnsplit(ptr,tpth.drv,tpth.dir,tpth.fil,tpth.ext);
            fnmerge(Fname[2],path.drv,path.dir,tpth.fil,tpth.ext);
            ptr = Fname[2];
        }
        puttext(8,11,72,21,scrn);       // restore file list location
        free(scrn);
        return ptr;
}

// select sub-directory,read only or archive file name on directory block
// function requires path included directory ,filename
// function returns array number if sucesses or NULL if escaped by user

char *File_select(char *pathprt)
{
        int             flprt,oldfl;
        char            temp[MAXPATH];
        char            count,done,flpage;
        PATH_STRUC      path;
        struct  ffblk   ffblk;

        count = 0;
        Write_header(WINDOW5);
        fnsplit(pathprt,path.drv,path.dir,path.fil,path.ext);
        fnmerge(temp,path.drv,path.dir,"*",".*");
        done = findfirst(temp,&ffblk,FA_DIREC); // get all sub-directory
        while (!done) {
```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```
    if (ffblk.ff_attrib == FA_DIREC && strcmp(ffblk.ff_name, ".") ) {
        strcpy(File_lst[count].ffblk.ff_name);
        strcat(File_lst[count],"\\");
        strncat(File_lst[count],Blank,12-strlen(File_lst[count]));
        count++;
    }
    done = findnext(&ffblk);
}
done = findfirst(pathprt,&ffblk,FA_ARCH);// get archive file
while (!done) {
    strcpy(File_lst[count],ffblk.ff_name);
    strncat(File_lst[count],Blank,12-strlen(ffblk.ff_name));
    done = findnext(&ffblk);
    count++;
}
gotoxy(2+Win[WINDOW5].left,1+Win[WINDOW5].top);
if  !count) {
    textattr(Win[WINDOW5].clr->norm_tx);
    cprintf("File not found. Press nay key.");
    getch();                    // wait any key
    return 0;
}
flpage =0;
flprt = 0;
File_list(flpage,count);
for (;;) {
    oldfl = flprt;
    Hili_filelist(flpage,flprt,Win[WINDOW5].clr->hili_tx);
    switch (getch()) {
        case 0:
            switch (getch()) {
                case PG_UP:
                    if (flpage   9) {
                        flpage-= 9;
                        flprt -= 36;
                        oldfl = flprt;
                        File_list(flpage,count);
                    }
                    else if (flpage > 0) {
                        flprt -= (flpage*4);
                        flpage = 0;
                        oldfl = flprt;
                        File_list(flpage,count);
                    }
                    break;

                case PG_DN:
                    if (flpage+8 < (count/4)) {
                        if (flprt+37 > count) {
                            while (flprt+4 < count)
                                flprt+=4;
                            flpage = (flprt/4)-7;
                        }
                        else {
                            flprt += 36;
                            flpage += 9;
                        }
                        oldfl = flprt;
                        File_list(flpage,count);
                    }
                    break;
```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```
case HOME:
    flprt = 0;
    if (flpage > 0) {
        flpage = 0;
        oldfl = 0;
        File_list(flpage,count);
    }
    break;

case END:
    oldfl = flprt = count-1;
    if (flpage+8 < count/4) {
        while (flpage+3 < count/4)
            flpage++;
        File_list(flpage,count);
    }
    break;

case ARW_RG:
    if (flprt < count-1) {
        if (flprt == (flpage*4)+35) {//at end of block ?
            flpage++;
            flprt++;
            oldfl = flprt;
            File_list(flpage,count);
        }
        else flprt++;
    }
    break;

case ARW_LF:
    if (flprt > 0) {
        if (flprt > (flpage*4)) {
            flprt--;
        }
        else {
            flprt--;
            flpage--;
            oldfl = flprt;
            File_list(flpage,count);
        }
    }
    break;

case ARW_UP:
    if (flprt-4 >= 0) {
        if (flprt-4 < flpage*4) {
            flpage--;
            flprt-=4;
            oldfl = flprt;
            File_list(flpage,count);
        }
        else flprt -= 4;
    }
    break;

case ARW_DN:
    if (flprt < count) {
        if (flprt+4 < count) {
            flprt += 4;
            if (flprt >= (flpage*4)+36) {
                flpage++;
```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```
                                    oldfl = flprt;
                                    File_list(flpage,count);
                                }
                            }
                            else if (flpage*4+36 < count) {
                                flpage++;
                                File_list(flpage,count);
                            }
                        }
                        break;
                    }
                    break;

                case ESCAPE:
                    return NULL;        // cancel function

                case ENTER:
                    return (File_list[flprt]);
            }
            if (count>=0) {             // file not found ?
                if (oldfl != flprt)
                    Hili_filelist(flpage,oldfl,Win[WINDOW5].clr->norm_tx);
            }
        }
}

// list file name in directory block
// flpt point to first filename to display . count is maximum filename list.

void File_list(char flpage,char count)
{
        register char   row,col,i;
        register int    flprt;

        flprt = flpage<<2;              // multiple 4
        for (row=i=0;i<36;i+=4,flprt++) {
            if (!(flprt%4))
                col=2,row++;
            else col+=16;

            textattr(Win[WINDOW5].clr->norm_tx);
            gotoxy(col+Win[WINDOW5].left,row+Win[WINDOW5].top);
            if (flprt < count)
                cprintf(" %-12s ",File_list[flprt]);
            else cputs(Blank);
        }
}

void Hili_filelist(char flpage,int flprt,char attr)
{
        register int    n;
        register char   row,col;

        textattr(attr);
        n = flprt-(flpage*4);
        row = n/4 +1;
        col = (n%4)*16 +2;
        gotoxy(col+Win[WINDOW5].left,row+Win[WINDOW5].top);
        cprintf(" %-12s ",File_list[flprt]);
}

//  Get_name() : get text in window and store in passed string
```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```
//      post  : cursor position on buffer
//      begin : point to first charector to display on the window
//  if successful return pointer to ptr,otherwise return NULL ,press ESC)

char *Get_name(char *str)
{
        char            winlen,row;
        char            col,attr;
        register char   c,edit;
        int             post,begin;
        char            temp[MAXPATH];

        edit = 0;
        begin = post = 0;
        strcpy(temp,str);
        row = Win[WINDOW5].top+1;
        col = Win[WINDOW5].left+2;
        winlen = Win[WINDOW5].right-Win[WINDOW5].left-3;
        attr = Win[WINDOW5].clr->hili_tx;
        Write_block(temp,0,winlen,row,col,attr);
        for (;;) {
            Cursor_on();
            gotoxy(col+post-begin,row);
            while (!kbhit() && !Stack_ptr);     // loop wait key here
            switch ((c=Getch())) {
                case 0:
                    switch(Getch()) {
                        case HOME:
                            edit = 1;
                            begin = post = 0;
                            break;

                        case END:
                            edit = 1;
                            post = strlen(temp);
                            if (post >= winlen)
                                begin = post-winlen+1;
                            else begin = 0;
                            break;

                        case INSERT:
                            strcpy(temp,str);
                            begin = post = edit = 0;
                            break;

                        case ARW_LF:
                            edit = 1;
                            if (post>0) {
                                post--;
                                if (post<begin) {
                                    if (begin>0) {
                                        begin--;
                                    }
                                }
                            }
                            break;

                        case ARW_RG:
                            edit = 1;
                            if (post < strlen(temp)) {
                                post++;
                                if (post > begin+winlen-1) {
```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```
                        begin++;
                    }
                }
                break;

            case DELETE:
                edit = 1;
                if (post < strlen(temp)) {
                    strcpy(temp+post,temp+post+1);
                    if (begin>0)
                        begin--;
                }
                break;
        }
        break;

    case ESCAPE:
        Cursor_off();
        return NULL;

    case BCKSPC:                // delete one character
        edit = 1;
        if (post>0) {           // goback 1 position
            post--;
            if (begin>0)
                begin--;
            if (strlen(temp)>post)
                strcpy(temp+post,temp+post+1);
        }
        break;

    case ENTER:
        c=0;
        while (temp[c]) {
            temp[c] = toupper(temp[c]); // to upper case
            c++;
        }
        Cursor_off();
        return strcpy(str,temp);        // store file name

    default:
        if (!edit) {
            edit = 1;
            temp[0] = 0;                // clear buffer
        }
        if (c>31 && c<127 && strlen(temp) < MAXPATH-1) {
            if (post < strlen(temp)) {
                char n;
                n = strlen(temp);
                movmem(temp+post,temp+post+1,n-post+1);
            }
            else temp[post+1] = 0;
            temp[post] = c;
            Ungetch(ARW_RG);
            Ungetch(0);
        }
        break;
    }
    if (!edit)
        attr = Win[WINDOW5].clr->hili_tx;
    else attr = Win[WINDOW5].clr->norm_tx;
    Write_block(temp+begin,begin,winlen,row,col,attr);
```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```
        }
    }

void write_block(char *str,char begin,char len,char row,
                 char col,char attr)
{
        register char   i,n;

        i = 0;
        n = col;
        Scroll_up(col,row,col+len,row,attr,1);  // fill blank
        while (i <len && * str+i) != NULL) {
            Cputch(col,row,attr,*(str+i));      // fill data
            col++;
            i++;
        }
        textcolor(0x04);
        gotoxy(n-1,row);
        if (begin)
            putch(174);                         // <
        else putch(0xFF);

        gotoxy(n+len,row);
        if (*(str+i) != NULL)
            putch(175);                         // >
        else putch(0xFF);
}

void Error(char *format,...)
{
        va_list  argptr;
        char     *scrn;
        char     temp[110];
        char     line,len,left,right;

        va_start(argptr,format);
        len = vsprintf(temp,format,argptr)+11;
        va_end(argptr);
        scrn = (char*) malloc (520);
        if (len>65) {
            char temp1[MAXPATH];
            line = 1;
            left = 8;
            right = 75;
            strncpy(temp1,&temp[65],len-76);
            temp1[len-76] = NULL;
            strcpy(&temp[66],temp1);
            temp[65] = NULL;            // two section
        }
        else {
            line=0;
            left = (80-len)>>1;
            right = left+len+2;
        }
        Make_window(left,11,right,13+line," Error ",&Win_attr[EROR_ATTR],
                                        &Win_style[STYLE3],scrn,WINDOW5);
        textattr(Win_attr[EROR_ATTR].norm_tx);
        gotoxy(left+2,12);
        cprintf("%s",temp);
        if (line) {
            gotoxy(left+2,13);
            cprintf("%s",&temp[66]);
```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งไม่อนุญาตให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```
        }
        cprintf(" Press ");
        textattr(Win_attr[EROR_ATTR].hot_key);
        cprintf("ESC.");
        while (getch() != 27);          // For exit press ESC key
        puttext(left,11,right,13+line,scrn);
        free (scrn);
}

// Return value : YES     1,
//                NO      0,
//                ESCAPE -1

char Verify(char *format,...)
{
        va_list argptr;
        char    scr[400];
        char    temp[MAXPATH];
        char    ch,left,right;

        va_start(argptr,format);
        ch = vsprintf(temp,format,argptr);
        va_end(argptr);
        left = (80-ch)>>1;
        right = left+ch+1;
        Make_window(left,8,right,10," Verify ",
                &Win_attr[VERF_ATTR],&Win_style[STYLE1],scr,WINDOW5);
        textattr(Win_attr[VERF_ATTR].norm_tx);
        gotoxy(left+1,9);
        ch = cprintf("%s",temp);
        ch = strstr(temp,"Y/N") - temp + 1;
        textattr(Win[WINDOW5].clr->hot_key);
        gotoxy(left+ch,9);  putch('Y');
        gotoxy(left+ch+2,9);  putch('N');

        while ((ch=toupper(getch())) != ESCAPE) {
            if (ch == 'Y' || ch == 'N')
                break;
        }
        if (ch == 'n' || ch == 'N') {
            ch = 0;
        }
        else {
            if (ch == ESCAPE)
                ch = -1;
            else ch = 1;
        }
        puttext(left,8,right,10,scr);
        return ch;
}

void Cursor_off(void)
{
        _AH = 1;
        _CH = 0x20;
        _CL = 0x07;
        geninterrupt(0x10);
}

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

void Cursor_on(void)
{
        _AH = 1;
```

```
        _CH = 0x07;
        _CL = 0x08;
        geninterrupt(0x10);
}

void Gen_sound(unsigned freq,unsigned del)
{
        sound(freq);
        delay(del);
        nosound();
}

char Getch(void)
{
        if (Stack_ptr>0)
            return Stack[--Stack_ptr];
        return getch();
}

void Ungetch(char c)
{
        Stack[Stack_ptr++] = c;
}

void Putch(char col,char row,char ch)
{
        *(Vid_buff+((row-1)*160)+((col-1)<<1)) = ch;
}

void Cputch(char col,char row,char attr,char ch)
{
        register unsigned int  p;

        p = (row-1 *160 + ((col-1)<<1);
        *(Vid_buff+p++) = ch;
        *(Vid_buff+p) = attr;
}

char True_charactor(unsigned char c)
{
        switch(Disp_mode) {
            case DISP_DTE:
                if ((c & MISC_DTE) && !(c & MISC_LEAD))
                    return 1;
                else return 0;

            case DISP_DCE:
                if (!(c & MISC_DTE) && !(c & MISC_LEAD))
                    return 1;
                else return 0;

            case DISP_DATA:
                if (!(c & MISC_LEAD))
                    return 1;
                else return 0;

            case DISP_STAT:
                return 1;
        }
        return 0;
}
```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```
char Append_list_table(long t)
{
        LIST_STRUC      *p;

        gotoxy(1,25);
        cprintf("free %u %lu",coreleft(),farcoreleft());
        if ((p=(LIST_STRUC *)malloc(sizeof(LIST_STRUC))) != NULL) {
            if (Str_list_tbl == NULL) {         // first list
                Str_list_tbl = p;
                p->prior = NULL;
            }
            else {
                End_list_tbl->next = p;
                p->prior = End_list_tbl;
            }
            End_list_tbl = p;
            p->next = NULL;
            p->ptr = t;
            return 1;
        }
        Error("Big file in fit memory, file is truncated. ");
        return 0;
}

void Delete_list_table(LIST_STRUC *p)
{
        if (p != NULL) {
            p = p->next;
            free(p);
            Delete_list_table(p);
        }
}
```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

ภาคผนวก ข.

8274 DATA SHEET



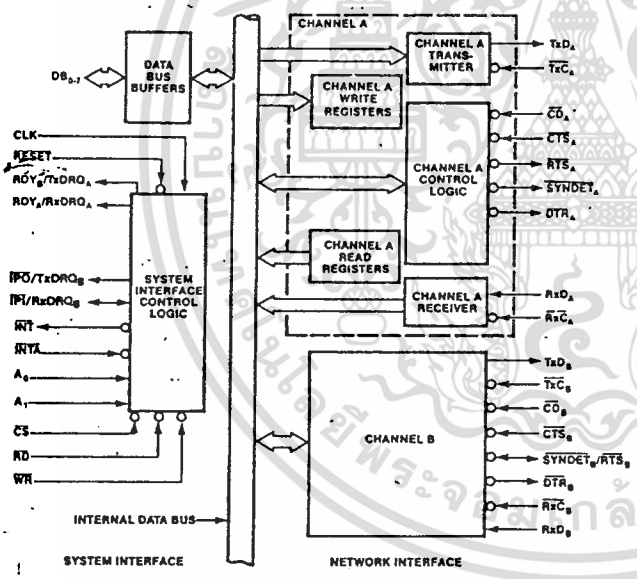
เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

8274
MULTI-PROTOCOL SERIAL CONTROLLER (MPSC)

- Asynchronous, Byte Synchronous and Bit Synchronous Operation
- Two Independent Full Duplex Transmitters and Receivers
- Fully Compatible with 8048, 8051, 8085, 8088, 8086, 80188 and 80186 CPU's; 8257 and 8237 DMA Controllers; and 8089 I/O Proc.
- 4 Independent DMA Channels
- Baud Rate: DC to 880K Baud
- Asynchronous:
  - 5–8 Bit Character; Odd, Even, or No Parity; 1, 1.5 or 2 Stop Bits
  - Error Detection: Framing, Overrun, and Parity
- Byte Synchronous:
  - Character Synchronization, Int. or Ext.
  - One or Two Sync Characters
  - Automatic CRC Generation and Checking (CRC-16)
  - IBM Bisync Compatible
- Bit Synchronous:
  - SDLC/HDLC Flag Generation and Recognition
  - 8 Bit Address Recognition
  - Automatic Zero Bit Insertion and Deletion
  - Automatic CRC Generation and Checking (CCITT-16)
  - CCITT X.25 Compatible
- Available in EXPRESS and Military

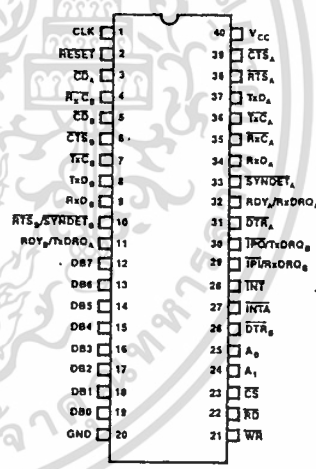
The Intel 8274 Multi-Protocol Series Controller (MPSC) is designed to interface High Speed Communications Lines using Asynchronous, IBM Bisync, and SDLC/HDLC protocol to Intel microcomputer systems. It can be interfaced with Intel's MCS-48, -85, -51; iAPX-86, -88, -186 and -188 families, the 8237 DMA Controller, or the 8089 I/O Processor in polled, interrupt driven, or DMA driven modes of operation.

The MPSC is a 40 pin device fabricated using Intel's High Performance HMOS Technology.



170102-1

Figure 1. Block Diagram



170102-2

Figure 2. Pin Configuration

November 1986
Order Number: 170102-002

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

Table 1. Pin Description

Symbol	Pin No.	Type	Name and Function
CLK	1	I	**CLOCK:** System clock, TTL compatible.
$\overline{RESET}$	2	I	**RESET:** A low signal on this pin will force the MPSC to an idle state. TxD_A and TxD_B are forced high. The modem interface output signals are forced high. The MPSC will remain idle until the control registers are initialized. Reset must be true for one complete CLK cycle.
$\overline{CD}_A$	3	I	**CARRIER DETECT (CHANNEL A):** This interface signal is supplied by the modem to indicate that a data carrier signal has been detected and that a valid data signal is present on the RxD_A line. If the auto enable control is set the 8274 will not enable the serial receiver until $\overline{CD}_A$ has been activated.
$\overline{RxC}_B$	4	I	**RECEIVE CLOCK (CHANNEL B):** The serial data are shifted into the Receive Data input (RxD_B) on the rising edge of the Receive Clock.
$\overline{CD}_B$	5	I	**CARRIER DETECT (CHANNEL B):** This interface signal is supplied by the modem to indicated that a data carrier signal has been detected and that a valid data signal is present on the RxD_B line. If the auto enable control is set the 8274 will not enable the serial receiver until $\overline{CD}_B$ has been activated.
$\overline{CTS}_B$	6	I	**CLEAR TO SEND (CHANNEL B):** This interface signal is supplied by the modem in response to an active $\overline{RTS}$ signal. $\overline{CTS}$ indicates that the data terminal/computer equipment is permitted to transmit data. In addition, if the auto enable control is set, the 8274 will not transmit data bytes until $\overline{CTS}$ has been activated.
$\overline{TxC}_B$	7	I	**TRANSMIT CLOCK (CHANNEL B):** The serial data are shifted out from the Transmit Data output (TxD_B) on the falling edge of the Transmit Clock.
TxD_B	8	O	**TRANSMIT DATA (CHANNEL B):** This pin transmits serial data to the communications channel (Channel B).
RxD_B	9	I	**RECEIVE DATA (CHANNEL B):** This pin receives serial data from the communications channel (Channel B).
$\overline{SYNDET}_B$ /$\overline{RTS}_B$	10	I/O	**SYNCHRONOUS DETECTION (CHANNEL B):** This pin is used in byte synchronous mode as either an internal sync detect (output) or as a means to force external synchronization (input). In SDLC mode, this pin is an output indicating Flag detection. In asynchronous mode it is a general purpose input (Channel B). **REQUEST TO SEND (CHANNEL B):** General purpose output, generally used to signal that Channel B is ready to send data. When the RTS bit is reset in asynchronous mode, the signal does not go inactive (High) until the transmitter is empty. $\overline{SYNDET}_B$ or $\overline{RTS}_B$ selection is done by WR2; D7. (Channel A).

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

Table 1. Pin Description (Continued)

Symbol	Pin No.	Type	Name and Function
RDY_B/ $TxDRQ_A$	11	O	**READY (CHANNEL B)/TRANSMITTER DMA REQUEST (CHANNEL A):** In mode 0 this pin is RDY_B and is used to synchronize data transfers between the processor and the MPSC (Channel B). In modes 1 and 2 this pin is $TxDRQ_A$ and is used by the Channel A transmitter to request a DMA transfer.
DB7	12	I/O	**DATA BUS:** The Data Bus lines are bidirectional three state lines which interface with the system's Data Bus.
DB6	13		
DB5	14		
DB4	15		
DB3	16		
DB2	17		
DB1	18		
DB0	19		
GND	20		GROUND.
V_{CC}	40		POWER: +5V Supply
$\overline{CTS}_A$	39	I	**CLEAR TO SEND (CHANNEL A):** This interface signal is supplied by the Modem in response to an active $\overline{RTS}$ signal. $\overline{CTS}$ indicates that the data terminal/computer equipment is permitted to transmit data. In addition, if the auto enable control is set, the 8274 will not transmit data bytes until $\overline{CTS}$ has been activated.
$\overline{RTS}_A$	38	O	**REQUEST TO SEND (CHANNEL A):** General purpose output commonly used to signal that Channel A is ready to send data. When the RTS bit is reset in asynchronous mode, the signal does not go inactive (High) until the transmitter is empty.
TxD_A	37	O	**TRANSMIT DATA (CHANNEL A):** This pin transmits serial data to the commmunications channel (Channel A).
$\overline{TxC}_A$	36	I	**TRANSMIT CLOCK (CHANNEL A):** The serial data are shifted out from the Transmit Data output (TxD_A) on the falling edge of the Transmit Clock.
$\overline{RxC}_A$	35	I	**RECEIVE CLOCK (CHANNEL A):** The serial data are shifted into the Receive Data input (RxD_A) on the rising edge of the Receive Clock.
RxD_A	34	I	**RECEIVE DATA (CHANNEL A):** This pin receives serial data from the communications channel (Channel A).
$\overline{SYNDET}_A$	33	I/O	**SYNCHRONOUS DETECTION (CHANNEL A):** This pin is used in byte synchronous mode as either an internal sync detect (output) or as a means to force external synchronization (input). In SDLC mode, this pin is an output indicating flag detection. In asynchronous mode it is a general purpose input (Channel A).

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

Table 1. Pin Description (Continued)

Symbol	Pin No.	Type	Name and Function
RDY_A/ $RxDRQ_A$	32	O	**READY:** In mode 0 this pin is RDY_A and is used to synchronize data transfers between the processor and the MPSC (Channel A). In modes 1 and 2 this pin is $RxDRQ_A$ and is used by the channel A receiver to request a DMA transfer.
$\overline{DTR}_A$	31	O	**DATA TERMINAL READY (CHANNEL A):** General purpose output.
$\overline{IPO}$/ $TxDRQ_B$	30	O	**INTERRUPT PRIORITY OUT/TRANSMITTER DMA REQUEST (CHANNEL B):** In modes 0 and 1, this pin is Interrupt Priority Out. It is used to establish a hardware interrupt priority scheme with $\overline{IPI}$. It is low only if $\overline{IPI}$ is low and the controlling processor is not servicing an interrupt from this MPSC. In mode 2 it is $TxDRQ_B$ and is used to request a DMA cycle for a transmit operation (Channel B).
$\overline{IPI}$/ $RxDRQ_B$	29	I/O	**INTERRUPT PRIORITY IN/RECEIVER DMA REQUEST (CHANNEL B):** In modes 0 and 1, $\overline{IPI}$ is Interrupt Priority In. A low on $\overline{IPI}$ means that no higher priority device is being serviced by the controlling processor's interrupt service routine. In mode 2 this pin is $RxDRQ_B$ and is used to request a DMA cycle for a receive operation (Channel B).
$\overline{INT}$	28	O	**INTERRUPT:** The interrupt signal indicates that the highest priority internal interrupt requires service (open collector). Priority can be resolved via an external interrupt controller or a daisy-chain scheme.
$\overline{INTA}$	27	I	**INTERRUPT ACKNOWLEDGE:** This Interrupt Acknowledge signal allows the highest priority interrupting device to generate an interrupt vector. This pin must be pulled high (inactive) in non-vector mode.
$\overline{DTR}_B$	26	O	**DATA TERMINAL READY (CHANNEL B):** This is a general purpose output.
A_0	25	I	**ADDRESS:** This line selects Channel A or B during data or command transfers. A low selects Channel A.
A_1	24	I	**ADDRESS:** This line selects between data or command information transfer. A low means data.
$\overline{CS}$	23	I	**CHIP SELECT:** This signal selects the MSPC and enables reading from or writing into registers.
$\overline{RD}$	22	I	**READ:** Read controls a data byte or status byte transfer from the MPSC to the CPU.
$\overline{WR}$	21	I	**WRITE:** Write controls transfer of data or commands to the MPSC.

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

RESET

When the 8274 $\overline{\text{RESET}}$ line is activitated, both MPSC channels enter the idle state. The serial output lines are forced to the marking state (high) and the modem interface signals ($\overline{\text{RTS}}$, $\overline{\text{DTR}}$) are forced high. In addition, the pointers registers are set to zero.

GENERAL DESCRIPTION

The Intel 8274 Multi-Protocol Serial Controller is a microcomputer peripheral device which supports Asynchronous, Byte Synchronous (Monosync, IBM Bisync), and Bit Synchronous (ISO's HDLC, IBM's SDLC) protocols. This controller's flexible architecture allows easy implementation of many variations of these three protocols with low software and hardware overhead.

The Multi-Protocol Serial controller (MPSC) implements two independent serial receiver/transmitter channels.

The MPSC supports several microprocessor interface options: Polled, Wait, Interrupt driven and DMA driven. The MPSC is designed to support INTEL's MCS-85 and iAPX 86, 88, 186, 188 families.

FUNCTIONAL DESCRIPTION

Additional information on Asynchronous and Synchronous Communications with the 8274 is available respectively in the Applications Notes AP 134 and AP 145.

Command, parameter, and status information is stored in 21 registers within the MPSC (8 writable registers for each channel, 2 readable registers for Channel A and 3 readable registers for Channel B).

In the following discussion, the writable registers will be referred to as WR0 through WR7 and the readable registers will be referred to as RR0 through RR2.

This section of the data sheet describes how the Asynchronous and Synchronous protocols are implemented in the MPSC. It describes general considerations, transmit operation, and receive operation for Asynchronous, Byte Synchronous, and Bit Synchronous protocols.

ASYNCHRONOUS OPERATIONS

Transmitter/Receiver Initialization

(See Detailed Command Description Section for complete information)

In order to operate in asynchronous mode, each MPSC channel must be initialized with the following information:

1. Transmit/Receive Clock Rate. This parameter is specified by bits 6 and 7 of WR4. The clock rate may be set to 1, 16, 32, or 64 times the data-link bit rate. If the X1 clock mode is selected, the bit synchronization must be accomplished externally.
2. Number of Stop Bits. This parameter is specified by bits 2 and 3 of WR4. The number of stop bits may be set to 1, 1½, or 2.
3. Parity Selection. Parity may be set for odd, even, or no parity by bits 0 and 1 of WR4.
4. Receiver Character Length. This parameter sets the length of received characters to 5, 6, 7, or 8 bits. This parameter is specified by bits 6 and 7 of WR3.
5. Receiver Enable. The serial-channel receiver operation may be enabled or disabled by setting or clearing bit 0 of WR3.
6. Transmitter Character Length. This parameter sets the length of transmitted characters to 5, 6, 7, or 8 bits. This parameter is specified by bits 5 and 6 of WR5. Characters of less than 5 bits in length may be transmitted by setting the transmitted length to five bits (set bits 5 and 6 of WR5 to 0).

   The MPSC then determines the actual number of bits to be transmitted from the character data byte. The bits to be transmitted must be right justified in the data byte, the next three bits must be set to 0 and all remaining bits must be set to 1. The following table illustrates the data formats for transmission of 1 to 5 bits of data.

Byte Written								Number of Bits Transmitted
D7	D6	D5	D4	D3	D2	D1	D0	(Character Length)
1	1	1	1	0	0	0	c	1
1	1	1	0	0	0	c	c	2
1	1	0	0	0	c	c	c	3
1	0	0	0	c	c	c	c	4
0	0	0	c	c	c	c	c	5

7. Transmitter Enable. The serial channel transmitter operation may be enabled or disabled by setting or clearing bit 3 of WR5.
8. Interrupt Mode. Specified by bits 3 and 4 of WR1.

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

For data transmission via a modem or RS-232-C interface, the following information must also be specified:

1. The Request To Send (RTS) (WR5; D1) and Data Terminal Ready (DTR) (WR5; D7) bits must be set along with the Transmit Enable bit (WR5; D3).
2. Auto Enable may be set to allow the MPSC to automatically enable the channel transmitter when the clear-to-send signal is active and to automatically enable the receiver when the carrier-detect signal is active. However, the Transmit Enable bit (WR3; D3) and Receive Enable bit (WR3; D1) must be set in order to use the Auto Enable mode. Auto Enable is controlled by bit 5 of WR3.

When loading Initialization parameters into the MPSC, WR4 information must be written before the WR1, WR3, WR5 parameters commands.

During initialization, it is desirable to guarantee that the external/status latches reflect the latest interface information. Since up to two state changes are internally stored by the MPSC, at least two Reset External/Status Interrupt commands must be issued. This procedure is most easily accomplished by simply issuing this reset command whenever the pointer register is set during initialization.

An MPSC initialization procedure (MPSCRXINIT) for asynchronous communication is listed in Intel Application Note AP 134.

TRANSMIT

The transmit function begins when the Transmit Enable bit (WR5; D3) is set. The MPSC automatically adds the start bit, the programmed parity bit (odd, even or no parity) and the programmed number of stop bits (1, 1.5 or 2 bits) to the data character being transmitted. 1.5 stop bits option must be used with X16, X32 or X64 clock options only. The data character is transmitted least significant bit first.

The serial data are shifted out from the Transmit Data (TxD) output on the falling edge of the Transmit Clock (TxC) input at a rate programmable to 1, 1/16th, 1/32nd, or 1/64th of the clock rate supplied to the TxC input.

The TxD output is held high when the transmitter has no data to send, unless, under program control, the Send Break (WR5; D4) command is issued to hold the TxD low.

If the External/Status Interrupt bit (WR1; D0) is set, the status of $\overline{CD}$, $\overline{CTS}$ and $\overline{SYNDET}$ are monitored and, if any changes occur for a period of time greater than the minimum specified pulse width, an interrupt is generated. $\overline{CTS}$ is usually monitored using this interrupt feature (e.g., Auto Enable option).

The Transmit Buffer Empty bit (RR0; D2) is set by the MPSC when the data byte from the buffer is loaded in the transmit shift register. Data should be written to the MPSC only when the Tx buffer becomes empty to prevent overwriting.

Receive

The receive function begins when the Receive Enable (WR3; D0) bit is set. If the Auto Enable (WR3; D5) option is selected, then Carrier Detect ($\overline{CD}$) must also be low. A valid start bit is detected if a low persists for at least ½ bit time on the Receive Data (RxD) input.

The data is sampled at mid-bit time, on the rising edge of RxC, until the entire character is assembled. The receiver inserts 1's when a character is less than 8 bits. If parity (WR4; D0) is enabled and the character is less than 8 bits the parity bit is not stripped from the character.

Error Reporting

The receiver also stores error status for each of the 3 data characters in the data buffer. Three error conditions may be encountered during data reception in the asynchronous mode:

1. **Parity.** If parity bits are computed and transmitted with each character and the MPSC is set to check parity (bit 0 in WR4 is set), a parity error will occur whenever the number of "1" bits within the character (including the parity bit) does not match the odd/even setting of the parity check flag (bit 1 in WR4). When a parity error is detected, the parity error flag (RR1; D4) is set and remains set until it is reset by the Error Reset command (WR0; D5, D4, D3).
2. **Framing.** A framing error will occur if a stop bit is not detected immediately following the parity bit (if parity checking is enabled) or immediately following the most-significant data bit (if parity checking is not enabled). When a Framing Error is detected, the Framing Error bit (RR1; D6) is set and remains set until reset by the Error Reset Command (WR0; D5, D4, D3). The detection of a Framing Error adds an additional ½ bit time to the character time so the Framing Error is not interpreted as a new start bit.
3. **Overrun.** If the CPU fails to read a data character while more than three characters have been received, the Receive Overrun bit (RR1; D5) is set. When this occurs, the fourth character assembled replaces the third character in the receive buffers. Only the overwritten character is flagged with the Receive Overrun bit. The Receive Overrun bit (RR1; D5) is reset by the Error Reset command (WR0; D5, D4, D3).

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

External/Status Latches

The MPSC continuously monitors the state of five external/status conditions:

1. CTS—clear-to-send input pin.
2. CD—carrier-detect input pin.
3. SYNDET—sync-detect input pin. This pin may be used as a general-purpose input in the asynchronous communication mode.
4. BREAK—a break condition (series of space bits on the receiver input pin).
5. TxUNDERRUN/EOM—Transmitter Underrun/End of Message.

A change of state in any of these monitored conditions will cause the associated status bit in RR0 to be latched (and optionally cause an interrupt).

If the External/Status Interrupt bit (WR1; D0) is enabled, Break Detect (RR0; D7) and Carrier Detect (RR0; D3) will cause an interrupt. Reset External/Status interrupts (WR0; D5, D4, D3) will clear Break Detect and Carrier Detect bits if they are set.

Command, parameter, and status information is stored in 21 registers within the MPSC (8 writable registers for each channel, 2 readable registers for Channel A and 3 readable registers for Channel B). They are all accessed via the command ports.

An internal pointer register selects which of the command or status registers will be read or written during a commmand/status access of an MPSC channel.

After reset, the contents of the pointer registers are zero. The first write to a command register causes the data to be loaded into Write Register 0 (WR0). The three least significant bits of WR0 are loaded into the Command/Status Pointer. The next read or write operation accesses the read or write register selected by the pointer. The pointer is reset after the read or write operation is completed.

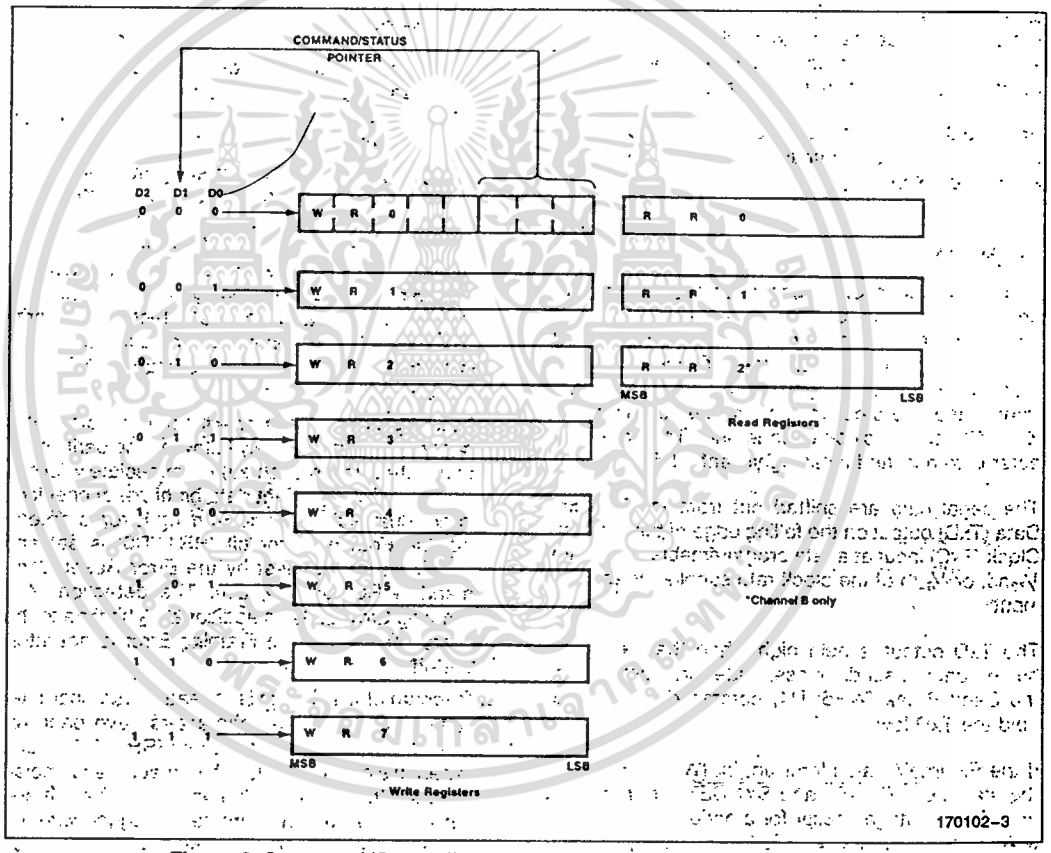


Figure 3. Command/Status Register Architecture (each serial channel)

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

Asynchronous Mode Register Setup

	D7	D6	D5	D4	D3	D2	D1	D0
WR3	00 Rx 5 b/char 01 Rx 7 b/char 10 Tx 6 b/char 11 Rx 8 b/char		AUTO ENABLE	0	0	0	0	Rx ENABLE
WR4	00 X1 Clock 01 X16 Clock 10 X32 Clock 11 X64 Clock		0	0	01 1 STOP BIT 10 1½ STOP BITS 11 2 STOP BITS		EVEN/ $\overline{\text{ODD}}$ PARITY	PARITY ENABLE
WR5	DTR	00 Tx ≤ 5 b/char 01 Tx 7 b/char 10 Tx 6 b/char 11 Tx 8 b/char		SEND BREAK	Tx ENABLE	0	RTS	0

SYNCHRONOUS OPERATION—MONOSYNC, BISYNC

General

The MPSC must be initialized with the following parameters: odd or even parity (WR4; D1, D0), X1 clock mode (WR4; D7, D6), 8- or 16-bit sync character (WR4; D5, D4), CRC polynomial (WR5; D2), Transmitter Enable (WR5; D3), interrupt modes (WR1, WR2), transmit character length (WR5; D6, D5) and receive character length (WR3; D7, D6). WR4 parameters must be written before WR1, WR3, WR5, WR6 and WR7.

The data is transmitted on the falling edge of the Transmit Clock, ($\overline{\text{TxC}}$) and is received on the rising edge of Receive Clock ($\overline{\text{RxC}}$). The X1 clock is used for both transmit and receive operations for all three sync modes: Mono, Bi and External.

Transmit Set-Up—Monosync, Bisync

Transmit data is held high after channel reset, or if the transmitter is not enabled. A break may be programmed to generate a spacing line that begins as soon as the Send Break (WR5; D4) bit is set. With the transmitter fully initialized and enabled, the default condition is continuous transmission of the 8- or 16-bit sync character.

Using interrupts for data transfer requires that the Transmit Interrupt/DMA Enable bit (WR1; D1) be set. An interrupt is generated each time the transmit buffer becomes empty. The interrupt can be satisfied either by writing another character into the transmitter or by resetting the Transmitter Interrupt/DMA Pending latch with a Reset Transmitter Inter-

Synchronous Mode Register Setup—Monosync, Bisync

	D7	D6	D5	D4	D3	D2	D1	D0
WR3	00 Rx 5 b/char 01 Rx 7 b/char 10 Tx 6 b/char 11 Rx 8 b/char		AUTO ENABLE	ENTER HUNT MODE	Rx CRC ENABLE	0	SYNC CHAR LOAD INHIBIT	Rx ENABLE
WR4	0	0	00 8 bit Sync 01 16 bit Sync 11 Ext Sync		0	0	EVEN/ $\overline{\text{ODD}}$ PARITY	PARITY ENABLE
WR5	DTR	00 Tx ≤ 5 b/char 01 Tx 7 b/char 10 Tx 6 b/char 11 Tx 8 b/char		SEND BREAK	Tx ENABLE	1 (SELECTS CRC-16)	RTS	Tx CRC ENABLE

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

rupt/DMA Pending Command (WR0; D5, D4, D3). If nothing more is written into the transmitter, there can be no further Transmit Buffer Empty interrupt, but this situation does cause a Transmit Underrun condition (RR0; D6).

Data Transfers using the RDY signal are for software controlled data transfers such as block moves. RDY tells the CPU that the MPSC is not ready to accept/provide data and that the CPU must extend the output/input cycle. DMA data transfers use the TxDRQ A/B signals which indicate that the transmit buffer is empty, and that the MPSC is ready to accept the next data character. If the data character is not loaded into the MPSC by the time the transmit shift register is empty, the MPSC enters the Transmit Underrun condition.

The MPSC has two programmable options for solving the transmit underrun condition: it can insert sync characters, or it can send the CRC characters generated so far, followed by sync characters. Following a chip or channel reset, the Transmit Underrun/EOM status bit (RR0; D6) is in a set condition allowing the insertion of sync characters when there is no data to send. The CRC is not calculated on these automatically inserted sync characters. When the CPU detects the end message, a Reset Transmit Underrun/EOM command can be issued. This allows CRC to be sent when the transmitter has no data to send.

In the case of sync insertion, an interrupt is generated only after the first automatically inserted sync character has been loaded in the Transmit Shift Register. The status register indicates the Transmit Underrun/EOM bit and the Transmit Buffer Empty bit are set.

In the case of CRC insertion, the Transmit Underrun/EOM bit is set and the Transmit Buffer Empty bit is reset while CRC is being sent. When CRC has been completely sent, the Transmit Buffer Empty status bit is set and an interrupt is generated to indicate to the CPU that another message can begin (this interrupt occurs because CRC has been sent and sync has been loaded into the Tx Shift Register). If no more messages are to be sent, the program can terminate transmission by resetting RTS, and disabling the transmitter (WR5; D3).

Bisync CRC Generation. Setting the Transmit CRC enable bit (WR5; D0) indicates CRC accumulation when the program sends the first data character to the MPSC. Although the MPSC automatically transmits up to two sync characters (16 bit sync), it is wise to send a few more sync characters ahead of the message (before enabling Transmit CRC) to ensure synchronization at the receiving end.

The Transmit CRC Enable bit can be changed on the fly any time in the message to include or exclude a particular data character from CRC accumulation. The Transmit CRC Enable bit should be in the desired state when the data character is loaded into the transmit shift register. To ensure this bit in the proper state, the Transmit CRC Enable bit must be issued before sending the data character to the MPSC.

Transmit Transparent Mode. Transparent mode (Bisync protocol) operation is made possible by the ability to change Transmit CRC Enable on the fly and by the additional capability of inserting 16 bit sync characters. Exclusion of DLE characters from CRC calculation can be achieved by disabling CRC calculation immediately preceding the DLE character transfer to the MPSC.

In the transmit mode, the transmitter always sends the programmed number of sync bits (8 or 16) (WR4; D5, D4). When in the Monosync mode, the transmitter sends from WR6 and the receiver compares against WR7. One or two CRC polynomials, CRC 16 or SDLC, may be used with synchronous modes. In the transmit initialization process, the CRC generator is initialized by setting the Reset Transmit CRC Generator command (WR0; D7, D6).

The External/Status interrupt (WR1; D0) mode can be used to monitor the status of the $\overline{CTS}$ input as well as the Transmit Underrun/EOM latch. Optionally, the Auto Enable (WR3; D5) feature can be used to enable the transmitter when $\overline{CTS}$ is active. The first data transfer to the MPSC can begin when the External/Status interrupt (CTS (RR0; D5) status bit set) occurs following the Transmit Enable command (WR5; D3).

Receive

After a channel reset, the receiver is in the Hunt phase, during which the MPSC looks for character synchronization. The Hunt begins only when the receiver is enabled and data transfer begins only when character synchronization has been achieved. If character synchronization is lost, the hunt phase can be re-entered by writing the Enter Hunt Phase (WR3; D4) bit. The assembly of received data continues until the MPSC is reset or until the receiver is disabled (by command or by $\overline{CD}$ while in the Auto Enables mode) or until the CPU sets the Enter Hunt Phase bit. Under program control, all the leading sync characters of the message can be inhibited from loading the receive buffers by setting the Sync Character Load Inhibit (WR3; D1) bit. After character synchronization is achieved the assembled characters are transferred to the receive data FIFO. After

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

receiving the first data character, the Sync Character Load Inhibit bit should be reset to zero so that all characters are received, including the sync characters. This is important because the received CRC may look like a sync character and not get received.

Data may be transferred with or without interrupts. Transferring data without interrupts is used for a purely polled operation or for off-line conditions. There are two interrupt modes available for data transfer: Interrupt on First Character Only and Interrupt on Every Character.

Interrupt on First Character Only mode is normally used to start a polling loop, a block transfer sequence using RDY to synchronize the CPU to the incoming data rate, or a DMA transfer using the RxDRQ signal. The MPSC interrupts on the first character and thereafter only interrupts after a Special Receive Condition is detected. This mode can be reinitialized using the Enable Interrupt On Next Receive Character (WR0; D5, D4, D3) command which allows the next character received to generate an interrupt. Parity Errors do not cause interrupts, but End of Frame (SDLC operation) and Receive Overrun do cause interrupts in this mode. If the external status interrupts (WR1; D0) are enabled an interrupt may be generated any time the $\overline{CD}$ changes state.

Interrupt On Every Character mode generates an interrupt whenever a character enters the receive buffer. Errors and Special Receive Conditions generate a special vector if the Status Affects Vector (WR1B; D2) is selected. Also the Parity Error may be programmed (WR1; D4, D3) not to generate the special vector while in the Interrupt On Every Character mode.

The Special Receive Condition interrupt can only occur while in the Receive Interrupt On First Character Only or the Interrupt On Every Receive Character modes. The Special Receive Condition interrupt is caused by the Receive Overrun (RR1; D5) error condition. The error status reflects an error in the current word in the receive buffer, in addition to any Parity or Overrun errors since the last Error Reset (WR0; D5, D4, D3). The Receive Overrun and Parity error status bits are latched and can only be reset by the Error Reset (WR0; D5, D4, D3) command.

The CRC check result may be obtained by checking for CRC bit (RR1; D6). This bit gives the valid CRC result 16 bit times after the second CRC byte has been read from the MPSC. After reading the second CRC byte, the user software must read two more characters (may be sync characters) before checking for CRC result in RR1. Also for proper CRC computation by the receiver, the user software must reset the Receive CRC Checker (WR0; D7, D6) after receiving the first valid data character. The receive CRC Enable bit (WR3; D3) may also be enabled at this time.

SYNCHRONOUS OPERATION—SDLC

General

Like the other synchronous operations the SDLC mode must be initialized with the following parameters: SDLC mode (WR4; D5, D4), SDLC polynomial (WR5; D2), Request to Send, Data Terminal Ready, transmit character length (WR5; D6, D5), interrupt modes (WR1; WR2), Transmit Enable (WR5; D3), Receive Enable (WR3; D0), Auto Enable (WR3; D5) and External/Status Interrupt (WR1; D0). WR4 parameters must be written before WR1, WR3, WR5, WR6 and WR7.

The Interrupt modes for SDLC operation are similar to those discussed previously in the synchronous operations section.

Synchronous Mode Register Setup—SDLC/HDLC

	D7	D6	D5	D4	D3	D2	D1	D0
WR3	00 Rx 5 b/char 01 Rx 7 b/char 10 Rx 6 b/char 11 Rx 8 b/char		AUTO ENABLES	ENTER HUNT MODE	Rx CRC ENABLE	ADDRESS SEARCH MODE	0	Rx
WR4	0	0	1	0 (SELECTS SDLC/ HDLC MODE)	0	0	0	0
WR5	DTR	00 Tx ≤ 5 b/char 01 Tx 7 b/char 10 Tx 6 b/char 11 Tx 8 b/char		SEND BREAK	Tx ENABLE	0 (SELECTS SDLC/HDLC CRC)	RTS	Tx CRC ENABLE

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

Transmit

After a channel reset, the MPSC begins sending SDLC flags.

Following the flags in an SDLC operation the 8-bit address field, control field and information field may be sent to the MPSC by the microprocessor. The MPSC transmits the Frame Check Sequence using the Transmit Underrun feature. The MPSC automatically inserts a zero after every sequence of 5 consecutive 1's except when transmitting Flags or Aborts.

SDLC—like protocols do not have provision for fill characters within a message. The MPSC therefore automatically terminates an SDLC frame when the transmit data buffer and output shift register have no more bits to send. It does this by sending the two bytes of CRC and then one or more flags. This allows very high-speed transmissions under DMA or CPU control without requiring the CPU to respond quickly to the end-of-message situation.

After a reset, the Transmit Underrun/EOM status bit is in the set state and prevents the insertion of CRC characters during the time there is no data to send. Flag characters are sent. The MPSC begins to send the frame when data is written into the transmit buffer. Between the time the first data byte is written, and the end of the message, the Reset Transmit Underrun/EOM (WR0; D7, D6) command must be issued. The Transmit Underrun/EOM status bit (RR0; D6) is in the reset state at the end of the message which automatically sends the CRC characters.

The MPSC may be programmed to issue a Send Abort command (WR0; D5, D4, D3). This command causes at least eight 1's but less than fourteen 1's to be sent before the line reverts to continuous flags.

Receive

After initialization, the MPSC enters the Hunt phase, and remains in the Hunt phase until the first Flag is received. The MPSC never again enters the Hunt phase unless the microprocessor writes the Enter Hunt command. The MPSC will also detect flags separated by a single zero. For example, the bit pattern 011111101111110 will be detected as two flags.

The MPSC can be programmed to receive all frames or it can be programmed to the Address Search Mode. In the Address Search Mode, only frames with addresses that match the value in WR6 or the global address (0FFH) are received by the MPSC. Extended address recognition must be done by the microprocessor software.

The control and information fields are received as data.

SDLC/HDLC CRC calculation does not have an 8-bit delay, since all characters are included in the calculation, unlike Byte Synchronous Protocols.

Reception of an abort sequence (7 or more 1's) will cause the Break/Abort bit (RR0; D7) to be set and will cause an External/Status interrupt, if enabled. After the Reset External/Status Interrupts Command has been issued, a second interrupt will occur at the end of the abort sequence.

MPSC

Detailed Command/Status Description

GENERAL

The MPSC supports an extremely flexible set of serial and system interface modes.

The system interface to the CPU consists of 8 ports or buffers:

$\overline{CS}$	A_1	A_0	Read Operation	Write Operation
0	0	0	Ch. A Data Read	Ch. A Data Write
0	1	0	Ch. A Status Read	Ch. A Command/Parameter
0	0	1	Ch. B Data Read	Ch. B Data Write
0	1	1	Ch. B Status Read	Ch. B Command/Parameter
1	X	X	High Impedance	High Impedance

Data buffers are addressed by $A_1 = 0$, and Command ports are addressed by $A_1 = 1$.

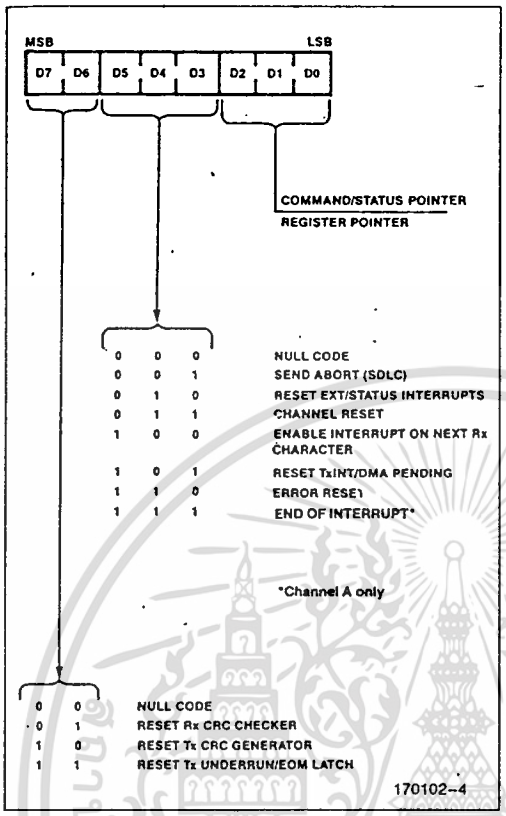
COMMAND/STATUS DESCRIPTION

The following command and status bytes are used during initialization and execution phases of operation. All Command/Status operations on the two channels are identical, and independent, except where noted.

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

Detailed Register Description

Write Register 0 (WR0):



WR0

D2, D1, D0—Command/Status Register Pointer bits determine which write-register the next byte is to be written into, or which read-register the next byte is to be read from. After reset, the first byte written into either channel goes into WR0. Following a read or write to any register (except WR0) the pointer will point to WR0.

D5, D4, D3—Command bits determine which of the basic seven commands are to be performed.

Command 0 Null—has no effect.

Command 1 Send Abort—causes the generation of eight to thirteen 1's when in the SDLC mode.

Command 2 Reset External/Status Interrupts—resets the latched status bits of RR0 and re-enables them, allowing interrupts to occur again.

Command 3 Channel Reset—resets the Latched Status bits of RR0, the interrupt prioritization logic and all control registers for the channel. Four extra system clock cycles should be allowed for MPSC reset time before any additional commands or controls are written into the channel.

Command 4 Enable Interrupt on Next Receive Character—if the Interrupt on First Receive Character mode is selected, this command reactivates that mode after each complete message is received to prepare the MPSC for the next message.

Command 5 Reset Transmitter Interrupt/DMA Pending—if The Transmit Interrupt/DMA Enable mode is selected, the MPSC automatically interrupts or requests DMA data transfer when the transmit buffer becomes empty. When there are no more characters to be sent, issuing this command prevents further transmitter interrupts or DMA requests until the next character has been completely sent.

Command 6 Error Reset—error latches, Parity and Overrun errors in RR1 are reset.

Command 7 End of Interrupt—resets the interrupt-in-service latch of the highest-priority internal device under service.

D7, D6 CRC Reset Code.

00 Null—has no effect.

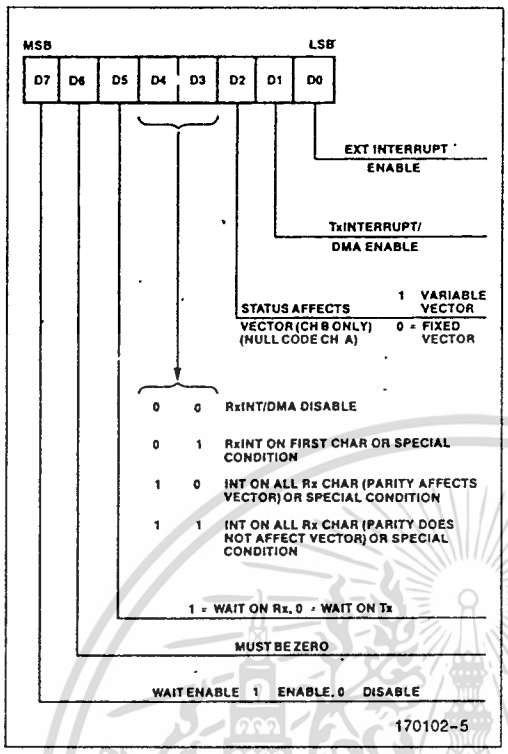
01 Reset Receive CRC Checker—resets the CRC checker to 0's. If in SDLC mode the CRC checker is initialized to all 1's.

10 Reset Transmit CRC Generator—resets the CRC generator to 0's. If in SDLC mode the CRC generator's initialized to all 1's.

11 Reset Tx Underrun/End of Message Latch.

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

Write Register 1 (WR1):



WR1

D0 — External/Status Interrupt Enable—allows interrupt to occur as the result of transitions on the $\overline{CD}$, $\overline{CTS}$ or $\overline{SYNDET}$ inputs. Also allows interrupts as the result of a Break/Abort detection and termination, or at the beginning of CRC, or sync character transmission when the Transmit Underrun/EOM latch becomes set.

D1 — Transmitter Interrupt/DMA Enable—allows the MPSC to interrupt or request a DMA transfer when the transmitter buffer becomes empty.

D2 — Status Affects vector—(WR1, D2 active in channel B only.) If this bit is not set, then the fixed vector, programmed in WR2, is returned from an interrupt acknowledge sequence. If the bit if set then the vector returned from an interrupt acknowledge is variable as shown in the Interrupt Vector Table.

D4, D3 — Receive Interrupt Mode.

- 0 0 — Receive Interrupts/DMA Disabled.
- 0 1 — Receive Interrupt on First Character Only or Special Condition.
- 1 0 — Interrupt on All Receive Characters or Special Condition (Parity Error is a Special Receive Condition).
- 1 1 — Interrupt on All Receive Characters or Special Condition (Parity Error is not a Special Receive Condition).

D5 — Wait on Receive/$\overline{Transmit}$—when the following conditions are met the RDY pin is activated, otherwise it is held in the High-Z state. (Conditions: Interrupt Enabled Mode, Wait Enabled, $\overline{CS}$ = 0, A0 = 0/1, and A1 = 0). The RDY pin is pulled low when the transmitter buffer is full or the receiver buffer is empty and it is driven High when the transmitter buffer is empty or the receiver buffer is full. The RDY_A and RDY_B may be wired OR connected since only one signal is active at any one time while the other is in the High Z state.

D6 — Must be Zero.

D7 — Wait Enable—enables the wait function.

WR2 — Channel A Only

D1, D0 — System Configuration—These specify the data transfer from MPSC channels to the CPU, either interrupt or DMA based.

- 0 0 — Channel A and Channel B both use interrupts.
- 0 1 — Channel A uses DMA, Channel B uses interrupts.
- 1 0 — Channel A and Channel B both use DMA.
- 1 1 — Illegal Code.

D2 — Priority—this bit specifies the relative priorities of the internal MPSC interrupt/DMA sources.

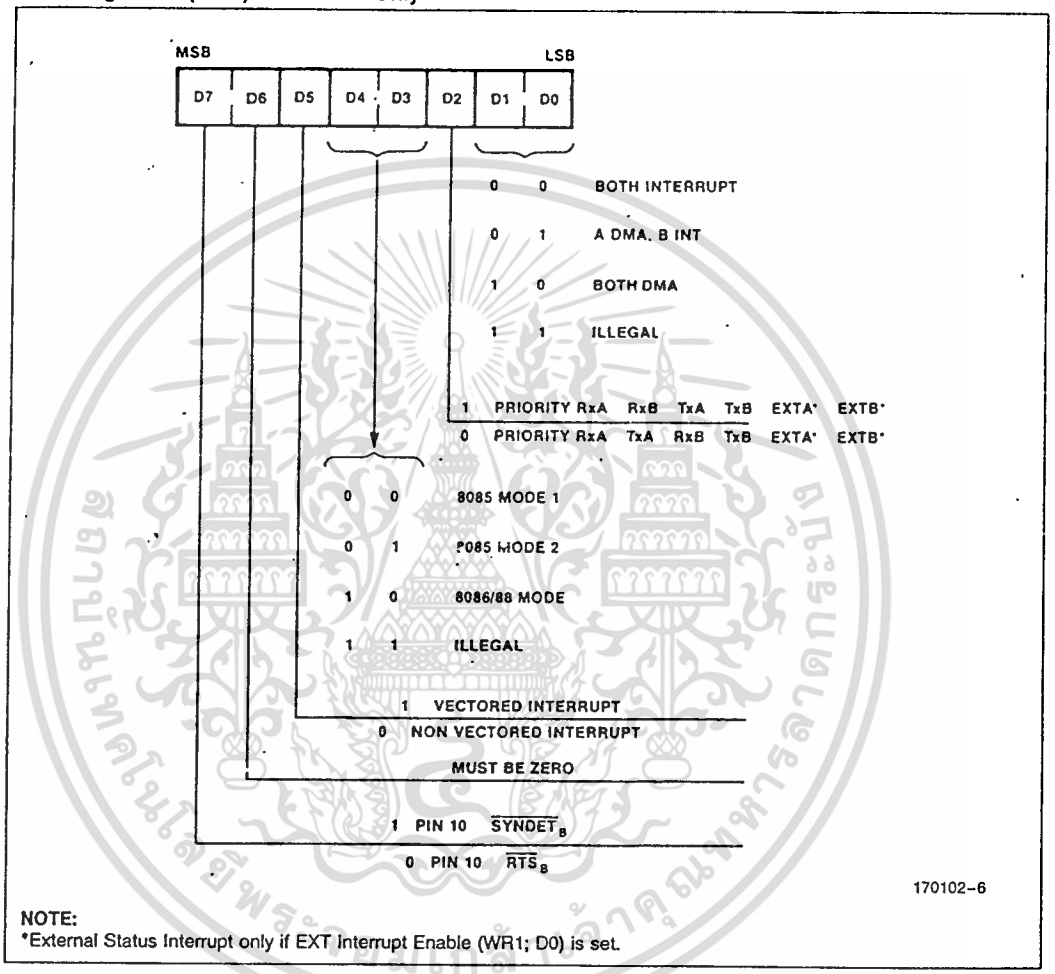
- 0 — (Highest) RxA, TxA, RxB, TxB, ExTA, ExTB (Lowest).
- 1 — (Highest) RxA, RxB, TxA, TxB, ExTA, ExTB (Lowest).

D5, D4, D3 — Interrupt Code—specifies the behavior of the MPSC when it receives an interrupt acknowledge sequence from the CPU. (See Interrupt Vector Mode Table.)

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

0 X X	Non-vectored interrupts—intended for use with external DMA CONTROLLER. The Data Bus remains in a high impedance state during INTA sequences.
1 0 0	8085 Vector Mode 1—intended for use as the primary MPSC in a daisy chained priority structure. (See System Interface section).
1 0 1	8085 Vector Mode 2—intended for use as any secondary MPSC in a daisy chained priority structure. (See System Interface section).
1 1 0	8086/88 Vector Mode—intended for use as either a primary or secondary in a daisy chained priority structure. (See System Interface section).
D6	Must be zero.
D7	zero Pin 10 = $\overline{RTS}_B$ one Pin 10 = $\overline{SYNDET}_B$

Write Register 2 (WR2): Channel A Only



NOTE:
*External Status Interrupt only if EXT Interrupt Enable (WR1; D0) is set.

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

The following table describes the MPSC's response to an interrupt acknowledge sequence:

D5	D4	D3	$\overline{IPI}$	MODE	INTA	Data Bus
0	X	X	X	Non-vectored	Any INTA	D7 High Impedance D0
1	0	0	0	85 Mode 1	1st INTA	1 1 0 0 1 1 0 1
					2nd INTA	V7 V6 V5 V4* V3* V2* V1 V0
					3rd INTA	0 0 0 0 0 0 0 0
1	0	0	1	85 Mode 1	1st INTA	1 1 0 0 1 1 0 1
					2nd INTA	High Impedance
					3rd INTA	High Impedance
1	1	0	0	86 Mode	1st INTA	High Impedance
					2nd INTA	V7 V6 V5 V4 V3 V2* V1* V0*
1	0	1	0	85 Mode 2	1st INTA	High Impedance
					2nd INTA	V7 V6 V5 V4* V3* V2* V1 V0
					3rd INTA	0 0 0 0 0 0 0 0
1	0	1	1	85 Mode 2	1st INTA	High Impedance
					2nd INTA	High Impedance
					3rd INTA	High Impedance
1	1	0	1	86 Mode	1st INTA	High Impedance
					2nd INTA	High Impedance

NOTE:
*These bits are variable if the "status affects vector" mode has been programmed, (WR1B, D2).

Interrupt/DMA Mode, Pin Functions, and Priority

Ch. A WR2			Int/DMA Mode		Pin Functions				Priority
					RDY_A/ $RxDRQ_A$ Pin 32	RDY_B/ $TxDRQ_A$ Pin 11	$\overline{PIP}$/ $RxDRQ_B$ Pin 29	$\overline{IPO}$/ $TxDRQ_B$ Pin 30	Highest Lowest
D_2	D_1	D_0	CH. A	CH. B					
0	0	0	INT	INT	RDY_A	RDY_B	$\overline{IPI}$	$\overline{IPO}$	RxA, TxA, RxB, TxB, EXT_A, EXT_B
1	0	0	INT	INT					RxA, RxB, TxA, TxB, EXT_A, EXT_B
0	0	1	DMA		$RxDRQ_A$	$TxDRQ_A$	$\overline{IPI}$	$\overline{IPO}$	RxA, TxA (DMA)
				INT					RxA(1), RxB, TxB, EXT_A, EXT_B (INT)
1	0	1	DMA						RxA, TxA (DMA)
				INT					RxA(1), RxB, TxB, EXT_A, EXT_B (INT)
0	1	0	DMA	DMA	$RxDRQ_A$	$TxDRQ_A$	$RxDRQ_B$	$TxDRQ_B$	RxA, TxA, RxB, TxB (DMA) RxA(1), RxB(1), EXT_A, EXT_B, (INT)
1	1	0	DMA	DMA					RxA, RxB, TxA, TxB, (DMA) RxA(1), RxB(1), EXT_A, EXT_B (INT)

NOTE:
1. Special Receive Condition

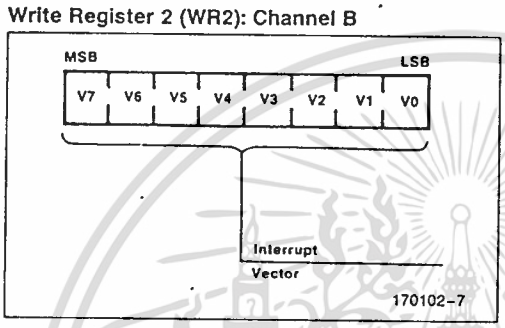
เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

Interrupt Vector Mode Table

8085 Modes 8086/88 Mode	V_4 V_2	V_3 V_1	V_2 V_0	Channel	Condition
	0 0 0 0	0 0 1 1	0 1 0 1	B	Tx Buffer Empty Ext/Status Change Rx Char. Available Special Rx Condition (Note 1)
	1 1 1 1	0 0 1 1	0 1 0 1	A	Tx Buffer Empty Ext/Status Change Rx Char. Available Special Rx Condition (Note 1)

NOTE:
1. Special Receive Condition = Parity Error, Rx Overrun Error, Framing Error, End of Frame (SDLC).

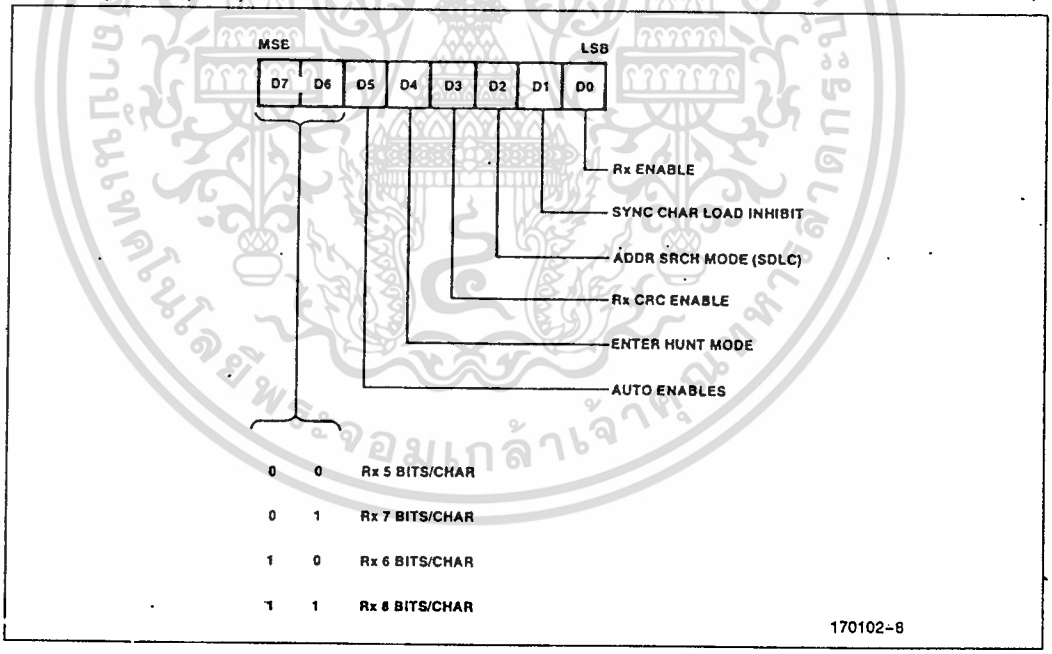
Write Register 2 (WR2): Channel B



WR2 CHANNEL B

D7–D0 Interrupt vector—This register contains the value of the interrupt vector placed on the data bus during interrupt acknowledge sequences.

Write Register 3 (WR3):



เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

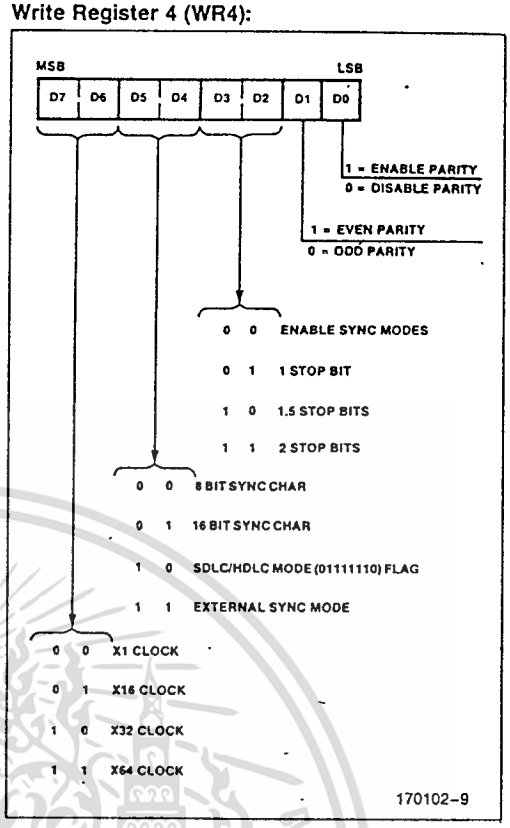
WR3

D0	Receiver Enable—A one enables the receiver to begin. This bit should be set only after the receiver has been initialized.
D1	Sync Character Load Inhibit—A one prevents the receiver from loading sync characters into the receive buffers. In SDLC, this bit must be zero.
D2	Address Search Mode—If the SDLC mode has been selected, the MPSC will receive all frames unless this bit is a 1. If this bit is a 1, the MPSC will receive only frames with address (0FFH) or the value loaded into WR6. This bit must be zero in non-SDLC modes.
D3	Receive CRC Enable—A one in this bit enables (or re-enables) CRC calculation. CRC calculation starts with the last character placed in the Receiver FIFO. A zero in this bit disables, but does not reset, the Receiver CRC generator.
D4	Enter Hunt Phase—After initialization, the MPSC automatically enters the Hunt mode. If synchronization is lost, the Hunt phase can be re-entered by writing a one to this bit.
D5	Auto Enable—A one written to this bit causes $\overline{CD}$ to be automatic enable signal for the receiver and $\overline{CTS}$ to be an automatic enable signal for the transmitter. A zero written to this bit limits the effect of $\overline{CD}$ and $\overline{CTS}$ signals to setting/resetting their corresponding bits in the status register (RR0).
D7, D6	Receive Character length
0 0	Receive 5 Data bits/character
0 1	Receive 7 Data bits/character
1 0	Receive 6 Data bits/character
1 1	Receive 8 Data bits/character

WR4

D0	Parity—A one in this bit causes a parity bit to be added to the programmed number of data bits per character for both the transmitted and received character. If the MPSC is programmed to receive 8 bits per character, the parity bit is not transferred to the microprocessor. With other receiver character lengths, the parity bit is transferred to the microprocessor.

Write Register 4 (WR4):

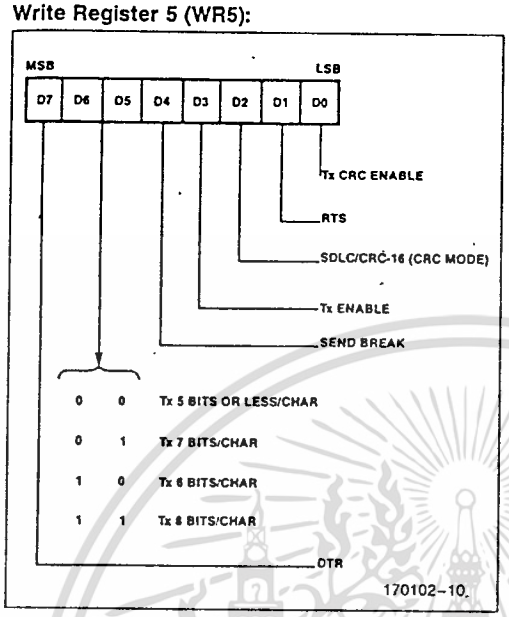


D1	Even/Odd Parity—If parity is enabled, a one in this bit causes the MPSC to transmit and expect even parity, and a zero causes it to send and expect odd parity.
D3, D2	Stop bits/sync mode
0 0	Selects synchronous modes
0 1	Async mode, 1 stop bit/character
1 0	Async mode, 1½ stop bits/character
1 1	Async mode, 2 stop bits/character
D5, D4	Sync mode select
0 0	8-bit sync character
0 1	16-bit sync character
1 0	SDLC mode (Flag sync)
1 1	External sync mode
D7, D6	Clock Mode—Selects the clock/data rate multiplier for both the receiver and the transmitter. 1x mode must be selected for synchronous modes. If the 1x mode is selected, bit synchronization must be done externally.

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

0	0	Clock rate = Data rate × 1
0	1	Clock rate = Data rate × 16
1	0	Clock rate = Data rate × 32
1	1	Clock rate = Data rate × 64

Write Register 5 (WR5):



WR5

D0 Transmit CRC Enable—A one in this bit enables the transmitter CRC generator. The CRC calculation is done when a character is moved from the transmit buffer into the shift register. A zero in this bit disables CRC calculations. If this bit is not set when a transmitter underrun occurs, the CRC will not be sent.

D1 Request to Send—A one in this bit forces the $\overline{RTS}$ pin active (low) and zero in this bit forces the $\overline{RTS}$ pin inactive (high).

D2 CRC Select—A one in this bit selects the CRC−16 polynomial ($X^{16} + X^{15} + X^2 + 1$) and a zero in this bit selects the CCITT-CRC polynomial ($X^{16} + X^{12} + X^5 + 1$). In SDLC mode, CCITT-CRC must be selected.

D3 Transmitter Enable—A zero in this bit forces a marking state on the transmitter output. If this bit is set to zero during data or sync character transmission, the marking state is entered after the character has been sent. If this bit is set to zero during transmission of a CRC character, sync or flag bits are substituted for the remainder of the CRC bits.

D4 Send Break—A one in this bit forces the transmit data low. A zero in this bit allows normal transmitter operation.

D6, D5 Transmit Character length

0	0	Transmit 1-5 bits/character
0	1	Transmit 7 bits/character
1	0	Transmit 6 bits/character
1	1	Transmit 8 bits/character

Bits to be sent must be right justified least significant bit first, e.g.:

D7	D6	D5	D4	D3	D2	D1	D0
0	0	B5	B4	B3	B2	B1	B0

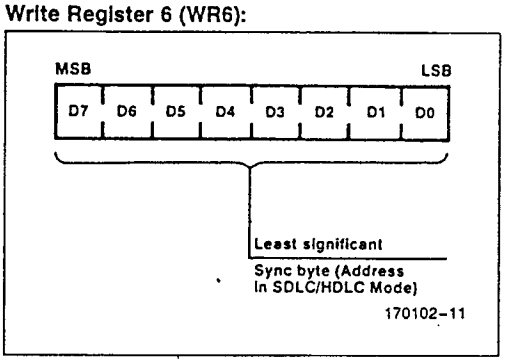
D7 Data Terminal Ready—When set, this bit forces the $\overline{DTR}$ pin active (low). When reset, this bit forces the $\overline{DTR}$ pin inactive (high).

Five or less mode allows transmission of one to five bits per character. The microprocessor must format the data in the following way:

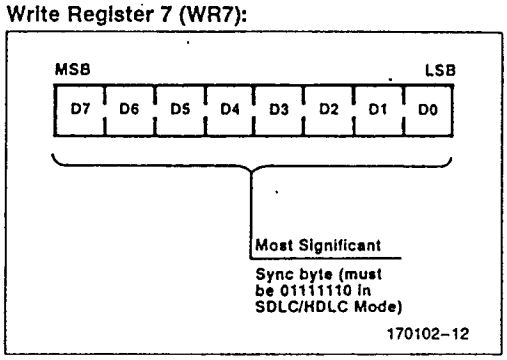
D7	D6	D5	D4	D3	D2	D1	D0	
1	1	1	1	0	0	0	B0	Sends one data bit
1	1	1	0	0	0	B1	B0	Sends two data bits
1	1	0	0	0	B2	B1	B0	Sends three data bits
1	0	0	0	B3	B2	B1	B0	Sends four data bits
0	0	0	B4	B3	B2	B1	B0	Sends five data bits

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

Write Register 6 (WR6):



Write Register 7 (WR7):



WR6

D7–D0 Sync/Address—This register contains the transmit sync character in Monosync mode, the low order 8 sync bits in Bisync mode, or the Address byte in SDLC mode.

WR7

D7–D0 Sync/Flag—This register contains the receive sync character in Monosync mode, the high order 8 sync bits in Bisync mode, or the Flag character (01111110) in SDLC mode. WR7 is not used in External Sync mode.

RR0

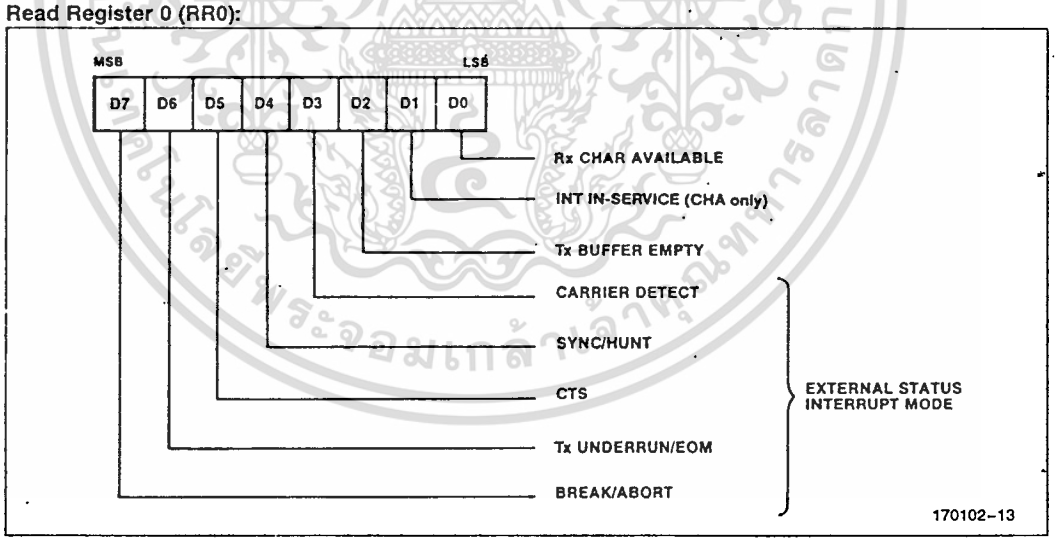
D0 Receive Character Available—This bit is set when the receive FIFO contains data and is reset when the FIFO is empty.

D1 Interrupt In-Service*—If an Internal Interrupt is pending, this bit is set at the falling edge of the second $\overline{INTA}$ pulse of an INTA cycle. In non-vectored mode, this bit is set at the falling edge of $\overline{RD}$ after pointer 2 is specified. This bit is reset when an EOI command is issued and there are no other interrupts in-service at that time.

D2 Transmit Buffer Empty—This bit is set whenever the transmit buffer is

*This bit is only valid when $\overline{IPI}$ is active low and is always zero in Channel B.

Read Register 0 (RR0):



เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

empty except when CRC characters are being sent in a synchronous mode. This bit is reset when the transmit buffer is loaded. This bit is set after an MPSC reset.

D3 Carrier Detect—This bit contains the state of the $\overline{CD}$ pin at the time of the last change of any of the External/Status bits (CD, CTS, Sync/Hunt, Break/Abort, or Tx Underrun/EOM). Any change of state of the $\overline{CD}$ pin causes the CD bit to be latched and causes an External/Status interrupt. This bit indicates current state of the $\overline{CD}$ pin immediately following a Reset External/Status Interrupt command.

D4 Sync/Hunt—In asynchronous modes, the operation of this bit is similar to the CD status bit, except that Sync/Hunt shows the state of the $\overline{SYNDET}$ input. Any High-to-Low transition on the $\overline{SYNDET}$ pin sets this bit, and causes an External/Status interrupt (if enabled). The Reset External/Status Interrupt command is issued to clear the interrupt. A Low-to-High transition clears this bit and sets the External/Status interrupt. When the External/Status interrupt is set by the change in state of any other input or condition, this bit shows the inverted state of the $\overline{SYNDET}$ pin at time of the change. This bit must be read immediately following a Reset External/Status Interrupt command to read the current state of the $\overline{SYNDET}$ input.

In the External Sync mode, the Sync/Hunt bit operates in a fashion similar to the Asynchronous mode, except the Enter Hunt Mode control bit enables the external sync detection logic. When the External Sync Mode and Enter Hunt Mode bits are set (for example, when the receiver is enabled following a reset), the $\overline{SYNDET}$ input must be held High by the external logic until external character synchronization is achieved. A High at the $\overline{SYNDET}$ input holds the Sync/Hunt status in the reset condition.

When external synchronization is achieved, $\overline{SYNDET}$ must be driven Low on the second rising edge of $\overline{RxC}$ after the rising edge of $\overline{RxC}$ on which the last bit of the sync character was received. In other words, after the sync pattern is detected, the external logic must wait for two full Receive Clock cycles to activitate the $\overline{SYNDET}$ input. Once $\overline{SYNDET}$ is forced Low, it is good practice to keep it Low until the CPU informs the external sync logic that synchronization has been lost or a new message is about to start. The High-to-Low transition of the $\overline{SYNDET}$ output sets the Sync/Hunt bit, which sets the External/Status interrupt. The CPU must clear the interrupt by issuing the Reset External/Status Interrupt Command.

When the $\overline{SYNDET}$ input goes High again, another External/Status interrupt is generated that must also be cleared. The Enter Hunt Mode control bit is set whenever character synchronization is lost or the end of message is detected. In this case, the MPSC again looks for a High-to-Low transition on the $\overline{SYNDET}$ input and the operation repeats as explained previously. This implies the CPU should also inform the external logic that character synchronization has been lost and that the MPSC is waiting for $\overline{SYNDET}$ to become active.

In the Monosync and Bisync Receive modes, the Sync/Hunt status bit is initially set to 1 by the Enter Hunt Mode bit. The Sync/Hunt bit is reset when the MPSC establishes character synchronization. The High-to-Low transition of the Sync/Hunt bit causes an External/Status interrupt that must be cleared by the CPU issuing the Reset External/Status Interrupt command. This enables the MPSC to detect the next transition of other External/Status bits.

When the CPU detects the end of message or that character synchronization is lost, it sets the Enter Hunt Mode control bit, which sets the Sync/Hunt bit to 1. The Low-to-High transition of the Sync/Hunt bit sets the External/Status Interrupt, which must also be cleared by the Reset External/Status !nterrupt Command. Note that the $\overline{SYNDET}$ pin acts as an output in this mode, and goes low every time a sync pattern is detected in the data stream.

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

In the SDLC mode, the Sync/Hunt bit is initially set by the Enter Hunt mode bit, or when the receiver is disabled. In any case, it is reset to 0 when the opening flag of the first frame is detected by the MPSC. The External/Status interrupt is also generated, and should be handled as discussed previously.

Unlike the Monosync and Bisync modes, once the Sync/Hunt bit is reset in the SDLC mode, it does not need to be set when the end of message is detected. The MPSC automatically maintains synchronization. The only way the Sync/Hunt bit can be set again is by the Enter Hunt Mode bit, or by disabling the receiver.

D5 — Clear to Send—This bit contains the inverted state of the $\overline{CTS}$ pin at the time of the last change of any of the External/Status bits (CD, CTS, Sync/Hunt, Break/Abort, or Tx Underrun/EOM). Any change of state of the $\overline{CTS}$ pin causes the CTS bit to be latched and causes an External/Status interrupt. This bit indicates the inverse of the current state of the $\overline{CTS}$ pin immediately following a Reset External/Status Interrupt command.

D6 — Transmitter Underrun/End of Message—This bit is in a set condition following a reset (internal or external). The only command that can reset this bit is the Reset Transmit Underrun/EOM Latch command (WR0, D_6 and D_7). When the Transmit Underrun condition occurs, this bit is set, which causes the External/Status Interrupt which must be reset by issuing a Reset External/Status command (WR0; command 2).

D7 — Break/Abort—In the Asynchronous Receive mode, this bit is set when a Break sequence (null character plus framing error) is detected in the data stream. The External/Status interrupt, if enabled, is set when break is detected. The interrupt service routine must issue the Reset External/Status Interrupt command (WR0, Command 2) to the break detection logic so the Break sequence termination can be recognized.

The Break/Abort bit is reset when the termination of the Break sequence is detected in the incoming data stream. The termination of the Break sequence also causes the External/Status interrupt to be set. The Reset External/Status Interrupt command must be issued to enable the break detection logic to look for the next Break sequence. A single extraneous null character is present in the receiver after the termination of a break; it should be read and discarded.

In the SDLC Receive mode, this status bit is set by the detection of an Abort sequence (seven or more 1's). The External/Status interrupt is handled the same way as in the case of a Break. The Break/Abort bit is not used in the Synchronous Receive mode.

D0 — All Sent—This bit is set when all characters have been sent, in asynchronous modes. It is reset when characters are in the transmitter, in asynchronous modes. In synchronous modes, this bit is always set.

RR1:

D3, D2, D1 — Residue Codes—Bit synchronous protocols allow I-fields that are not an integral number of characters. Since transfers from the MPSC to the CPU are character oriented, the residue codes provide the capability of receiving leftover bits. Residue bits are right justified in the last data byte received or first CRC byte.

D4 — Parity Error—If parity is enabled, this bit is set for received characters whose parity does not match the programmed sense (Even/Odd). This bit is latched. Once an error occurs, it remains set until the Error Reset command is written.

D5 — Receive Overrun Error—This bit indicates that the receive FIFO has been overloaded by the receiver. The last character in the FIFO is overwritten and flagged with this error. Once the overwritten character is read, this error condition is latched until reset by the Error Reset command. If the MPSC is in the status affects vector mode, the overrun causes a special Receive Condition Vector.

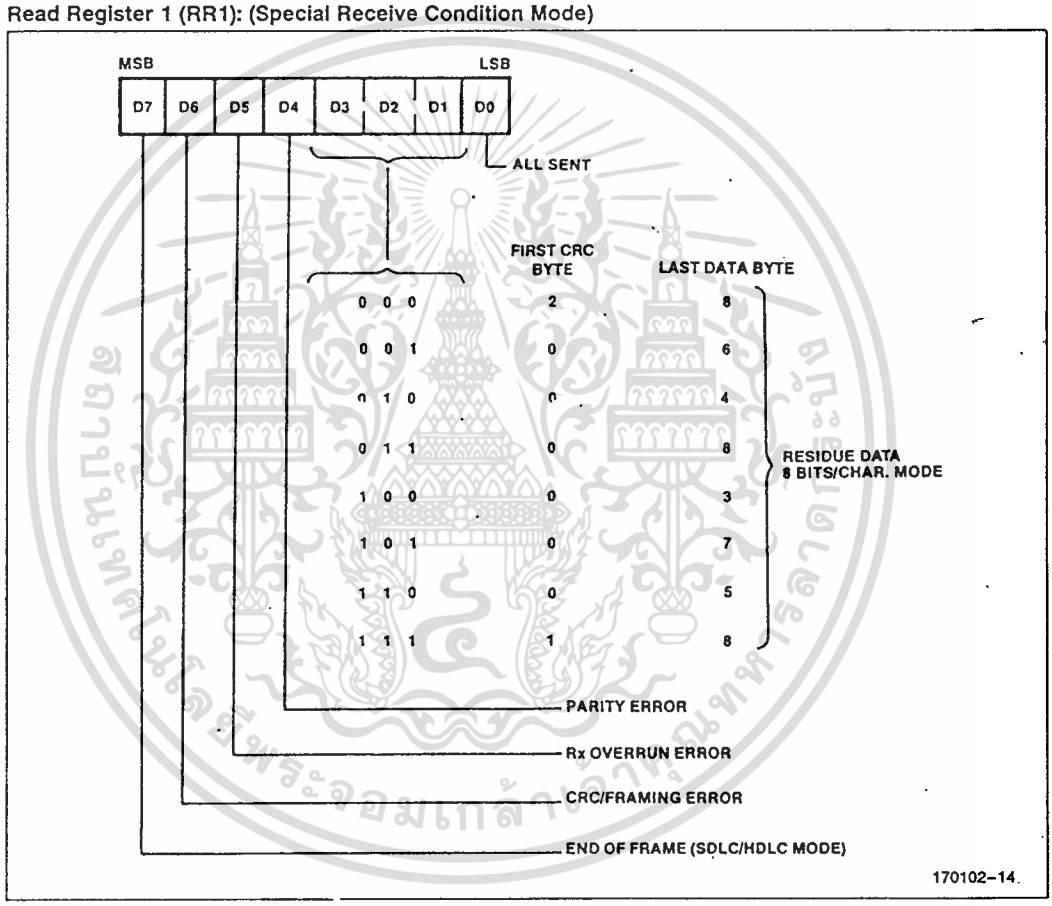
D6 — CRC/Framing Error—In async modes, a one in this bit indicates a receive framing error. In synchronous modes, a one in this bit indicates that the calculated CRC value does not match the last two bytes received. It can be reset by issuing an Error Reset command.

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

SDLC Residue Code Table (I Field Bits in 2 Previous Bytes)

RR1			8 bits/char		7 bits/char		6 bits/char		5 bits/char	
D3	D2	D1	First CRC Byte	Last Data Byte	First CRC Byte	Last Data Byte	First CRC Byte	Last Data Byte	First CRC Byte	Last Data Byte
1	0	0	0	3	0	2	0	1	0	5
0	1	0	0	4	0	3	0	2	0	1
1	1	0	0	5	0	4	0	3	0	2
0	0	1	0	6	0	5	0	4	0	3
1	0	1	0	7	0	6	0	5	—	—
0	1	1	0	8	0	—	—	—	—	—
1	1	1	1	8	—	—	—	—	—	—
0	0	0	2	8	1	7	0	6	0	4

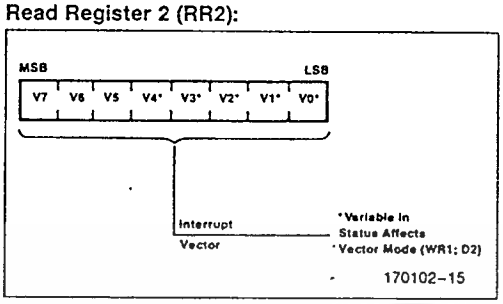
Read Register 1 (RR1): (Special Receive Condition Mode)



เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

D7 End of Frame—This bit is valid only in SDLC mode. A one indicates that a valid ending flag has been received. This bit is reset either by an Error Reset command or upon reception of the first character of the next frame.

Read Register 2 (RR2):



RR2 Channel B

D7–D0 Interrupt Vector—Contains the interrupt vector programmed into WR2. If the status affects vector mode is selected (WR1; D2), it contains the modified vector (See WR2). RR2 contains the modified vector for the highest priority interrupt pending. If no interrupts are pending, the variable bits in the vector are set to one.

SYSTEM INTERFACE

General

The MPSC to Microprocessor System interface can be configured in many flexible ways. The basic interface types are polled, wait, interrupt driven, or direct memory access driven.

Polled operation is accomplished by repetitively reading the status of the MPSC, and making decisions based on that status. The MPSC can be polled at any time.

Wait operation allows slightly faster data throughput for the MPSC by manipulating the Ready input to the microprocessor. Block Read or Write Operations to the MPSC are started at will by the microprocessor and the MPSC deactivates its RDY signal if it is not yet ready to transmit the new byte, or if reception of new byte is not completed.

Interrupt driven operation is accomplished via an internal or external interrupt controller. When the MPSC requires service, it sends an interrupt request signal to the microprocessor, which responds with an interrupt acknowledge signal. When the internal or external interrupt controller receives the acknowledge, it vectors the microprocessor to a service routine, in which the transaction occurs.

DMA operation is accomplished via an external DMA controller. When the MPSC needs a data transfer, it requests a DMA cycle from the DMA controller. The DMA controller then takes control of the bus and simultaneously does a read from the MPSC and a write to memory or vice-versa.

The following section describes the many configurations of these basic types of system interface techniques for both serial channels.

POLLED OPERATION

In the polled mode, the CPU must monitor the desired conditions within the MPSC by reading the appropriate bits in the read registers. All data available, status, and error conditions are represented by the appropriate bits in read registers 0 and 1 for channels A and B.

There are two ways in which the software task of monitoring the status of the MPSC has been reduced. One is the "ORing" of all conditions into the Interrupt Pending bit. (RR0; D1 channel A only). This bit is set when the MPSC requires service, allowing the CPU to monitor one bit instead of four status registers. The other is available when the "status-affects-vector" mode is selected. By reading RR2 Channel B, the CPU can read a vector who's value will indicate that one or more of group of conditions has occurred, narrowing the field of possible conditions. See WR2 and RR2 in the Detailed Command Description section.

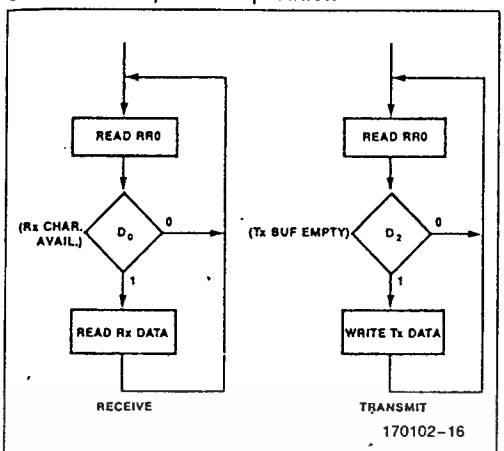
WAIT OPERATION

Wait Operation is intended to facilitate data transmission or reception using block move operations. If a block of data is to be transmitted, for example, the CPU can execute a String I/O instruction to the MPSC. After writing the first byte, the CPU will attempt to write a second byte immediately as is the case of block move. The MPSC forces the RDY signal low which inserts wait states in the CPU's write cycle until the transmit buffer is ready to accept a new byte. At that time, the RDY signal is high allowing the CPU to finish the write cycle. The CPU then attempts the third write and the process is repeated.

Similar operation can programmed for the receiver. During initialization, wait on transmit (WR1; D5 = 0)

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

Software Flow, Polled Operation

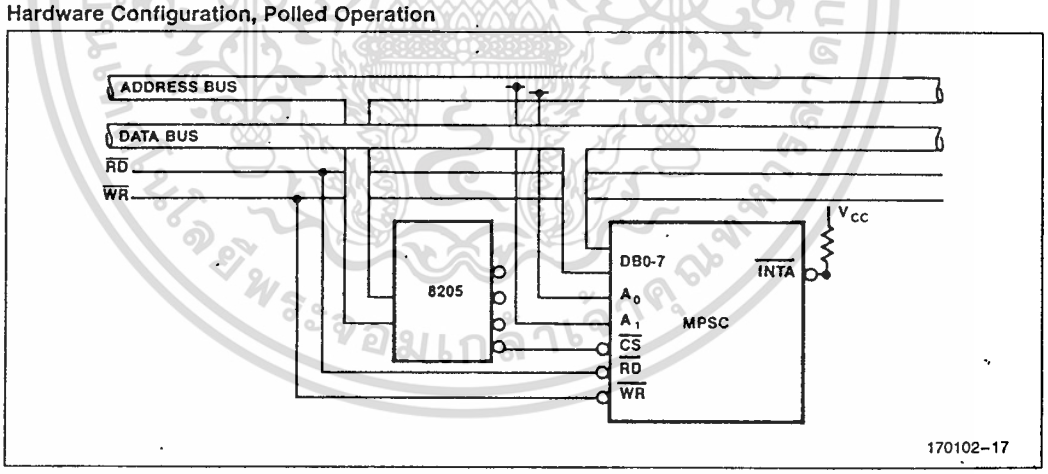


NOTES:
1. RR0; D0 is reset automatically when the data is read.
2. RR0; D2 is reset automatically when the data is written.

or wait on receive (WR1; D5 = 1) can be selected. The wait operation can be enabled/disabled by setting/resetting the Wait Enable Bit (WR1; D7).

NOTE:
CAUTION: ANY CONDITION THAT CAN CAUSE THE TRANSMITTER TO STOP (E.G., CTS GOES INACTIVE) OR THE RECEIVER TO STOP (E.G., RX DATA STOPS) WILL CAUSE THE MPSC TO HANG THE CPU UP IN WAIT STATES UNTIL RESET. EXTREME CARE SHOULD BE TAKEN WHEN USING THIS FEATURE.

Hardware Configuration, Polled Operation



INTERRUPT DRIVEN OPERATION

The MPSC can be programmed into several interrupt modes: Non-Vectored, 8085 vectored, and 8088/86 vectored. In both vectored modes, multiple MPSC's can be daisy-chained.

In the vectored mode, the MPSC responds to an interrupt acknowledge sequence by placing a call instruction (8085 mode) and interrupt vector (8085 and 8088/86 mode) on the data bus.

The MPSC can be programmed to cause an interrupt due to up to 14 conditions in each channel. The status of these interrupt conditions is contained in Read Registers 0 and 1. These 14 conditions are all directed to cause 3 different types of internal interrupt request for each channel: receive/interrupts, transmit interrupts and external/status interrupts (if enabled).

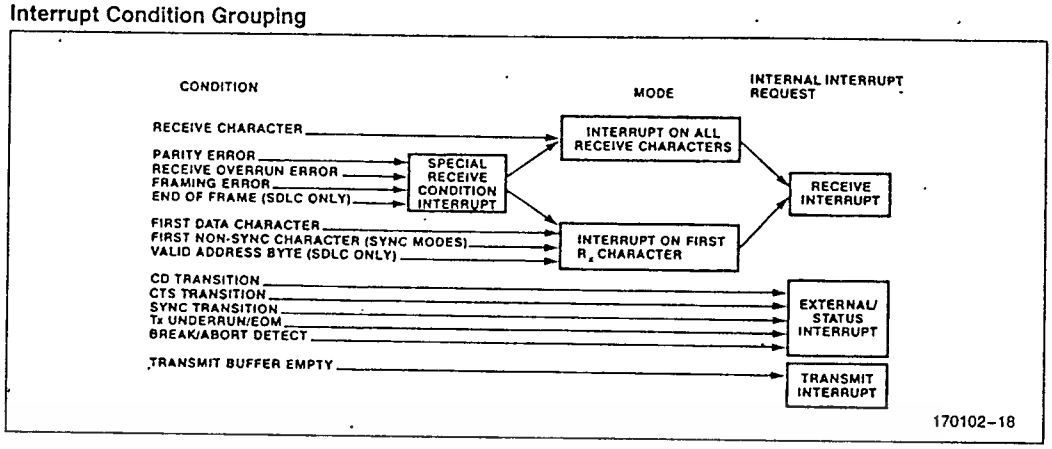
This results in up to 6 internal interrupt request signals. The priority of those signals can be programmed to one of two fixed modes:

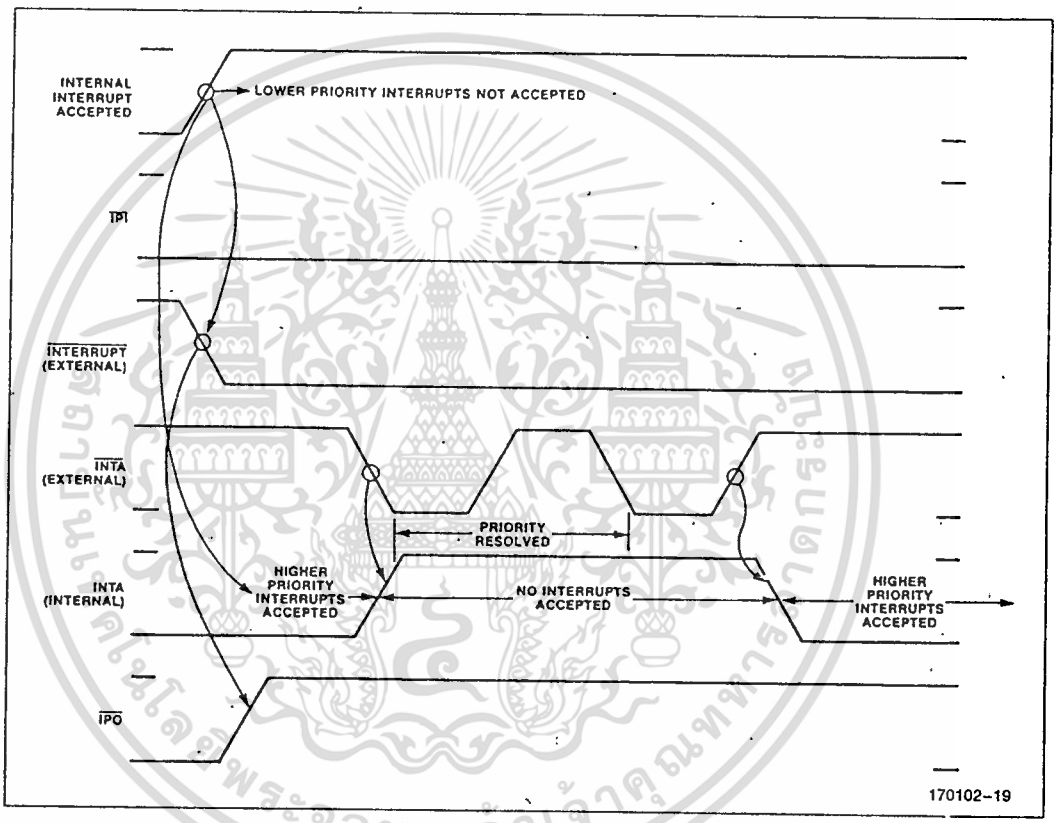
Highest Priority					Lowest Priority
RxA	RxB	TxA	TxB	ExTA	ExTB
RxA	TxA	RxB	TxB	ExTA	ExTB

The interrupt priority resolution works differently for vectored and non-vectored modes.

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

Interrupt Condition Grouping



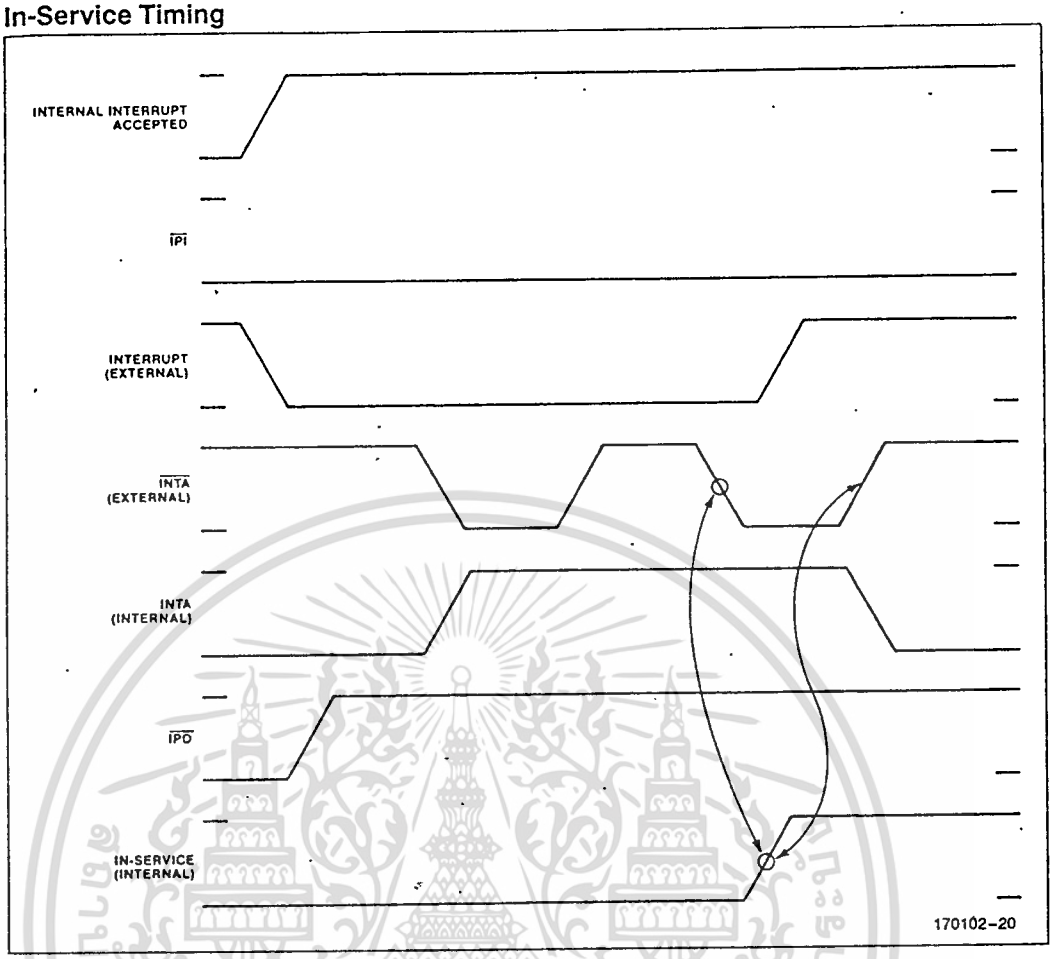


PRIORITY RESOLUTION: VECTORED MODE

Any interrupt condition can be accepted internally to the MPSC at any time, unless the MPSC's internal INTA signal is active, unless a higher priority interrupt is currently accepted, or if $\overline{IPI}$ is inactive (high). The MPSC's internal INTA is set on the leading (falling) edge of the first External $\overline{INTA}$ pulse and reset on the trailing (rising) edge of the second External $\overline{INTA}$ pulse. After an interrupt is accepted internally, and External $\overline{INT}$ request is generated and the $\overline{IPO}$ goes inactive. $\overline{IPO}$ and $\overline{IPI}$ are used for daisy-chaining MPSC's together.

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

In-Service Timing



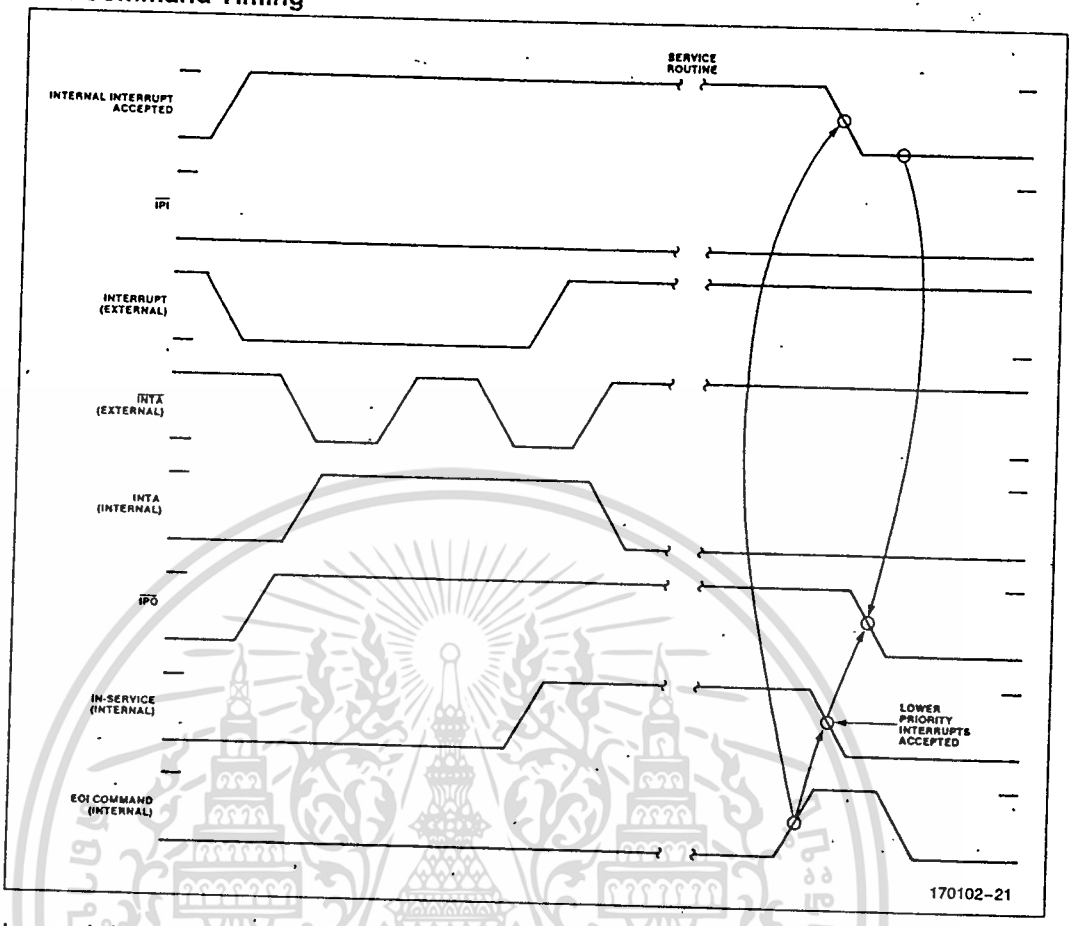
The MPSC's internal INTA is set on the leading (falling) edge of the first external $\overline{\text{INTA}}$ pulse, and reset on the trailing (rising) edge of the second external $\overline{\text{INTA}}$ pulse. After an interrupt is accepted internally, and external $\overline{\text{INT}}$ request is generated and $\overline{\text{IPO}}$ goes inactive (high). $\overline{\text{IPO}}$ and $\overline{\text{IPI}}$ are used for daisy-chaining MPSC's together.

Each of the six interrupt sources has an associated In-Service latch. After priority has been resolved, the highest priority In-Service latch is set. After the In-Service latch is set, the $\overline{\text{INT}}$ pin goes inactive (high).

NOTE:
If the External $\overline{\text{INT}}$ pin is active and the IPI signal is pulled inactive high, the $\overline{\text{INT}}$ signal will also go inactive. $\overline{\text{IPI}}$ qualifies the External $\overline{\text{INT}}$ Signal.

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

EOI Command Timing

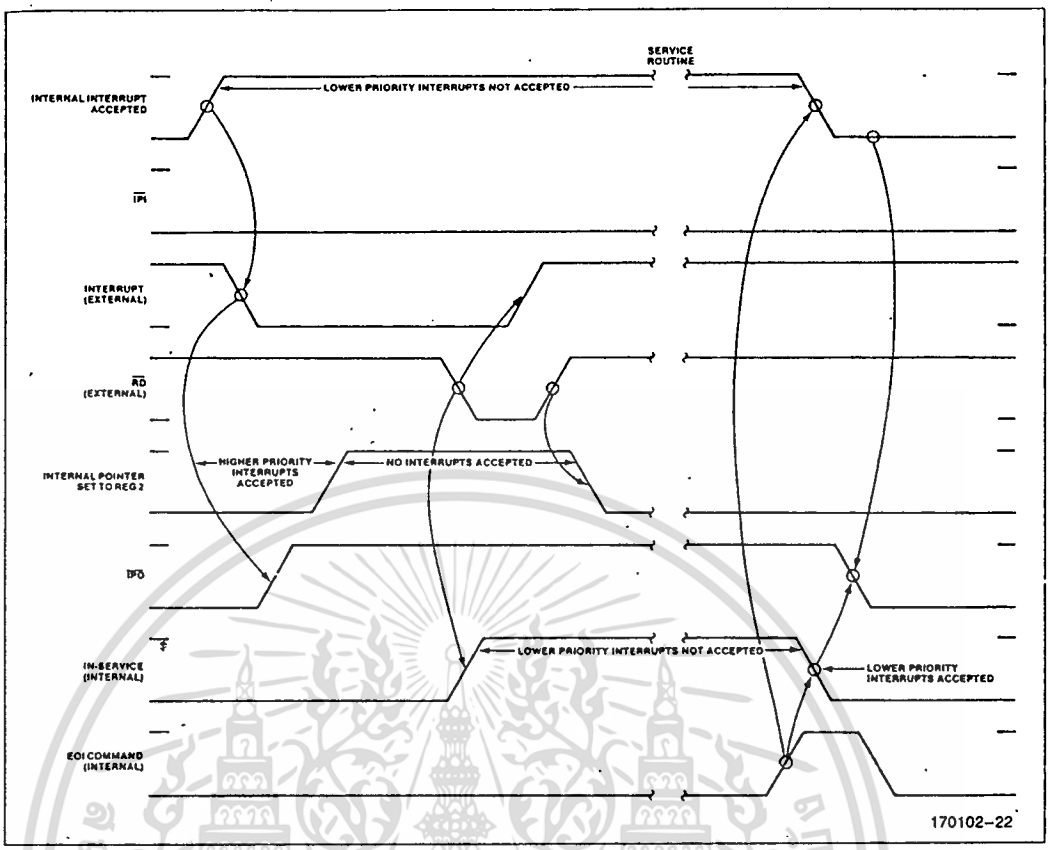


Lower priority interrupts are not accepted internally while the In-Service latch is set. However, higher priority interrupts are accepted internally and a new external $\overline{INT}$ request is generated. If the CPU responds with a new INTA sequence, the MPSC will respond as before, suspending the lower priority interrupt.

After the interrupt is serviced, the End-of-Interrupt (EOI) command should be written to the MPSC. This command will cause an internal pulse that is used to reset the In-Service Latch which allows service for lower priority interrupts in the daisy-chain to resume, provided a new INTA sequence does not start for a higher priority interrupt (higher than the highest under service). If there is no interrupt pending internally, the $\overline{IPO}$ follows $\overline{IPI}$.

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

Non-Vectored Interrupt Timing



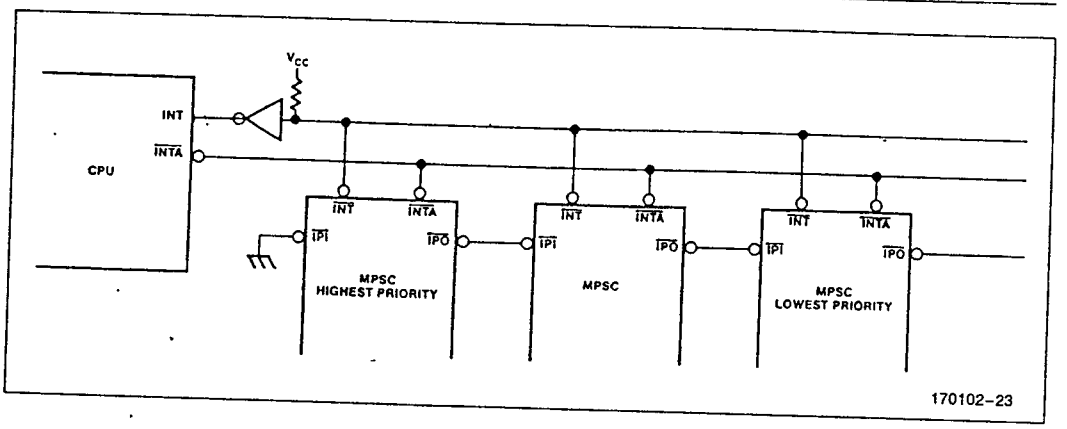
PRIORITY RESOLUTION: NON-VECTORED MODE

In non-vectored mode, the MPSC does not respond to interrupt acknowledge sequences. The $\overline{INTA}$ input (pin 27) must be pulled high for proper operation. The MPSC should be programmed to the Status-Affects-Vector mode, and the CPU should read RR2 (Ch. B) in its service routine to determine which interrupt requires service.

In this case, the internal pointer being set to RR2 provides the same function as the internal INTA signal in the vectored mode. It inhibits acceptance of any additional internal interrupts and its leading edge starts the interrupt priority resolution circuit. The interrupt priority resolution is ended by the leading edge of the read signal used by the CPU to retrieve the modified vector. The leading edge of read sets the In-Service latch and forces the external $\overline{INT}$ output inactive (high). The internal pointer is reset to zero after the trailing edge of the read pulse.

NOTE:
That if RR2 is specified but not read, no internal interrupts, regardless of priority, are accepted.

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้



DAISY CHAINING MPSC

In the vectored interrupt mode, multiple MPSC's can be daisy-chained on the same $\overline{INT}$, $\overline{INTA}$ signals. These signals, in conjunction with the $\overline{IPI}$ and $\overline{IPO}$ allow a daisy-chain-like interrupt resolution scheme. This scheme can be configured for either 8085 or 8086/88 based system.

In either mode, the same hardware configuration is called for. The $\overline{INT}$ request lines are wire-OR'ed together at the input of a TTL inverter which drives the INT pin of the CPU. The $\overline{INTA}$ signal from the CPU drives all of the daisy-chained MPSC's.

The MPSC drives $\overline{IPO}$ (Interrupt Priority Output) inactive (high) if $\overline{IPI}$ (Interrupt Priority Input) is inactive (high), or if the MPSC has an interrupt pending.

The $\overline{IPO}$ of the highest priority MPSC is connected to the $\overline{IPI}$ of the next highest priority MPSC, and so on.

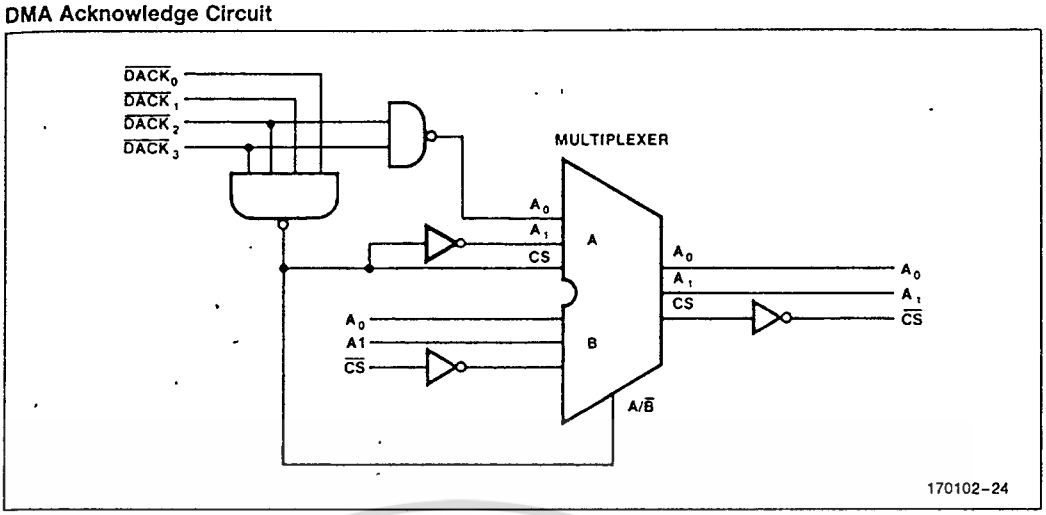
If $\overline{IPI}$ is active (low), the MPSC knows that all higher priority MPSC's have no interrupts pending. The $\overline{IPI}$ pin of the highest priority MPSC is strapped active (low) to ensure that it always has priority over the rest.

MPSC's Daisy-chained on an 8088/86 CPU should be programmed to the 8088/86 Interrupt mode (WR2; D4, D3 Ch. A). MPSC's Daisy-chained on an 8085 CPU should be programmed to 8085 interrupt mode 1 if it is the highest priority MPSC. In this mode, the highest priority MPSC issues the CALL instruction during the first INTA cycle, and the interrupting MPSC provides the interrupt vector during the following INTA cycles. Lower priority MPSC's should be programmed to 8085 interrupt mode 2.

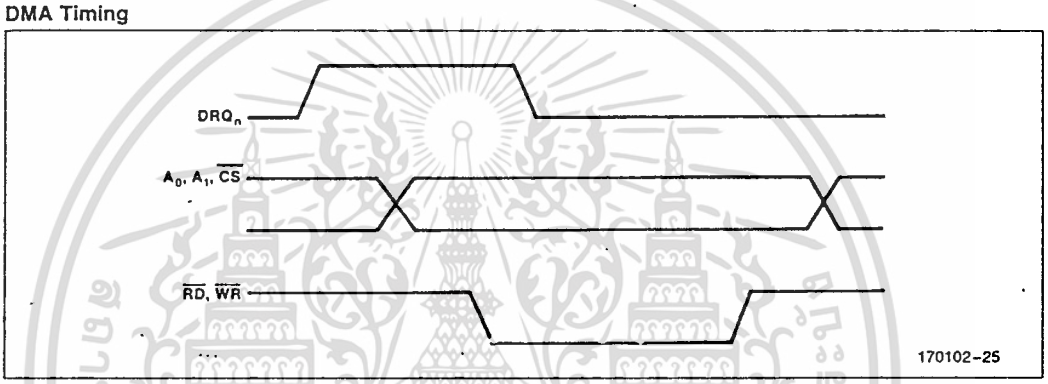
MPSC's used alone in 8085 systems should be programmed to 8085 mode 1 interrupt operation.

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

DMA Acknowledge Circuit



DMA Timing



DMA OPERATION

Each MPSC can be programmed to utilize up to four DMA channels: Transmit Channel A, Receive Channel A, Transmit Channel B, Receive Channel B. Each DMA Channel has an associated DMA Request line. Acknowledgement of a DMA cycle is done via normal data read or write cycles. This is accomplished by encoding the $\overline{DACK}$ signals to generate A_0, A_1, and $\overline{CS}$, and multiplexing them with the normal A_0, A_1, and $\overline{CS}$ signals.

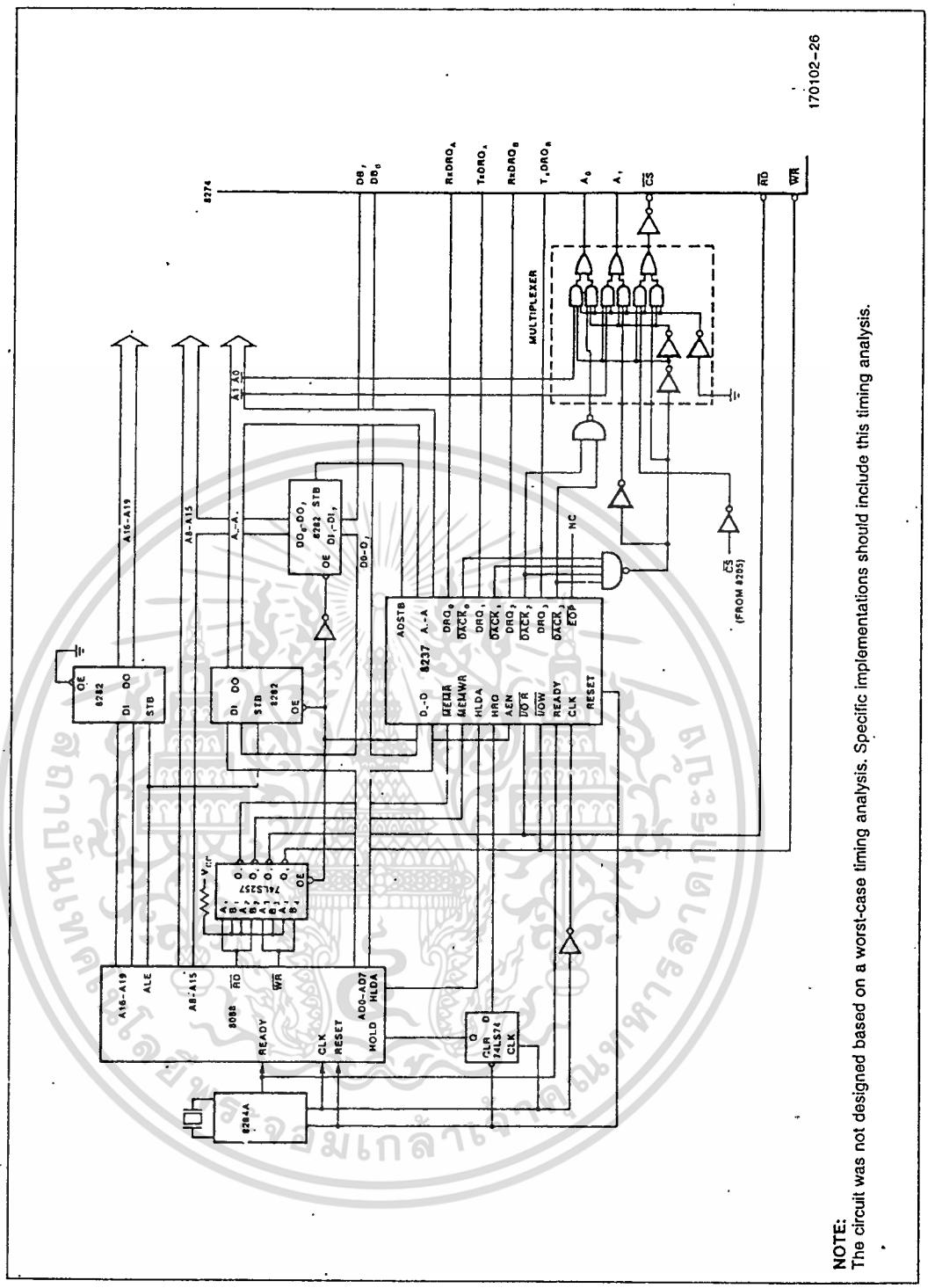
PERMUTATIONS

Channels A and B can be used with different system interface modes. In all cases it is possible to poll the MPSC. The following table shows the possible permutations of interrupt, wait, and DMA modes for channels A and B. Bits D_1, D_0 of WR2 Ch. A determine these permutations.

Permutation WR2 Ch. A D_1	D_0	Channel A	Channel B
0	0	Wait Interrupt Polled	Wait Interrupt Polled
0	1	DMA Polled	Interrupt Polled
1	0	DMA Polled	DMA Polled

NOTE:
D1, D0 = 1, 1 is illegal.

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้



NOTE:
The circuit was not designed based on a worst-case timing analysis. Specific implementations should include this timing analysis.

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

PROGRAMMING HINTS

This section will describe some useful programming hints which may be useful in program development.

Asynchronous Operation

At the end of transmission, the CPU must issue "Reset Transmit Interrupt/DMA Pending" command in WR0 to reset the last transmit empty request which was not satisfied. Failing to do so will result in the MPSC locking up in a transmit empty state forever.

Non-Vectored Mode

In non-vectored mode, the Interrupt Acknowledge pin (INTA) on the MPSC must be tied high through a pull-up resistor. Failing to do so will result in unpredictable response from the 8274.

HDLC/SDLC Mode

When receiving data in SDLC mode, the CRC bytes must be read by the CPU (or DMA controller) just like any other data field. Failing to do so will result in receiver buffer overflow. The CRC bytes are not to be used for CRC verification. Residue bits may be contained in the first CRC byte. Also, the End of Frame Interrupt indicates that the entire frame has been received. At this point, the CRC result (RR1: D6) and residue code (RR1; D3, D2, D1) may be checked.

Status Register RR2

RR2 contains the vector which gets modified to indicate the source of interrupt (See the section titled MPSC Modes of Operation). However, the state of the vector does not change if no new interrupts are generated. The contents of RR2 are only changed when a new interrupt is generated. In order to get the correct information, RR2 must be read only after an interrrupt is generated, otherwise it will indicate the previous state.

Initialization Sequence

The MPSC initialization routine must issue a channel Reset Command at the beginning. WR4 should be defined before other registers. At the end of the initialization sequence, Reset External/Status and Error Reset commands should be issued to clear any spurious interrupts which may have been caused at power up.

Transmit Under-run/EOM Latch

In SDLC/HDLC, bisync and monosync mode, the transmit under-run/EOM must be reset to enable the CRC check bytes to be appended to the transmit frame or transmit message. The transmit under-run/EOM latch can be reset only after the first character is loaded into the transmit buffer. When the transmitter under-runs at the end of the frame, CRC check bytes are appended to the frame/message. The transmit under-run/EOM latch can be reset at any time during the transmission after the first character. However, it should be reset *before* the transmitter under-runs otherwise, both bytes of the CRC may not be appended to the frame/message. In the receive mode in bisync operation, the CPU must read the CRC bytes and two more SYNC characters before checking for valid CRC result in RR1.

Sync Character Load Inhibit

In bisync/monosync mode only, it is possible to prevent loading sync characters into the receive buffers by setting the sync character load inhibit bit (WR3; D1 = 1). Caution must be exercised in using this option. It may be possible to get a CRC character in the received message which may match the sync character and not get transferred to the receive buffer. However, sync character load inhibit should be enabled during all pre-frame sync characters so the software routine does not have to read them from the MPSC.

In SDLC/HDLC mode, sync character load inhibit bit must be reset to zero for proper operation.

EOI Command

EOI command can only be issued through channel A irrespective of which channel had generated the interrupt.

Priority in DMA Mode

There is no priority in DMA mode between the following four signals: TxDRQ(CHA), RxDRQ(CHA), TxDRQ(CHB), RxDRQ(CHB). The priority between these four signals must be resolved by the DMA controller. At any given time, all four DMA channels from the 8274 are capable of going active.

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

ABSOLUTE MAXIMUM RATINGS*

Ambient Temperature
Under Bias ..... 0°C to +70°C
Storage Temperature
(Ceramic Package) ..... −65°C to +150°C
(Plastic Package) ..... −40°C to +125°C
Voltage on Any Pin with
Respect to Ground ..... −0.5V to +7.0V

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

D.C. CHARACTERISTICS T_A = 0°C to +70°C; V_{CC} = +5V ±10%

Symbol	Parameter	Min	Max	Units	Test Conditions
V_{IL}	Input Low Voltage	−0.5	+0.8	V	
V_{IH}	Input High Voltage	+2.0	V_{CC} +0.5	V	
V_{OL}	Output Low Voltage		+0.45	V	I_{OL} = 2.0 mA
V_{OH}	Output High Voltage	+2.4		V	I_{OH} = 200 µA
I_{IL}	Input Leakage Current		±10	µA	V_{IN} = V_{CC} to 0V
I_{OFL}	Output Leakage Current		±10	µA	V_{OUT} = V_{CC} to 0.45V
I_{CC}	V_{CC} Supply Current		200	mA	

NOTE:
1. For Extended Temperature EXPRESS, use MIL8274 electrical Parameters.

CAPACITANCE T_A = 25°C; V_{CC} = GND = 0V

Symbol	Parameter	Min	Max	Units	Test Conditions
C_{IN}	Input Capacitance		10	pF	f_c = 1 MHz
C_{OUT}	Output Capacitance		15	pF	Unmeasured pins returned to GND
$C_{I/O}$	Input/Output Capacitance		20	pF	

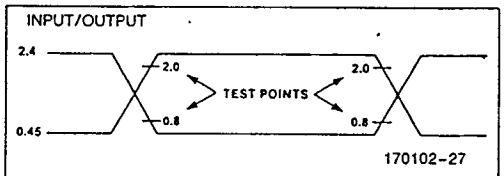
เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

A.C. CHARACTERISTICS T_A = 0°C to +70°C; V_{CC} = +5V ±10%

Symbol	Parameter	Min	Max	Units	Test Conditions
t_{CY}	CLK Period	250	4000	ns	
t_{CL}	CLK Low Time	105	2000	ns	
t_{CH}	CLK High Time	105	2000	ns	
t_r	CLK Rise Time	0	30	ns	
t_f	CLK Fall Time	0	30	ns	
t_{AR}	A0, A1 Setup to $\overline{RD}\downarrow$	0		ns	
t_{AD}	A0, A1 to Data Output Delay		200	ns	C_L = 150 pF
t_{RA}	A0, A1 Hold after $\overline{RD}\uparrow$	0		ns	
t_{RD}	$\overline{RD}\downarrow$ to Data Output Delay		200	ns	C_L = 150 pF
t_{RR}	$\overline{RD}$ Pulse Width	250		ns	
t_{DF}	Output Float Delay		120	ns	
t_{AW}	$\overline{CS}$, A0, A1 Setup to $\overline{WR}\downarrow$	0		ns	
t_{WA}	$\overline{CS}$, A0, A1 Hold after $\overline{WR}\uparrow$	0		ns	
t_{WW}	$\overline{WR}$ Pulse Width	250		ns	
t_{DW}	Data Setup to $\overline{WR}\uparrow$	150		ns	
t_{WD}	Data Hold after $\overline{WR}\uparrow$	0		ns	
t_{PI}	$\overline{IPI}$ Setup to $\overline{INTA}\downarrow$	0		ns	
t_{IP}	$\overline{IPI}$ Hold after $\overline{INTA}\uparrow$	10		ns	
t_{II}	$\overline{INTA}$ Pulse Width	250		ns	
t_{PIPO}	$\overline{IPI}\downarrow$ to $\overline{IPO}$ Delay		100	ns	
t_{ID}	$\overline{INTA}\downarrow$ to Data Output Delay		200	ns	
t_{CQ}	$\overline{RD}$ or $\overline{WR}$ to DRQ $\downarrow$		150	ns	
t_{RV}	Recovery Time Between Controls	300		ns	
t_{CW}	$\overline{CS}$, A0, A1 to RDY_A or RDY_B Delay		140	ns	
t_{DCY}	Data Clock Cycle	4.5		tcy	
t_{DCL}	Data Clock Low Time	180		ns	
t_{DCH}	Data Clock High Time	180		ns	
t_{TD}	$\overline{TxC}$ to TxD Delay (x1 Mode)		300	ns	
t_{DS}	RxD Setup to $\overline{RxC}\uparrow$	0		ns	
t_{DH}	RxD Hold after $\overline{RxC}\uparrow$	140		ns	
t_{ITD}	$\overline{TxC}$ to $\overline{INT}$ Delay	4	6	tcy	
t_{IRD}	RxC to $\overline{INT}$ Delay	7	10	tcy	
t_{PL}	$\overline{CTS}$, $\overline{CD}$, $\overline{SYNDET}$ Low Time	200		ns	
t_{PH}	$\overline{CTS}$, $\overline{CD}$, $\overline{SYNDET}$ High Time	200		ns	
t_{IPD}	External $\overline{INT}$ from $\overline{CTS}$, $\overline{CD}$, $\overline{SYNDET}$		500	ns	

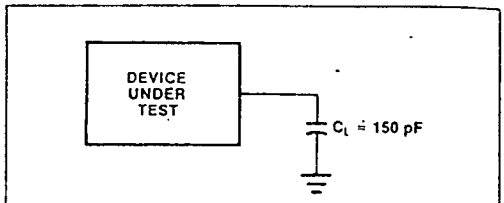
เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

A.C. TESTING INPUT/OUTPUT WAVEFORM



A.C. Testing; Inputs are driven at 2.4V for a Logic "1" and 0.45V for a Logic "0". Timing measurements are made at 2.0V for a Logic "1" and 0.8V for a Logic "0".

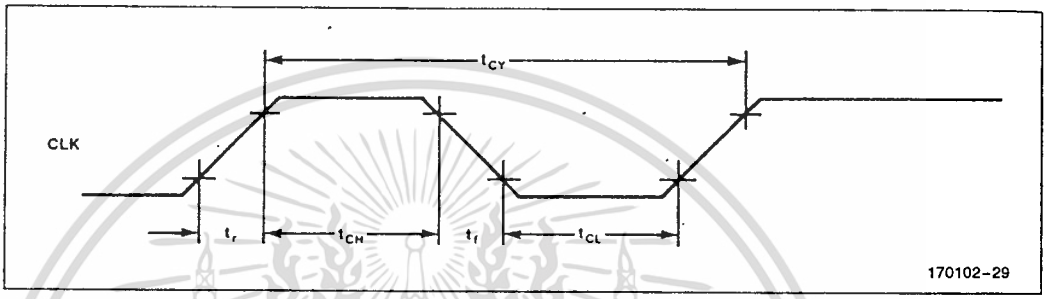
A.C. TESTING LOAD CIRCUIT



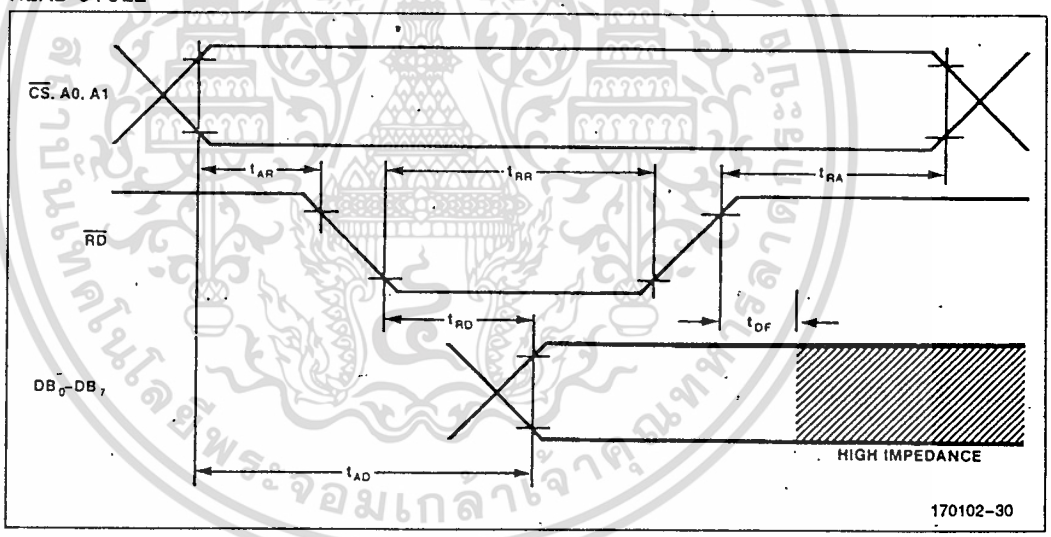
C_L = 100 pF
C_L Includes Jig Capacitance

WAVEFORMS

CLOCK CYCLE



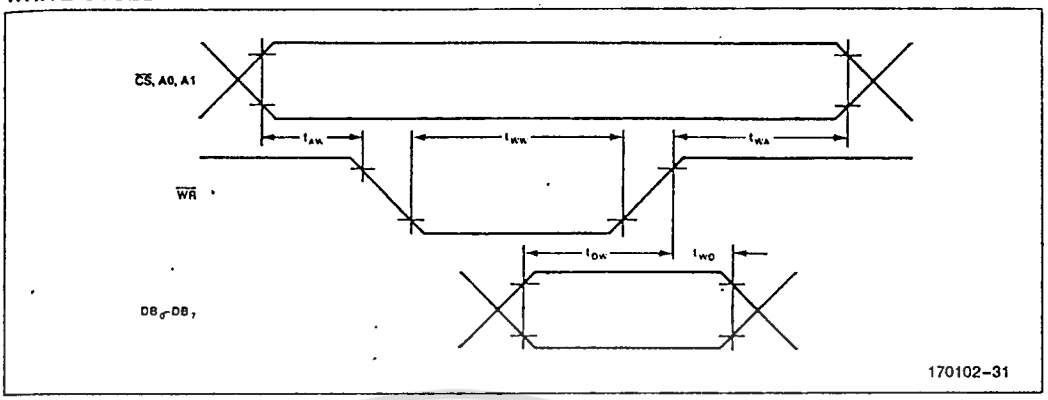
READ CYCLE



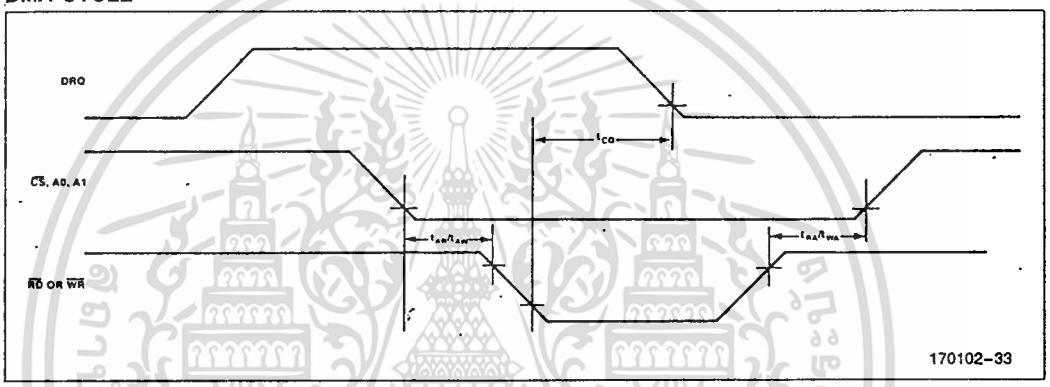
เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

WAVEFORMS (Continued)

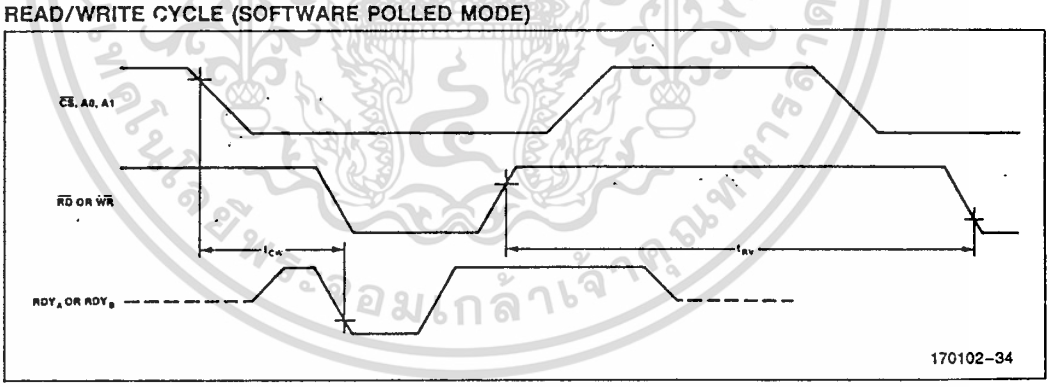
WRITE CYCLE



DMA CYCLE

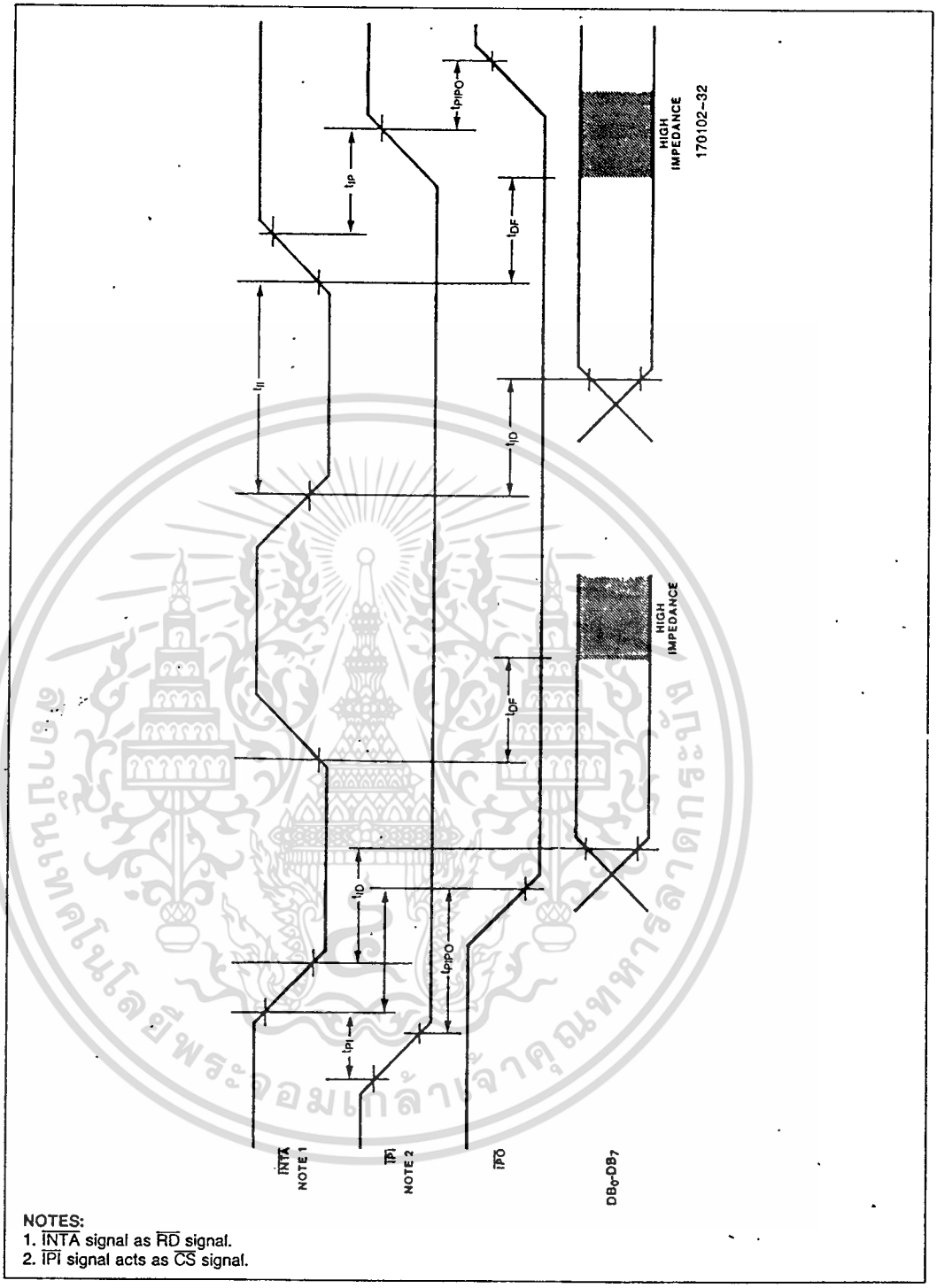


READ/WRITE CYCLE (SOFTWARE POLLED MODE)



เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

INTA CYCLE

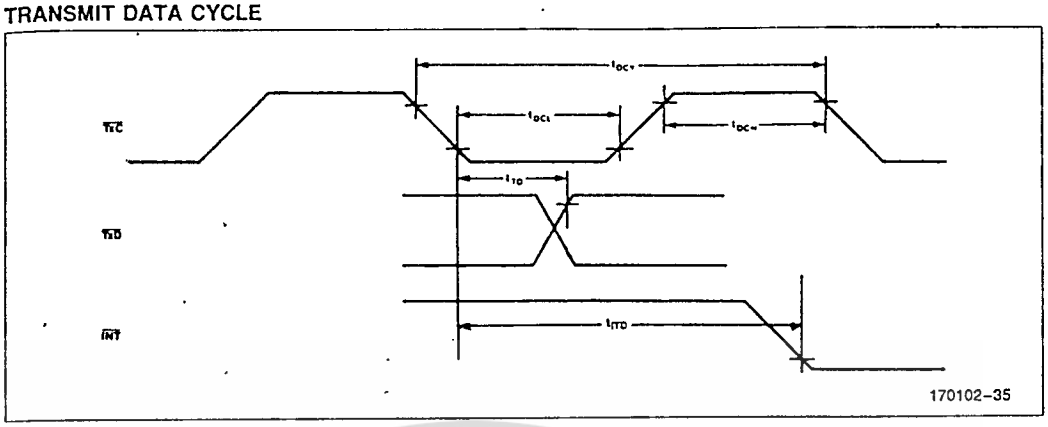


NOTES:
1. $\overline{INTA}$ signal as $\overline{RD}$ signal.
2. $\overline{IPI}$ signal acts as $\overline{CS}$ signal.

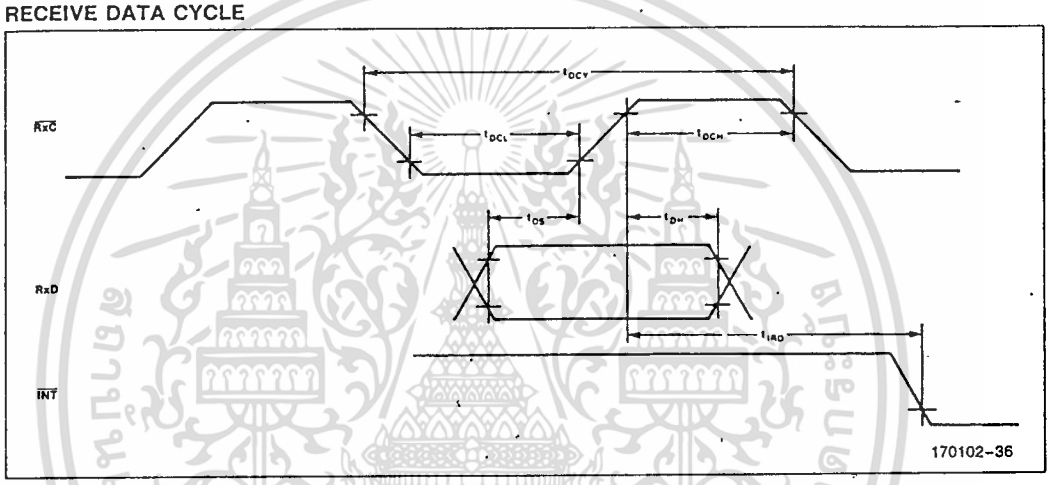
เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

WAVEFORMS. (Continued)

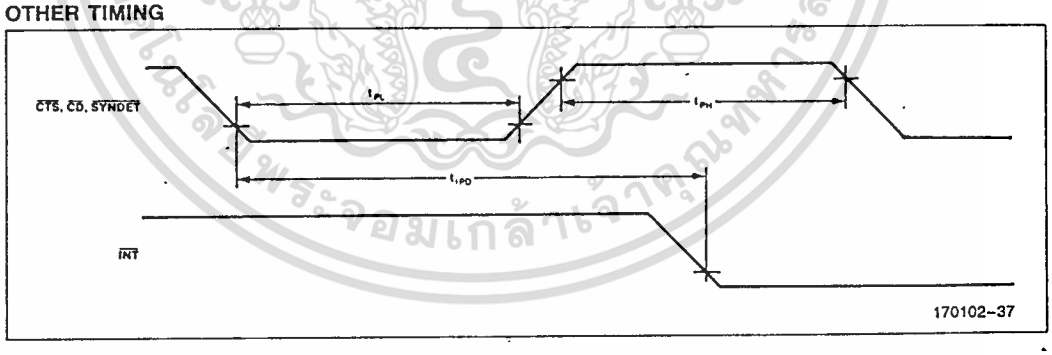
TRANSMIT DATA CYCLE



RECEIVE DATA CYCLE



OTHER TIMING



เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

intel

APPLICATION NOTE

AP-134

October 1986

Asynchronous Communication with the 8274 Multiple-Protocol Serial Controller

Order Number: 210311-002

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

INTRODUCTION

The 8274 Multiprotocol serial controller (MPSC) is a sophisticated dual-channel communications controller that interfaces microprocessor systems to high-speed serial data links (at speeds to 880K bits per second) using synchronous or asynchronous protocols. The 8274 interfaces easily to most common microprocessors (e.g., 8048, 8051, 8085, 8086, and 8088), to DMA controllers such as the 8237 and 8257, and to the 8089 I/O processor. Both MPSC communication channels are completely independent and can operate in a full-duplex communication mode (simultaneous data transmission and reception).

Communication Functions

The 8274 performs many communications-oriented functions, including:

— Converting data bytes from a microprocessor system into a serial bit stream for transmission over the data link to a receiving system.

— Receiving serial bit streams and reconverting the data into parallel data bytes that can easily be processed by the microprocessor system.

— Performing error checking during data transfers. Error checking functions include computing/transmitting error codes (such as parity bits or CRC bytes) and using these codes to check the validity of received data.

— Operating independently of the system processor in a manner designed to reduce the system overhead involved in data transfers.

System Interface

The MPSC system interface is extremely flexible, supporting the following data transfer modes:

1. Polled Mode. The system processor periodically reads (polls) an 8274 status register to determine when a character has been received, when a character is needed for transmission, and when transmission errors are detected.
2. Interrupt Mode. The MPSC interrupts the system processor when a character has been received, when a character is needed for transmission, and when transmission errors are detected.
3. DMA Mode. The MPSC automatically requests data transfers from system memory for both transmit and receive functions by means of two DMA request signals per serial channel. These DMA request signals may be directly interfaced to an 8237 or 8257 DMA controller or to an 8089 I/O processor.
4. WAIT Mode. The MPSC ready signal is used to synchronize processor data transfers by forcing the processor to enter wait states until the 8274 is ready for another data byte. This feature enables the 8274 to interface directly to an 8086 or 8088 processor by means of string I/O instructions for very high-speed data links.

Scope

This application note describes the use of the 8274 in asynchronous communication modes. Asynchronous communication is typically used to transfer data to/from video display terminals, modems, printers, and other low-to-medium-speed peripheral devices. Use of the 8274 in both interrupt-driven and polled system environments is described. Use of the DMA and WAIT modes are not described since these modes are employed mainly in synchronous communication systems where extremely high data rates are common. Programming examples are written in PL/M-86 (Appendix B and Appendix C). PL/M-86 is executed by the iAPX-86 and iAPX-88 processor families. In addition, PL/M-86 is very similar to PL/M-80 (executed by the MCS-80 and MCS-85 processor families). In addition, Appendix D describes a simple application example using an SDK-86 in an iAPX-86/88 environment.

SERIAL-ASYNCHRONOUS DATA LINKS

A serial asynchronous interface is a method of data transmission in which the receiving and transmitting systems need not be synchronized. Instead of transmitting clocking information with the data, locally generated clocks (16, 32 or 64 times as fast as the data transmission rate) are used by the transmitting and receiving systems. When a character of information is sent by the transmitting system, the character data is framed (preceded and followed) by special START and STOP bits. This framing information permits the receiving system to temporarily synchronize with the data transmission. (Refer to Figure 1 during the following discussion of asynchronous data transmission.)

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

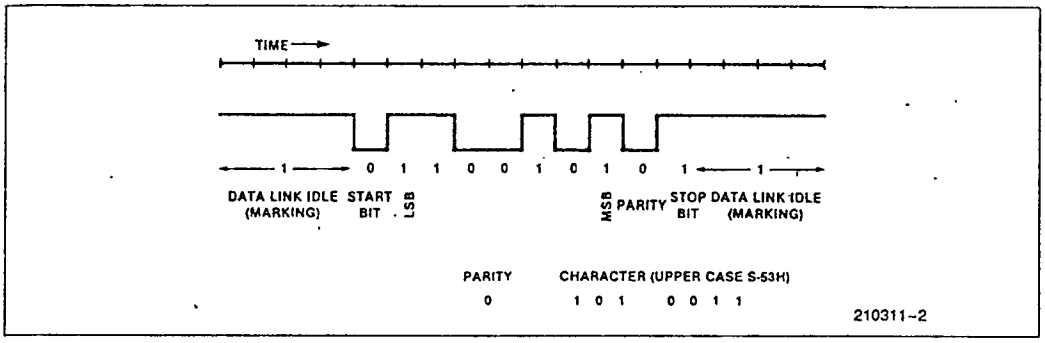


Figure 1. Transmission of a 7-Bit ASCII Character with Even Parity

Normally the data link is in an idle or marking state, continuously transmitting a "mark" (binary 1). When a character is to be sent, the character data bits are immediately preceded by a "space" (binary 0 START bit). The mark-to-space transition informs the receiving system that a character of information will immediately follow the start bit. Figure 1 illustrates the transmission of a 7-bit ASCII character (upper case S) with even parity. Note that the character is transmitted immediately following the start bit. Data bits within the character are transmitted from least-significant to most-significant. The parity bit is transmitted immediately following the character data bits and the STOP framing bit (binary 1) signifies the end of the character.

Asynchronous interfaces are often used with human interface devices such as CRT/keyboard units where the time between data transmissions is extremely variable.

Characters

In asynchronous mode, characters may vary in length from five to eight bits. The character length depends on the coding method used. For example, five-bit characters are used when transmitting Baudot Code, seven-bit characters are required for ASCII data, and eight-bit characters are needed for EBCDIC and binary data. To transmit messages composed of multiple characters, each character is framed and transmitted separately (Figure 2).

This framing method ensures that the receiving system can easily synchronize with the start and stop bits of each character, preventing receiver synchronization errors. In addition, this synchronization method makes both transmitting and receiving systems insensitive to possible time delays between character transmissions.

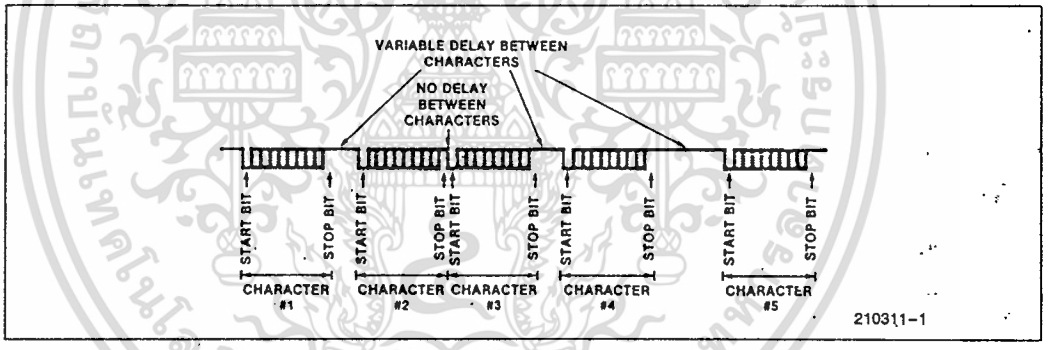


Figure 2. Multiple Character Transmission

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

Framing

Character framing is accomplished by the START and STOP bits described previously. When the START bit transition (mark-to-space) is detected, the receiving system assumes that a character of data will follow. In order to test this assumption (and isolate noise pulses on the data link), the receiving system waits one-half bit time and samples the data link again. If the link has returned to the marking state, noise is assumed, and the receiver waits for another START bit transition.

When a valid START bit is detected, the receiver samples the data link for each bit of the following character. Character data bits and the parity bit (if required) are sampled at their nominal centers until all required characters are received. Immediately following the data bits, the receiver samples the data link for the STOP bit, indicating the end of the character. Most systems permit specification of 1, 1½, or 2 stop bits.

Timing

The transmitter and receiver in an asynchronous data link arrangement are clocked independently. Normally, each clock is generated locally and the clocks are not synchronized. In fact, each clock may be a slightly different frequency. (In practice, the frequency difference should not exceed a few percent. If the transmitter and receiver clock rates vary substantially, errors will occur because data bits may be incorrectly identified as START or STOP framing bits.) These clocks are designed to operate at 16, 32, or 64 times the communications data rate. These clock speeds allow the receiving device to correctly sample the incoming bit stream.

Serial-interface data rates are measured in bits/second. The term "baud" is used to specify the number of times per second that the transmitted signal level can change states. In general, the baud is not equal to the bit rate. Only when the transmitted signal has two states (electrical levels) is the baud rate equal to the bit rate. Most point-to-point serial data links use RS-232-C, RS-422, or RS-423 electrical interfaces. These specifications call for two electrical signal levels (the baud is equal to the bit rate). Modem interfaces, however, may often have differing bit and baud rates.

While there are generally no limitations on the data transmission rates used in an aysnchronous data link, a limited set of rates has been standardized to promote equipment interconnection. These rates vary from 75 bits per second to 38,400 bits per second. Table 1 illustrates typical asynchronous data rates and the associated clock frequencies required for the transmitter and receiver circuits.

Table 1. Communication Data Rates and Associated Transmitter/Receiver Clock Rates

Data Rate (Bits/Second)	Clock Rate (kHz)		
	X16	X32	X64
75	1.2	2.4	4.8
150	2.4	4.8	9.6
300	4.8	9.6	19.2
600	9.6	19.2	38.4
1200	19.2	38.4	76.8
2400	38.4	76.8	153.6
4800	76.8	153.6	307.2
9600	153.6	307.2	614.2
19200	307.2	614.4	—
38400	614.4	—	—

Parity

In order to detect transmission errors, a parity bit may be added to the character data as it is transferred over the data link. The parity bit is set or cleared to make the total number of "one" bits in the character even (even parity) or odd (odd parity). For example, the letter "A" is represented by the seven-bit ASCII code 1000001 (41H). The transmitted data code (with parity) for this character contains eight bits; 01000001 (41H) for even parity and 11000001 (OC1H) for odd parity. Note that a single bit error changes the parity of the received character and is therefore easily detected. The 8274 supports both odd and even parity checking as well as a parity disable mode to support binary data transfers.

Communication Modes

Serial data transmission between two devices can occur in one of three modes. In the simplex transmission mode, a data link can transmit data in one direction only. In the half-duplex mode, the data link can transmit data in both directions, but not simultaneously. In the full-duplex mode (the most common), the data link can transmit data in both directions simultaneously. The 8274 directly supports the full-duplex mode and will interface to simplex and half-duplex communication data links with appropriate software controls.

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

BREAK Condition

Asynchronous data links often include a special sequence known as a break condition. A break condition is initiated when the transmitting device forces the data link to a spacing state (binary 0) for an extended length of time (typically 150 milliseconds). Many terminals contain keys to initiate a break sequence. Under software control, the 8274 can initiate a break sequence when transmitting data and detect a break sequence when receiving data.

MPSC SYSTEM INTERFACE

Hardware Environment

The 8274 MPSC interfaces to the system processor over an 8-bit data bus. Each serial I/O channel responds to two I/O or memory addresses as shown in Table 2. In addition, the MPSC supports non-vectored and vectored interrupts.

The 8274 may be configured for memory-mapped or I/O-mapped operation.

The 8274-processor hardware interface can be configured in a flexible manner, depending on the operating mode selected—polled, interrupt-driven, DMA, or WAIT. Figure 3 illustrates typical MPSC configurations for use with an 8088 microprocessor in the polled and interrupt-driven modes.

All serial-to-parallel conversion, parallel-to-serial conversion, and parity checking required during asynchronous serial I/O operation is automatically performed by the MPSC.

Operational Interface

Command, parameter, and status information is stored in 21 registers within the MPSC (8 writable registers and 2 readable registers for each channel, plus the interrupt vector register). These registers are all accessed by means of the command/status ports for each channel. An internal pointer register selects which of the command or status registers will be written or read during a command/status access of an MPSC channel. Figure 4 diagrams the command/status register architecture for each serial channel. In the following discussion, the writable registers will be referred to as WR0 through WR7 and the readable registers will be referred to as RR0 through RR2.

Table 2. 8274 Addressing

$\overline{CS}$	A_1	A_0	Read Operation	Write Operation
0	0	0	Ch. A Data Read	Ch. A Data Write
0	1	0	Ch. A Status Read	Ch. A Command/Parameter
0	0	1	Ch. B Data Read	Ch. B Data Write
0	1	1	Ch. B Status Read	Ch. B Command/Parameter
1	X	X	High Impedance	High Impedance

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

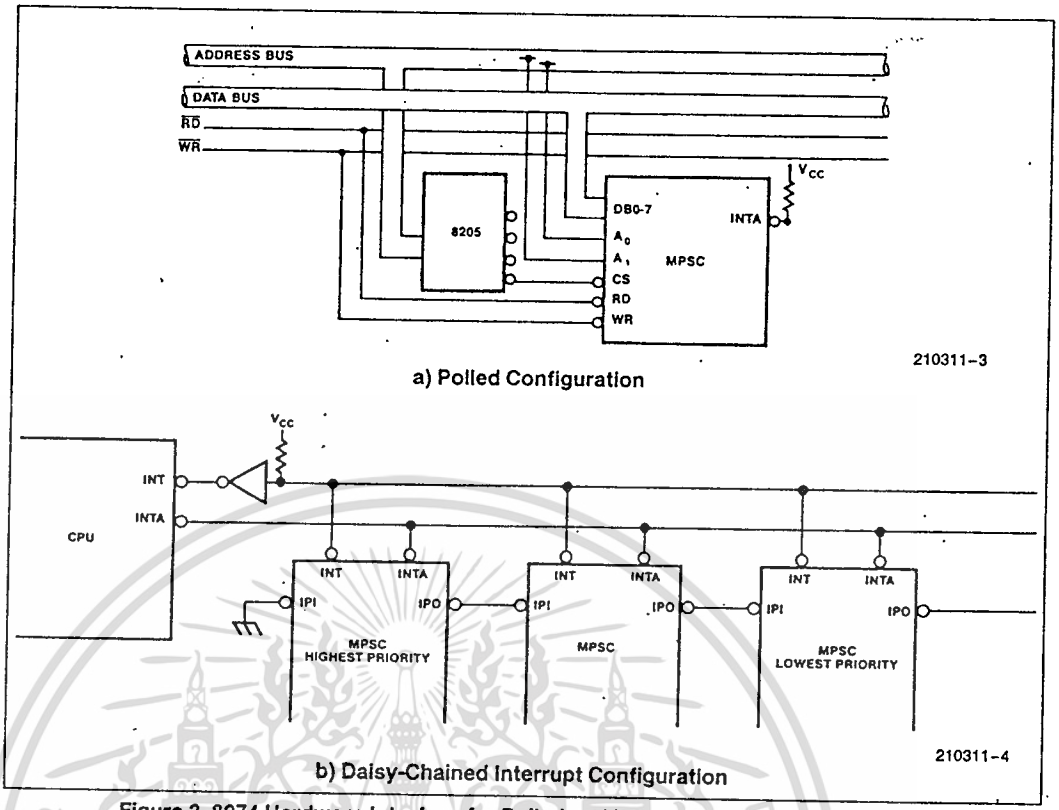


Figure 3. 8274 Hardware Interface for Polled and Interrupt-Driven Environments

The least-significant three bits of WR0 are automatically loaded into the pointer register every time WR0 is written. After reset, WR0 is set to zero so that the first write to a command register causes the data to be loaded into WR0 (thereby setting the pointer register). After WR0 is written, the following read or write accesses the register selected by the pointer. The pointer is reset after the read or write operation is completed. In this manner, reading or writing an arbitrary MPSC channel register requires two I/O accesses. The first access is always a write command. This write command is used to set the pointer register. The second access is either a read or a write command; the pointer register (previously set) will ensure that the correct internal register is read or written. After this second access, the pointer register is automatically reset. Note that writing WR0 and reading RR0 does not require presetting of the pointer register.

During initialization and normal MPSC operation, various registers are read and/or written by the system processor. These actions are discussed in detail in the following paragraphs. Note that WR6 and WR7 are not used in the asynchronous communication modes.

RESET

When the 8274 $\overline{\text{RESET}}$ line is activated, both MPSC channels enter the idle state. The serial output lines are forced to the marking state (high) and the modem interface signals ($\overline{\text{RTS}}$, $\overline{\text{DTR}}$) are forced high. In addition, the pointer register is set to zero.

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

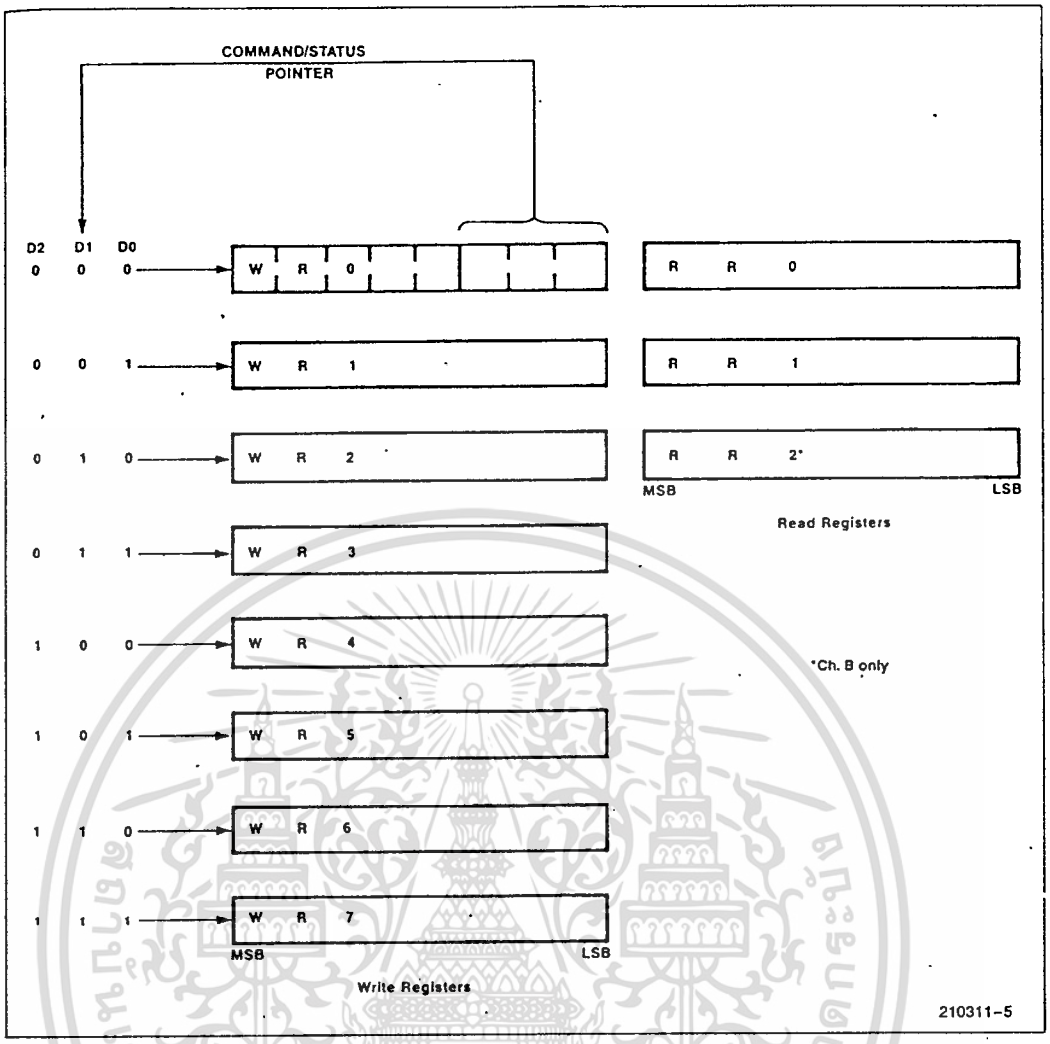


Figure 4. Command/Status Register Architecture (Each Serial Channel)

External/Status Latches

The MPSC continuously monitors the state of four external/status conditions:

1. CTS—clear-to-send input pin.
2. CD—carrier-detect input pin.
3. SYNDET—sync-detect input pin. This pin may be used as a general-purpose input in the asynchronous communication mode.
4. BREAK—a break condition (series of space bits on the receiver input pin).

A change of state in any of these monitored conditions will cause the associated status bit in RR0 (Appendix A) to be latched (and optionally cause an interrupt).

Error Reporting

Three error conditions may be encountered during data reception in the asynchronous mode:

1. Parity. If parity bits are computed and transmitted with each character and the MPSC is set to check parity (bit 0 in WR4 is set), a parity error will occur whenever the number of "1" bits within the character (including the parity bit) does not match the odd/even setting of the parity check flag (bit 1 in WR4).

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

2. Framing. A framing error will occur if a stop bit is not detected immediately following the parity bit (if parity checking is enabled) or immediately following the most-significant data bit (if parity checking is not enabled).
3. Overrun. If an input character has been assembled but the receiver buffers are full (because the previously received characters have not been read by the system processor), an overrun error will occur. When an overrun error occurs, the input character that has just been received will overwrite the immediately preceding character.

Transmitter/Receiver Initialization

In order to operate in the asynchronous mode, each MPSC channel must be initialized with the following information:

1. Clock Rate. This parameter is specified by bits 6 and 7 of WR4. The clock rate may be set to 16, 32, or 64 times the data-link bit rate. (See Appendix A for WR4 details.)
2. Number of Stop Bits. This parameter is specified by bits 2 and 3 of WR4. The number of stop bits may be set to 1, 1½, or 2. (See Appendix A for WR4 details.)
3. Parity Selection. Parity may be set for odd, even, or no parity by bits 0 and 1 of WR4. (See Appendix A for WR4 details.)
4. Receiver Character Length. This parameter sets the length of received characters to 5, 6, 7, or 8 bits. This parameter is specified by bits 6 and 7 of WR3. (See Appendix A for WR3 details.)
5. Receiver Enable. The serial-channel receiver operation may be enabled or disabled by setting or clearing bit 0 of WR3. (See Appendix A for WR3 details.)
6. Transmitter Character Length. This parameter sets the length of transmitted characters to 5, 6, 7, or 8 bits. This parameter is specified by bits 5 and 6 of WR5. (See Appendix A for WR5 details.) Characters of less than 5 bits in length may be transmitted by setting the transmitted length to five bits (set bits 5 and 6 of WR5 to 1).

   The MPSC then determines the actual number of bits to be transmitted from the character data byte. The bits to be transmitted must be right justified in the data byte, the next three bits must be set to 0 and all remaining bits must be set to 1. The following table illustrates the data formats for transmission of 1 to 5 bits of data:

D7	D6	D5	D4	D3	D2	D1	D0	Number of Bits Transmitted (Character Length)
1	1	1	1	0	0	0	c	1
1	1	1	0	0	0	c	c	2
1	1	0	0	0	c	c	c	3
1	0	0	0	c	c	c	c	4
0	0	0	c	c	c	c	c	5

7. Transmitter Enable. The serial channel transmitter operation may be enabled or disabled by setting or clearing bit 3 of WR5. (See Appendix A for WR5 details.)

For data transmissions via a modem or RS-232-C interface, the following information must also be specified:

1. Request-to-Send/Data-Terminal-Ready. Must be set to indicate status of data terminal equipment. Request-to-send is controlled by bit 1 of WR5 and data terminal ready is controlled by bit 7. (See Appendix A for WR5 details.)
2. Auto Enable. May be set to allow the MPSC to automatically enable the channel transmitter when the clear-to-send signal is active and to automatically enable the receiver when the carrier-detect signal is active. Auto Enable is controleld by bit 5 of WR3. (See Appendix A for WR3 details.)

During initialization, it is desirable to guarantee that the external/status latches reflect the latest interface information. Since up to two state changes are internally stored by the MPSC, at least two Reset External/Status Interrupt commands must be issued. This procedure is most easily accomplished by simply issuing this reset command whenever the pointer register is set during initialization.

An MPSC initialization procedure (MPSCRXINIT) for asynchronous communication is listed in Appendix B. Figure 5 illustrates typical MPSC initialization parameters for use with this procedure.

```
call MPSC$RXSINIT(41,  1,1,0,1,  3,1,1,  3,1,1,0,1);

        initializes the 8274 at address 41 as follows:

X16 clock rate          Enable transmitter
1 stop bit                and receiver
Odd parity              Auto enable set
8-bit characters        DTR and RTS set
  (Tx and Rx)           Break transmission disabled
```

Figure 5. Sample 8274 Initialization Procedure for Polled Operation

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

Polled Operation

In the polled mode, the processor must monitor the MPSC status by testing the appropriate bits in the read register. Data available, status, and error conditions are represented in RR0 and RR1 for channels A and B. An example of MPSC-polled transmitter/receiver routines are given in Appendix B. The following routines are detailed:

1. MPSC$POLL$RCV$CHARACTER—This procedure receives a character from the serial data link. The routine waits until the character-available flag in RR0 has been set. When this flag indicates that a character is available, RR1 is checked for errors (overrun, parity, or framing). If an error is detected, the character in the MPSC receive buffer must be read and discarded and the error routine (RECEIVE$ERROR) is called. If no receive errors have been detected, the character is input from the 8274 data port and returned to the calling program.

   MPSC$POLL$RCV$CHARACTER requires three parameters—the address of the 8274 channel data port (data$port), the address of the 8274 channel command port (cmd$port), and the address of a byte variable in which to store the received character (character$ptr).

2. MPSC$POLL$TRAN$CHARACTER—This procedure transmits a character to the serial data link. The routine waits until the transmitter-buffer-empty flag has been set in RR0 before writing the character to the 8274.

   MPSC$POLL$TRAN$CHARACTER requires three parameters—the address of the 8274 channel data port (data$port), the address of the 8274 channel command port (cmd$port), and the character of data that is to be transmitted (character).

3. RECEIVE$ERROR—This procedure processes receiver errors. First, an Error Reset command is written to the affected channel. All additional error processing is dependent on the specific application. For example, the receiving device may immediately request retransmission of the character or wait until a message has been completed.

   RECEIVE$ERROR requires two parameters—the address of the affected 8274 command port (cmd$port) and the error status (status) from 8274 register RR1.

Interrupt-Driven Operation

In an interrupt-driven environment, all receiver operations are reported to the system processor by means of interrupts. Once a character has been received and assembled, the MPSC interrupts the system processor. The system processor must then read the character from the MPSC data buffer and clear the current interrupt. During transmission, the system processor starts serial I/O by writing the first character of a message to the MPSC. The MPSC interrupts the system processor whenever the next character is required (i.e., when the transmitter buffer is empty) and the processor responds by writing the next character of the message to the MPSC data port for the appropriate channel.

By using interrupt-driven I/O, the MPSC proceeds independently of the system processor, signalling the processor only when characters are required for transmission, when characters are received from the data link, or when errors occur. In this manner, the system processor may continue execution of other tasks while serial I/O is performed concurrently.

Interrupt Configurations

The 8274 is designed to interface to 8085- and 8086-type processors in much the same manner as the 8259A is designed. When operating in the 8085 mode, the 8274 causes a "call" to a prespecified, interrupt-service routine location. In the 8086 mode, the 8274 presents the processor with a one-byte interrupt-type number. This interrupt-type number is used to "vector" through the 8086 interrupt service table. In either case, the interrupt service address or interrupt-type number is specified during MPSC initialization.

To shorten interrupt latency, the 8274 can be programmed to modify the prespecified interrupt vector so that no software overhead is required to determine the cause of an interrupt. When this "status affects vector" mode is enabled, the following eight interrupts are differentiated automatically by the 8274 hardware:

1. Channel B Transmitter Buffer Empty.
2. Channel B External/Status Transition.
3. Channel B Character Available.
4. Channel B Receive Error.
5. Channel A Transmitter Buffer Empty.
6. Channel A External/Status Transition.
7. Channel A Character Available.
8. Channel A Receive Error.

Interrupt Sources/Priorities

The 8274 has three interrupt sources for each channel:

1. Receiver (RxA, RxB). An interrupt is initiated when a character is available in the receiver buffer or when a receiver error (parity, framing, or overrun) is detected.
2. Transmitter (TxA, TxB). An interrupt is initiated when the transmitter buffer is empty and the 8274 is ready to accept another character for transmission..

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

3. External/Status (ExTA, ExTB). An interrupt is initiated when one of the external/status conditions (CDE, CTS, SYNDET, BREAK) changes state.

The 8274 supports two interrupt priority orderings (selectable during MPSC initialization) as detailed in Appendix A, WR2, CH-A.

Interrupt Initialization

In addition to the initialization parameters required for polled operation, the following parameters must be supplied to the 8274 to specify interrupt operation:

1. Transmit Interrupt Enable. Transmitter-buffer-empty interrupts are separately enabled by bit 1 of WR1. (See Appendix A for WR1 details.)
2. Receive Interrupt Enable. Receiver interrupts are separately enabled in one of three modes: a) interrupt on first received character only and on receive errors (used for message-oriented transmission systems), b) interrupt on all received characters and on receive errors, but do not interrupt on parity errors, and c) interrupt on all received characters and on receive errors (including parity errors). The ability to separately disable parity interrupts can be extremely useful when transmitting messages. Since the parity error bit in RR1 is latched, it will not be reset until an error reset operation is performed. Therefore, the parity error bit will be set if any parity errors were detected in a multi-character message. If this mode is used, the serial I/O software must poll the parity error bit at the completion of a message and issue an error reset if appropriate. The receiver interrupt mode is controlled by bits 3 and 4 of WR1. (See Appendix A for WR1 details.)
3. External/Status Interrupts. External/Status interrupts can be separately enabled by bit 0 of WR1. (See Appendix A for WR1 details.)
4. Interrupt Vector. An eight-bit interrupt-service routine location (8085) or interrupt type (8086) is specified through WR2 of channel B. (See Appendix A for WR2 details.) Table 3 lists interrupt vector addresses generated by the 8274 in the "status affects vector" mode.
5. "Status Affects Vector" Mode. The 8274 will automatically modify the interrupt vector if bit 3 of WR1 is set. (See Appendix A for WR1 details.)
6. System Configuration. Specifies the 8274 data transfer mode. Three configuration modes are available: a) interrupt-driven operation for both channels, b) DMA operation for both channels, and c) DMA operation for channel A, interrupt-driven operation for channel B. The system configuration is specified by means of bits 0 and 1 of WR2 (channel A). (See Appendix A for WR2 details.)
7. Interrupt Priorities. The 8274 permits software specification of receive/transmit priorities by means of bit 2 of WR2 (channel A). (See Appendix A for WR2 details.)
8. Interrupt Mode. Specifies whether the MPSC is to operate in a non-vectored mode (for use with an external interrupt controller), in an 8086-vectored mode, or in an 8085-vectored mode. This parameter is specified through bits 3 and 4 of WR2 (channel A). (See Appendix A for WR2 details.)

An MPSC interrupt initialization procedure (MPSCSINT$INIT) is listed in Appendix C.

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

Table 3. MPSC-Generated Interrupt Vectors in "Status Affects Vector" Mode

V7	V6	V5	V4	V3	V2	V1	V0	V7	V6	V5	V4	V3	V2	V1	V0	Original Vector (Specified during Initialization)
8086 Interrupt Type								8085 Interrupt Location								Interrupt Condition
V7	V6	V5	V4	V3	0	0	0	V7	V6	V5	0	0	0	V1	V0	Channel B Transmitter Buffer Empty
V7	V6	V5	V4	V3	0	0	1	V7	V6	V5	0	0	1	V1	V0	Channel B External/Status Change
V7	V6	V5	V4	V3	0	1	0	V7	V6	V5	0	1	0	V1	V0	Channel B Receiver Character Available
V7	V6	V5	V4	V3	0	1	1	V7	V6	V5	0	1	1	V1	V0	Channel B Receive Error
V7	V6	V5	V4	V3	1	0	0	V7	V6	V5	1	0	0	V1	V0	Channel A Transmitter Buffer Empty
V7	V6	V5	V4	V3	1	0	1	V7	V6	V5	1	0	1	V1	V0	Channel A External/Status Change
V7	V6	V5	V4	V3	1	1	0	V7	V6	V5	1	1	0	V1	V0	Channel A Receiver Character Available
V7	V6	V5	V4	V3	1	1	1	V7	V6	V5	1	1	1	V1	V0	Channel A Receive Error

Interrupt Service Routines

Appendix C lists four interrupt service procedures, a buffer transmission procedure, and a buffer reception procedure that illustrate the use of the 8274 in interrupt-driven environments. Use of these procedures assumes that the 8086/8088 interrupt vector is set to 20H and that channel B is used with the "status affects vector" mode enabled.

1. TRANSMIT$BUFFER—This procedure begins serial transmission of a data buffer. Two parameters are required—a pointer to the buffer (buf$ptr) and the length of the buffer (buf$length). The procedure first sets the global buffer pointer, buffer length, and initial index for the transmitter-interrupt service routine and initiates transmission by writing the first character of the buffer to the 8274. The procedure then enters a wait loop until the I/O completion status is set by the transmit-interrupt service routine (MPSC$TRANSMIT$CHARACTER$INT).
2. RECEIVE$BUFFER—This procedure inputs a line (terminated by a line feed) from a serial I/O port. Two parameters are required—a pointer to the input buffer (buf$ptr) and a pointer to the buffer length variable (buf$length$ptr). The buffer length will be set by this procedure when the complete line has been input. The procedure first sets the global buffer pointer and initial index for the receiver interrupt service routine. RECEIVE$BUFFER then enters a wait loop until the I/O completion status is set by the receive interrupt routine (MPSC$RECEIVE$CHARACTER$INT).
3. MPSC$TRANSMIT$CHARACTER$INT—This procedure is executed when the MPSC Tx-buffer-empty interrupt is acknowledged. If the current transmit buffer index is less than the buffer length, the next character in the buffer is written to the MPSC data port and the buffer pointer is updated. Otherwise, the transmission complete status is posted.
4. MPSC$RECEIVE$CHARACTER$INT—This procedure is executed when a character has been assembled by the MPSC and the MPSC has issued a character-available interrupt. If no input buffer has been set up by RECEIVE$BUFFER, the character is ignored. If a buffer has been set up, but it is full, a receive overrun error is posted. Otherwise, the received character is read from the MPSC data port and the buffer index is updated. Finally, if the received character is a line feed, the reception complete status is posted.
5. RECEIVE$ERROR$INT—This procedure is executed when a receive error is detected. First, the error conditions are read from RR1 and the character currently in the MPSC receive buffer is read and discarded. Next, an Error Reset command is written to the affected channel. All additional error processing is application dependent.
6. EXTERNAL$STATUS$CHANGE$INT—This procedure is executed when an external status condition change is detected. The status conditions are read from RR0 and a Reset External/Status Interrupt command is issued. Further error processing is application dependent.

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

DATA LINK INTERFACE

Serial Data Interface

Each serial I/O channel within the 8274 MPSC interfaces to two data link lines—one line for transmitting data and one for receiving data. During transmission, characters are converted from parallel data format (as supplied by the system processor or DMA device) into a serial bit stream (with START and STOP bits) and clocked out on the TxD pin. During reception, a serial bit stream is input on the RxD pin, framing bits are stripped out of the data stream, and the resulting character is converted to parallel data format and passed to the system processor or DMA device.

Data Clocking

As discussed previously, the frequency of data transmission/reception on the data link is controlled by the MPSC clock in conjunction with the programmed clock divider (in register WR4). The 8274 is designed to permit all four serial interface lines (TxD and RxD for each channel) to operate at different data rates. Four clock input pins (TxC and RxC for each channel) are available for this function. Note that the clock rate divider spcified in WR4 is used for both RxC and TxC on the appropriate channel; clock rate dividers for each channel are independent.

Modem Control

The following four modem interface signals may be connected to the 8274:

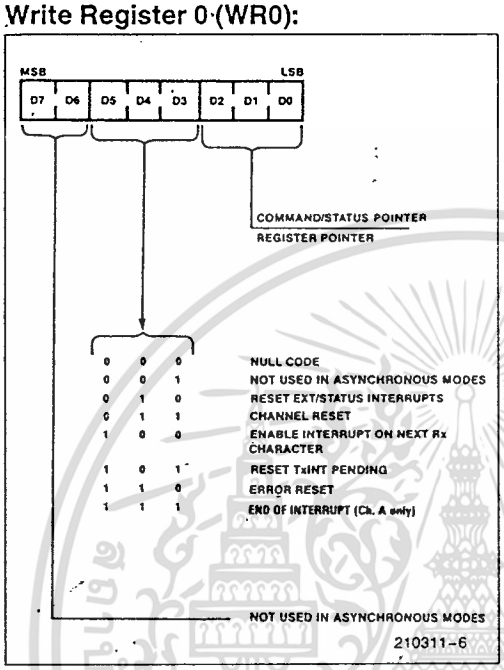
1. Data Terminal Ready (DTR). This interface signal (output by the 8274) is software controlled through bit 7 of WR5. When active, DTR indicates that the data terminal/computer equipment is active and ready to interact with the data communications channel. In addition, this signal prepares the modem for connection to the communication channel and maintains connections previously established (e.g., manual call origination).
2. Request To Send (RTS). This interface signal (output by the 8274) is software controlled through bit 1 of WR5. When active, RTS indicates that the data terminal/computer equipment is ready to transmit data. When the RTS bit is reset in asynchronous mode, the signal does not go high until the transmitter is empty.
3. Clear To Send (CTS). This interface signal (input to the 8274) is supplied by the modem in response to an active RTS signal. CTS indicates that the data terminal/computer equipment is permitted to transmit data. The state of CTS is available to the programmer as bit 5 of RR0. In addition, if the auto enable control is set (bit 5 of WR3), the 8274 will not transmit data bytes until CTS has been activated. If CTS becomes inactive during transmission of a character, the current character transmission is completed before the transmitter is disabled.
4. Carrier Detect (CD). This interface signal (input to the 8274) is supplied by the modem to indicate that a data carrier signal has been detected and that a valid data signal is present on the RxD line. The state of CD is available to the programmer as bit 3 of RR0. In addition, if the auto enable control is set (bit 5 of WR3), the 8274 will not enable the serial receiver until CD has been activated. If the CD signal becomes inactive during reception of a character, the receiver is disabled, and the partially received character is lost.

In addition to the above modem interface signals, the 8274 SYNDET input pin for channel A may be used as a general-purpose input in the asynchronous communication mode. The status of this signal is available to the programmer as bit 4 of status register RR0.

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

APPENDIX A
COMMAND/STATUS DETAILS FOR ASYNCHRONOUS COMMUNICATION

Write Register 0 (WR0):



D2,D1,D0 Command/Status Register Pointer bits determine which write-register the next byte is to be written into, or which read-register the next byte is to be read from. After reset, the first byte written into either channel goes into WR0. Following a read or write to any register (except WR0) the pointer will point to WR0.

D5,D4,D3 Command bits determine which of the basic seven commands are to be performed.

Command 0 Null—has no effect.

Command 1 Note used in asynchronous modes.

Command 2 Reset External/Status Interrupts—resets the latched status bits of RR0 and reenables them, allowing interrupts to occur again.

Command 3 Channel Reset—resets the Latched Status bits of RR0, the interrupt prioritization logic and all control registers for the channel. Four extra system clock cycles should be allowed for MPSC reset time before any additional commands or controls are written into the channel.

Command 4 Enable Interrupt on Next Receive Character—if the Interrupt-on-First-Receive Character mode is selected, this command reactivates that mode after each complete message is received to prepare the MPSC for the next message.

Command 5 Reset Transmitter Interrupt Pending—if the Transmit Interrupt mode is selected, the MPSC automatically interrupts data when the transmit buffer becomes empty. When there are no more characters to be sent, issuing this command prevents further transmitter interrupts until the next character has been completely sent.

Command 6 Error Reset—error latches, Parity and Overrun errors in RR1 are reset.

Command 7 End of Interrupt—resets the interrupt-in-service latch of the highest-priority internal device under service.

Write Register 1 (WR1):

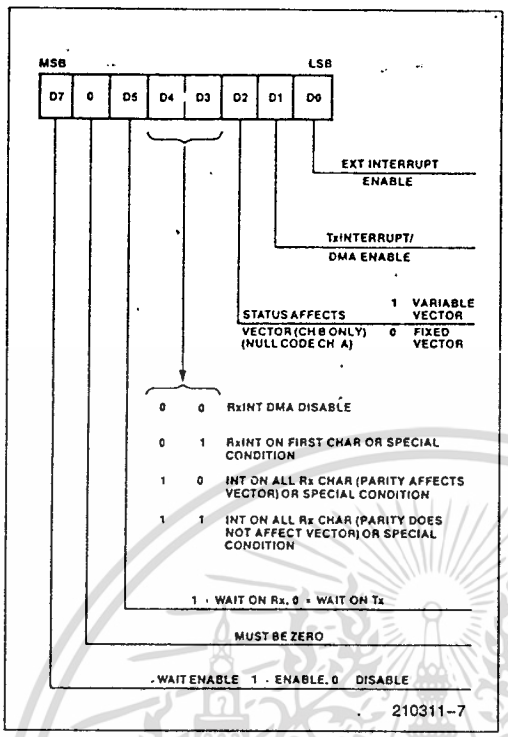
D0 External/Status Interrupt Enable—allows interrupt to occur as the result of transitions on the $\overline{CD}$, $\overline{CTS}$ or $\overline{SYNDET}$ inputs. Also allows interrupts as the result of a Break/Abort detection and termination, or at the beginning of CRC, or sync character transmission when the Transmit Underrun/EOM latch becomes set.

D1 Transmitter Interrupt/DMA Enable—allows the MPSC to interrupt or request a DMA transfer when the transmitter buffer becomes empty.

D2 Status Affects Vector—(WR1, D2 active in channel B only.) If this bit is not set, then the fixed vector, programmed in WR2, is returned from an interrupt acknowledge sequence. If the bit is set, then the vector returned from an interrupt acknowledge is variable as shown in the Interrupt Vector Table.

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

Write Register 1 (WR1):



D4,D3 Receive Interrupt Mode.

0 0 Receive Interrupts/DMA Disabled.

0 1 Receive Interrupt on First Character Only or Special Condition.

1 0 Interrupt on All Receive Characters of Special Condition (Parity Error is a Special Receive Condition).

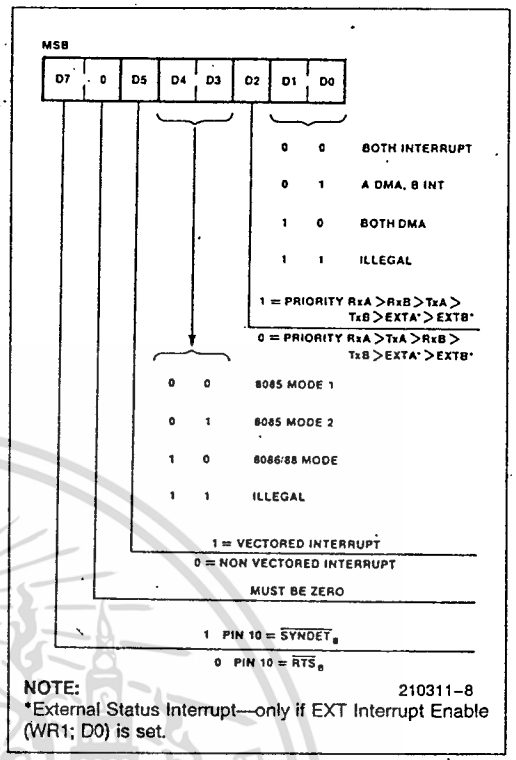
1 1 Interrupt on All Receive Characters or Special Condition (Parity Error is not a Special Receive Condition).

D5 Wait on Receive/$\overline{\text{Transmit}}$—when the following conditions are met, the RDY pin is activated, otherwise it is held in the High-Z state. (Conditions: Interrupt Enabled Mode, Wait Enabled, $\overline{CS} = 0$, A0 = 0/1, and A1 = 0). The RDY pin is pulled low when the transmitter buffer is full or the receiver buffer is empty and it is driven High when the transmitter buffer is empty or the receiver buffer is full. The RDY_A and RDY_B may be wired *or* connected since only one signal is active at any one time while the other is in the High Z state.

D6 Must be Zero.

D7 Wait Enable—enables the wait function.

Write Register 2 (WR2): Channel A



NOTE:
*External Status Interrupt—only if EXT Interrupt Enable (WR1; D0) is set.

D1,D0 System Configuration—These specify the data transfer from MPSC channels to the CPU, either interrupt or DMA based.

0 0 Channel A and Channel B both use interrupts.

0 1 Channel A uses DMA, Channel B uses interrupt.

1 0 Channel A and Channel B both use DMA.

1 1 Illegal Code.

D2 Priority—this bit specifies the relative priorities of the internal MPSC interrupt/DMA sources.

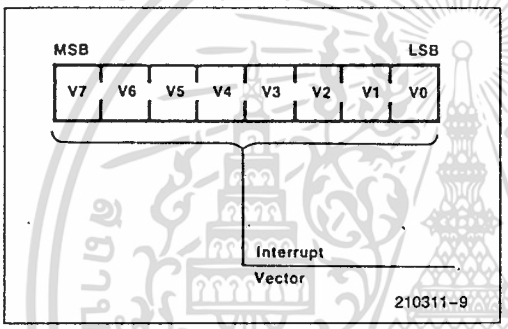
0 (Highest) RxA, TxA, RxA, RxB, TxBExTA, ExTB (Lowest).

1 (Highest) RxA, RxB, TxA, TxB ExTA, ExTB (Lowest).

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

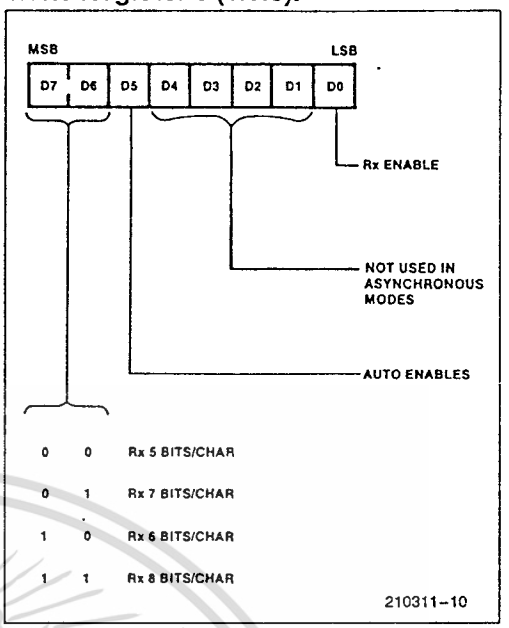
D5,D4,D3	Interrupt Code—specifies the behavior of the MPSC when it receives an interrupt acknowledge sequence from the CPU. (See Interrupt Vector Mode Table.)
0 X X	Non-vectored interrupts—intended for use with an external interrupt controller such as the 8259A.
1 0 0	8085 Vector Mode 1—intended for use as the primary MPSC in a daisy-chained priority structure.
1 0 1	8085 Vector Mode 2—intended for use as any secondary MPSC in a daisy-chained priority structure.
1 1 0	8086/88 Vector Mode—intended for use as either a primary or secondary in a daisy-chained priority structure.
D6	Must be Zero.
D7	
0	Pin 10 = $\overline{RTS}_B$.
1	Pin 10 = $\overline{SYNDET}_B$.

Write Register 2 (WR2): Channel B



D7–D0 Interrupt vector—this register contains the value of the interrupt vector placed on the data bus during acknowledge sequences.

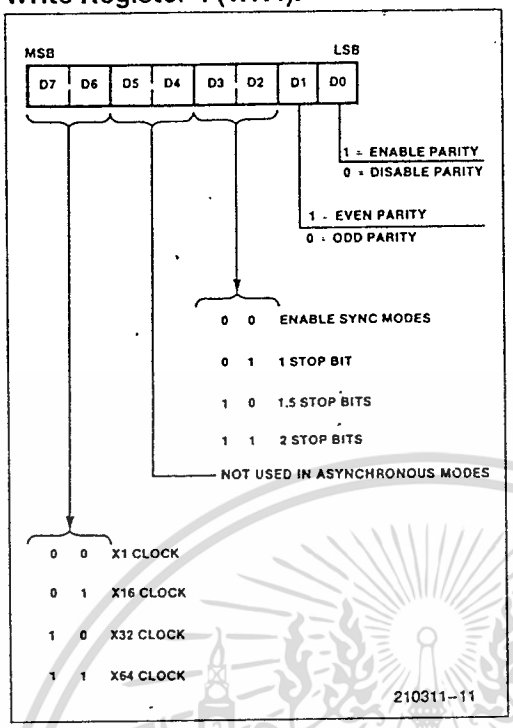
Write Register 3 (WR3):



D0	Receiver Enable—a one enables the receiver to begin. This bit should be set only after the receiver has been initialized.
D5	Auto Enables—a one written to this bit causes $\overline{CD}$ to be an automatic enable signal for the receiver and $\overline{CTS}$ to be an automatic enable signal for the transmitter: A zero written to this bit limits the effect of $\overline{CD}$ and $\overline{CTS}$ signals to setting/resetting their corresponding bits in the status register (RR0).
D7,D6	Receiver Character length.
0 0	Receive 5 Data bits/character.
0 1	Receive 7 Data bits/character.
1 0	Receive 6 Data bits/character.
1 1	Receive 8 Data bits/character.

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

Write Register 4 (WR4):



D0 Parity—a one in this bit causes a parity bit to be added to the programmed number of data bits per character for both the transmitted and received character. If the MPSC is programmed to receive 8 bits per character, the parity bit is not transferred to the microprocessor. With other receiver character lengths, the parity bit is transferred to the microprocessor.

D1 Even/Odd Parity—if parity is enabled, a one in this bit causes the MPSC to transmit and expect even parity, and zero causes it to send and expect odd parity.

D3,D2 Stop Bits.

0 0 Selects synchronous modes.

0 1 Async mode, 1 stop bit/character.

1 0 Async mode, 1½ stop bits/character.

1 1 Async mode, 2 stop bits/character.

D7,D6 Clock mode—selects the clock/data rate multiplier for both the receiver and the transmitter. If the 1x mode is selected, bit synchronization must be done externally.

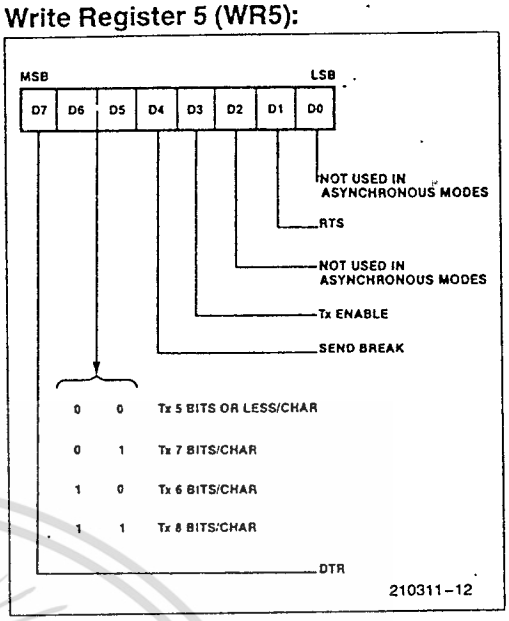
0 0 Clock rate = Data rate × 1.

0 1 Clock rate = Data rate × 16.

1 0 Clock rate = Data rate × 32.

1 1 Clock rate = Data rate × 64.

Write Register 5 (WR5):



D1 Request to Send—a one in this bit forces the $\overline{RTS}$ pin active (low) and zero in this bit forces the $\overline{RTS}$ pin inactive (high). When the RTS bit is reset in asynchronous mode, the signal does not go inactive until the transmitter is empty.

D3 Transmitter Enable—a zero in this bit forces a marking state on the transmitter output. If this bit is set to zero during data or sync character transmission, the marking state is entered after the character has been sent. If this bit is set to zero during transmission of a CRC character, sync or flag bits are substituted for the remainder of the CRC bits.

D4 Send Break—a one in this bit forces the transmit data low. A zero in this bit allows normal transmitter operation.

D6,D5 Transmit Character length.

0 0 Transmit 5 or less bits/character.

0 1 Transmit 7 bits/character.

1 0 Transmit 6 bits/character.

1 1 Transmit 8 bits/character.

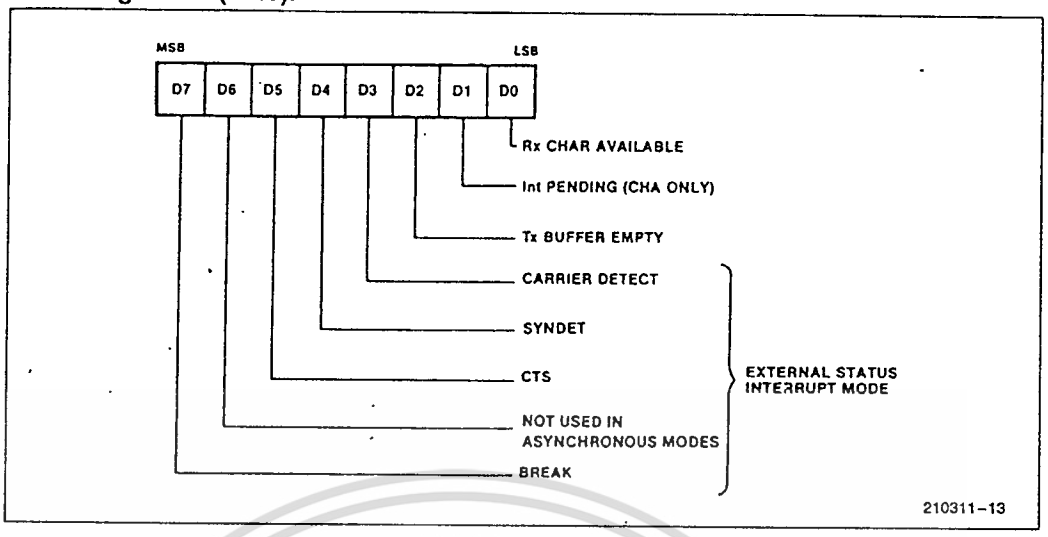
Bits to be sent must be right justified, least-significant bit first, e.g.:

D7 D6 D5 D4 D3 D2 D1 D0

0 0 B5 B4 B3 B2 B1 B0

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

Read Register 0 (RR0):



D0 Receive Character Available—this bit is set when the receive FIFO contains data and is reset when the FIFO is empty.

D1 Interrupt Pending—This Interrupt-Pending bit is reset when an E01 command is issued and there is no other interrupt request pending at that time. In vector mode, this bit is set at the falling edge of the second INTA in an INTA cycle for an internal interrupt request. In non-vector mode, this bit is set at the falling edge of RD input after pointer 2 is specified. This bit is always zero in Channel B.

D2 Transmit Buffer Empty—This bit is set whenever the transmit buffer is empty except when CRC characters are being sent in a synchronous mode. This bit is reset when the transmit buffer is loaded. This bit is set after an MPSC reset.

D3 Carrier Detect—This bit contains the state of the $\overline{CD}$ pin at the time of the last change of any of the External/Status bits (CD, CTS, Sync/Hunt, Break/Abort, or Tx Underrun/EOM). Any change of state of the $\overline{CD}$ pin causes the CD bit to be latched and causes an External/Status interrupt. This bit indicates current state of the $\overline{CD}$ pin immediately following a Reset External/Status Interrupt command.

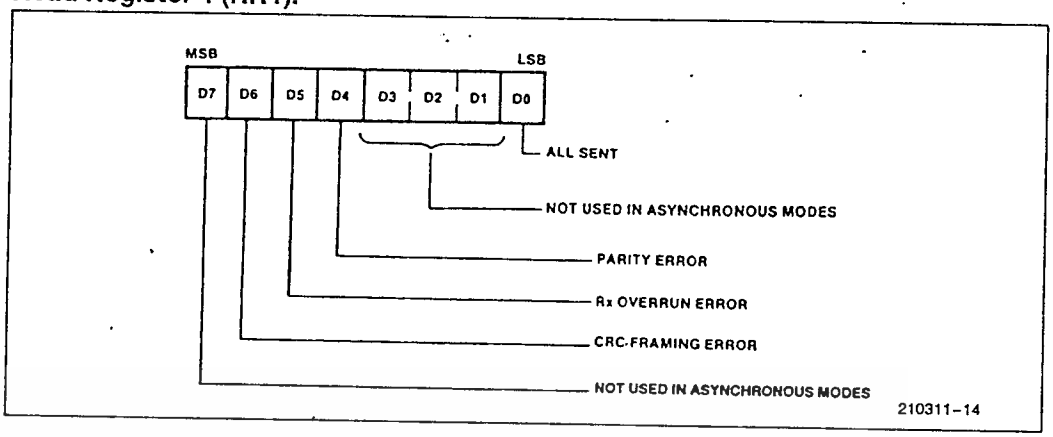
D4 SYNDET—In asynchronous modes, the operation of this bit is similar to the CD status bit, except that it shows the state of the $\overline{SYNDET}$ input. Any High-to-Low transition on the $\overline{SYNDET}$ pin sets this bit, and causes an External/Status interrupt (if enabled). The Reset External/Status Interrupt command is issued to clear the interrupt. A Low-to-High transition clears this bit and sets the External/Status interrupt. When the External/Status interrupt is set by the change in state of any other input or condition, this bit shows the inverted state of the $\overline{SYNDET}$ pin at time of the change. This bit must be read immediately following a Reset External/Status Interrupt command to read the current state of the $\overline{SYNDET}$ input.

D5 Clear to Send—this bit contains the inverted state of the $\overline{CTS}$ pin at the time of the last change of any of the External/Status bits (CD, CTS, Sync/Hunt, Break/Abort, or Tx Underrun/EOM). Any change of state of the $\overline{CTS}$ pin causes the CTS bit to be latched and causes an External/Status interrupt. This bit indicates the inverse of the current state of the $\overline{CTS}$ pin immediately following a Reset External/Status Interrupt command.

D7 Break—in the Asynchronous Receive mode, this bit is set when a Break sequence (null character plus framing error) is detected in the data stream. The External/Status interrupt, if enabled, is set when break is detected. The interrupt service routine must issue the Reset External/Status Interrupt command (WR0, Command 2) to the break detection logic so the Break sequence termination can be recognized.

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

Read Register 1 (RR1):



The Break bit is reset when the termination of the Break sequence is detected in the incoming data stream. The termination of the Break sequence also causes the External/Status interrupt to be set. The Reset External/Status Interrupt command must be issued to enable the break detection logic to look for the next Break sequence. A single, extraneous null character is present in the receiver after the termination of a break; it should be read and discarded.

D0 All sent—this bit is set when all characters have been sent. It is reset when characters are in the transmitter. In synchronous modes, this bit is always set.

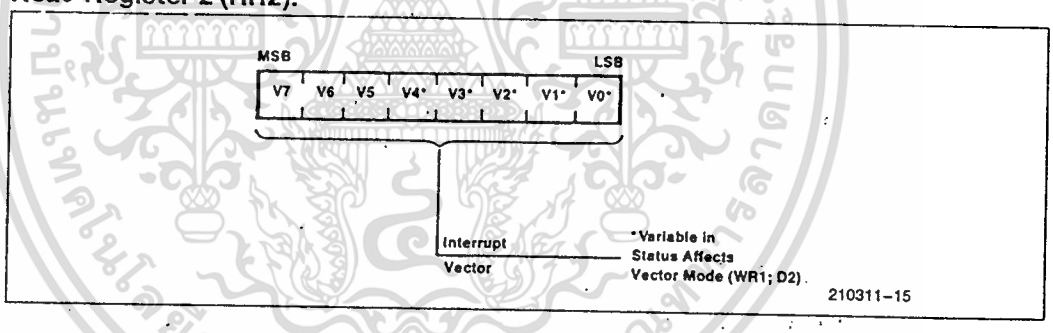
D4 Parity Error—if parity is enabled, this bit is set for received characters whose parity does not match the programmed sense (Even/Odd). This bit is latched. Once an error occurs, it remains set until the Error Reset command is written.

D5 Receive Overrun Error—this bit indicates that the receive FIFO has been overloaded by the receiver. The last character in the FIFO is overwrittenand flagged with this error. Once the overwritten character is read, this error condition is latched until reset by the Error Reset command. If the MPSC is in the "status affects vector" mode, the overrun causes a Special Receive Error Vector.

D6 Framing Error—in async modes, a one in this bit indicates a receive framing error. It can be reset by issuing an Error Reset command.

RR2 Channel B

Read Register 2 (RR2):



D7–D0 Interrupt vector—contains the interrupt vector programmed into WR2. If the "status affects vector" mode is selected, it contains the modified vector. (See WR2.) RR2 contains the modified vector for the highest priority interrupt pending. If no interrupts are pending, the variable bits in the vector are set to one. May be read from Channel B only.

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

APPENDIX B
MPSC-POLLED TRANSMIT/RECEIVE CHARACTER ROUTINES

```
MPSC$RX$INIT: procedure (cmd$port,
                         clock$rate,stop$bits,parity$type,parity$enable,
                         rx$char$length,rx$enable,auto$enable,
                         tx$char$length,tx$enable,dtr,brk,rts);

  declare cmd$port       byte,
          clock$rate     byte,
          stop$bits      byte,
          parity$type    byte,
          parity$enable  byte,
          rx$char$length byte,
          rx$enable      byte,
          auto$enable    byte,
          tx$char$length byte,
          tx$enable      byte,
          dtr            byte,
          brk            byte,
          rts            byte;


  output(cmd$port)=30H;              /* channel reset */

  output(cmd$port)=14H;              /* point to WR4 */
  /* set clock rate, stop bits, and parity information */
  output(cmd$port)=shl(clock$rate,6) or shl(stop$bits,2) or shl(parity$type,1)
                   or parity$enable;

  output(cmd$port)=13H;              /* point to WR3 */
  /* set up receiver parameters */
  output(cmd$port)=shl(rx$char$length,6) or rx$enable or shl(auto$enable,5);

  output(cmd$port)=15H;              /* point to WR5 */
  /* set up transmitter parameters */
  output(cmd$port)=shl(tx$char$length,5) or shl(tx$enable,3) or shl(dtr,7)
                   or shl(brk,4) or shl(rts,1);

  end MPSC$RX$INIT;
```

210311-16

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```
MPSC$POLL$RCV$CHARACTER: procedure(data$port,cmd$port,character$ptr) byte;

  declare data$port     byte,
          cmd$port      byte,
          character$ptr pointer,
          character     based character$ptr byte,
          status        byte;

  declare char$avail    literally '1',
          rcv$error     literally '70H';

  /* wait for input character ready */
  while (input(cmd$port) and char$avail) <> 0 do; end;

  /* check for errors in received character */
  output(cmd$port)=1;                          /* point to RR1 */
  if (status:=input(cmd$port) and rcv$error)
    then do;
      character=input(data$port);              /* read character to clear MPSC */
      call RECEIVE$ERROR(cmd$port,status);     /* clear receiver errors */
      return 0;                                /* error return - no character avail */
      end;
    else do;
      character=input(data$port);
      return 0FFH;                              /* good return - character avail */
      end;

end MPSC$POLL$RCV$CHARACTER;


MPSC$POLL$TRAN$CHARACTER: procedure(data$port,cmd$port,character);

  declare data$port      byte,
          cmd$port       byte,
          character      byte;

  declare tx$buffer$empty literally '4';

  /* wait for transmitter buffer empty */
  while not (input(cmd$port) and tx$buffer$empty) do; end;

  /* output character */
  output(data$port)=character;

end MPSC$POLL$TRAN$CHARACTER;


RECEIVE$ERROR: procedure(cmd$port,status);

  declare cmd$port      byte,
          status        byte;

  output(cmd$port)=30H;                /* error reset */

  /* *** other application dependent
         error processing should be placed here   *** */

end RECEIVE$ERROR;
```

210311-17

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```
TRANSMIT$BUFFER: procedure(buf$ptr,buf$length)

  declare
    buf$ptr       pointer,
    buf$length    byte;

  /* set up transmit buffer pointer and buffer length in global variables for
     interrupt service */
  tx$buffer$ptr=buf$ptr;
  transmit$length=buf$length;

  transmit$status=not$complete;              /* setup status for not complete */
  output(data$port)=transmit$buffer(0);      /* transmit first character */
  transmit$index=1;                          /* first character transmitted */

  /* wait until transmission complete or error detected */
  while transmit$status = not$complete do; end;
  if transmit$status <> complete
    then return false;
    else return true;

end TRANSMIT$BUFFER;


RECEIVE$BUFFER: procedure (buf$ptr,buf$length$ptr);

  declare
    buf$ptr         pointer,
    buf$length$ptr  pointer,
    buf$length      based buf$length$ptr byte;

  /* set up receive buffer pointer in global variable for interrupt service */
  rx$buffer$ptr=buf$ptr;
  receive$index=0;

  receive$status=not$complete;           /* set status to not complete */
  /* wait until buffer received */
  while receive$status = not$complete do; end;
  buf$length=receive$length;
  if receive$status = complete
    then return true;
    else return false;

end RECEIVE$BUFFER;
```

210311-18

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

APPENDIX C
INTERRUPT-DRIVEN TRANSMIT/RECEIVE SOFTWARE

```
declare
  /* global variables for buffer manipulation */

  rx$buffer$ptr          pointer,           /* pointer to receive buffer */
  receive$buffer based rx$buffer$ptr(128) byte,
  receive$status         byte initial(0),   /* indicates receive buffer status */
  receive$index          byte,              /* current index into receive buffer */
  receive$length         byte,              /* length of final receive buffer */

  tx$buffer$ptr          pointer,           /* pointer to transmit buffer */
  transmit$buffer based tx$buffer$ptr(128) byte,
  transmit$status        byte initial(0),   /* indicates transmit buffer status */
  transmit$index         byte,              /* current index into transmit buffer */
  transmit$length        byte,              /* length of buffer to be transmitted */

  cmd$port               literally '43H',
  data$port              literally '41H',
  a$cmd$port             literally '42H',
  b$cmd$port             literally '43H',
  line$feed              literally '0AH',
  not$complete           literally '0',
  complete               literally '0FFH',
  overrun                literally '1',

  channel$reset          literally '18H',
  error$reset            literally '30H',
  reset$ext$status       literally '10H';
```

210311-20

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```
MPSC$INT$INIT: procedure (clock$rate,stop$bits,parity$type,parity$enable,
                          rx$char$length,rx$enable,auto$enable,
                          tx$char$length,tx$enable,dtr,brk,rts,
                          ext$en,tx$en,rx$en,stat$affects$vector,
                          config,priority,vector$int$mode,int$vector);

  declare
        clock$rate        byte,         /* 2-bit code for clock rate divisor */
        stop$bits         byte,         /* 2-bit code for number of stop bits */
        parity$type       byte,         /* 1-bit parity type */
        parity$enable     byte,         /* 1-bit parity enable */
        rx$char$length    byte,         /* 2-bit receive character length */
        rx$enable         byte,         /* 1-bit receiver enable */
        auto$enable       byte,         /* 1-bit auto enable flag */
        tx$char$length    byte,         /* 2-bit transmit character length */
        tx$enable         byte,         /* 1-bit transmitter enable */
        dtr               byte,         /* 1-bit status of DTR pin */
        brk               byte,         /* 1-bit data link break enable */
        rts               byte,         /* 1-bit status of RTS pin */
        ext$en            byte,         /* 1-bit external/status enable */
        tx$en             byte,         /* 1-bit Tx interrupt enable */
        rx$en             byte,         /* 2-bit Rx interrupt enable/mode */
        stat$aff$vector   byte,         /* 1-bit status affects vector flag */
        config            byte,         /* 2-bit system config - int/DMA */
        priority          byte,         /* 1-bit priority flag */
        vector$int$mode   byte,         /* 3-bit interrupt mode code */
        int$vector        byte;         /* 8-bit interrupt type code */


  output(b$cmd$port)=channel$reset;   /* channel reset */

  output(b$cmd$port)=14H;             /* point to WR4 */
  /* set clock rate, stop bits, and parity information */
  output(b$cmd$port)=shl(clock$rate,6) or shl(stop$bits,2) or shl(parity$type,1)
                     or parity$enable;

  output(b$cmd$port)=13H;             /* point to WR3 */
  /* set up receiver parameters */
  output(b$cmd$port)=shl(rx$char$length,6) or rx$enable or shl(auto$enable,5);

  output(b$cmd$port)=15H;           /* point to WR5 */
  /* set up transmitter parameters */
  output(b$cmd$port)=shl(tx$char$length,5) or shl(tx$enable,3) or shl(dtr,7)
                  or shl(brk,4) or shl(rts,1);

  output(b$cmd$port)=12H;             /* point to WR2 */
  /* set up interrupt vector */
  output(b$cmd$port)=int$vector;

  output(a$cmd$port)=12H;             /* point to WR2, channel A */
  /* set up interrupt modes */
  output(a$cmd$port)=shl(vector$int$mode,3) or shl(priority,2) or config;

  output(b$cmd$port)=11H;             /* point to WR1 */
  /* set up interrupt enables */
  output(b$cmd$port)=shl(rx$en,3) or shl(stat$aff$vector,2) or shl(tx$en,1)
                     or ext$en;

end MPSC$INT$INIT;
```

210311-21

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```
MPSC$RECEIVE$CHARACTER$INT: procedure interrupt 22H;

  /* ignore input if no open buffer */
  if receive$status <> not$complete then return;

  /* check for receive buffer overrun */
  if receive$index = 128
    then receive$status=overrun;
    else do;
      /* read character from MPSC and place in buffer - note that the
         parity of the character must be masked off during this step if
         the character is less than 8 bits (e.g., ASCII)  */
      receive$buffer(receive$index),character=input(data$port) and 7FH;
      receive$index=receive$index+1;      /* update receive buffer index */

      /* check for line feed to end line */
      if character = line$feed
        then do; receive$length=receive$index; receive$status=complete; end;
    end;

end MPSC$RECEIVE$CHARACTER$INT;


MPSC$TRANSMIT$CHARACTER$INT: procedure interrupt 20H;

  /* check for more characters to transfer */
  if transmit$index < transmit$length
    then do;
      /* write next character from buffer to MPSC */
      output(data$port)=transmit$buffer(transmit$index);
      transmit$index=transmit$index+1;     /* update transmit buffer index */
    end;
    else transmit$status=complete;

end MPSC$TRANSMIT$CHARACTER$INT;


RECEIVE$ERROR$INT: procedure interrupt 23H;

  declare
    temp                byte;    /* temporary character storage */

  output(cmd$port)=1;            /* point to RR1 */
  receive$status=input(cmd$port);
  temp=input(data$port);         /* discard character */
  output(cmd$port)=error$reset;  /* send error reset */

  /* *** other application dependent
         error processing should be placed here   *** */

end RECEIVE$ERROR$INT;


EXTERNAL$STATUS$CHANGE$INT: procedure interrupt 21H;

  transmit$status=input(cmd$port)      /* input status change information */
  output(cmd$port)=reset$ext$status;

  /* *** other application dependent
         error processing should be placed here   *** */

end EXTERNAL$STATUS$CHANGE$INT;
```

210311-19

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

APPENDIX D
APPLICATION EXAMPLE USING SDK-86

This application example shows the 8274 in a simple iAPX-86/88 system. The 8274 controls two separate asynchronous channels using its internal interrupt controller to request all data transfers. The 8274 driver software is described which transmits and receives data buffers provided by the CPU. Also, status registers are maintained in system memory to allow the CPU to monitor progress of the buffers and error conditions.

THE HARDWARE INTERFACE

Nothing could be easier than the hardware design of an interrupt-driven 8274 system. Simply connect the data bus lines, a few bus control lines, supply a timing clock for baud rate and, voila, it's done! For this example, the ubiquitous SDK-86 is used as the host CPU system. The 8274 interface is constructed on the wire-wrap area provided. While discussing the hardware interface, please refer to Diagram 1.

Placing the 8274 on the lower 8 bits of the 8086 data bus allows byte-wide data transfers at even I/O addresses. For simplicity, the 8274's $\overline{CS}$ input is generated by combining the M/IO select line with address line A7 via a 7432. This places the 8274 address range in multiple spots within the 8086 I/O address space. (While fine for this example, a more complete address decoding is recommended for actual prototype systems.) The 8086's A1 and A2 address lines are connected to the A0 and A1 8274 register select inputs respectively. Although other port assignments are possible because of the overlapping address spaces, the following I/O port assignments are used in this example:

Port Function	I/O Address
Data channel A	0000H
Command/status A	0002H
Data channel B	0004H
Command/status B	0006H

To connect the 8274's interrupt controller into the system an inverter and pull-up resistor are needed to convert the 8274's active-low, interrupt-request output, $\overline{INT}$, into the correct polarity for the 8086's INTR interrupt input. The 8274 recognizes interrupt-acknowledge bus cycles by connecting the $\overline{INTA}$ (INTerrupt Acknowledge) lines of the 8274 and 8086 together.

The 8274 ReaD and WRite lines directly connect to the respective 8086 lines. The RESET line requires an inverter. The system clock for the 8274 is provided by the PCLK (peripheral clock) output of the 8284A clock generator.

On the 8274's serial side, traditional 1488 and 1489 RS-232 drivers and receivers are used for the serial interface. The onboard baud rate generator supplies the channel baud rate timing. In this example, both sides of both channels operate at the same baud rate although this certainly is not a requirement. (On the SDK-86, the baud rate selection is hard-wired thru jumpers. A more flexible approach would be to incorporate an 8253 Programmable Interval Timer to allow software-configurable baud rate selection.)

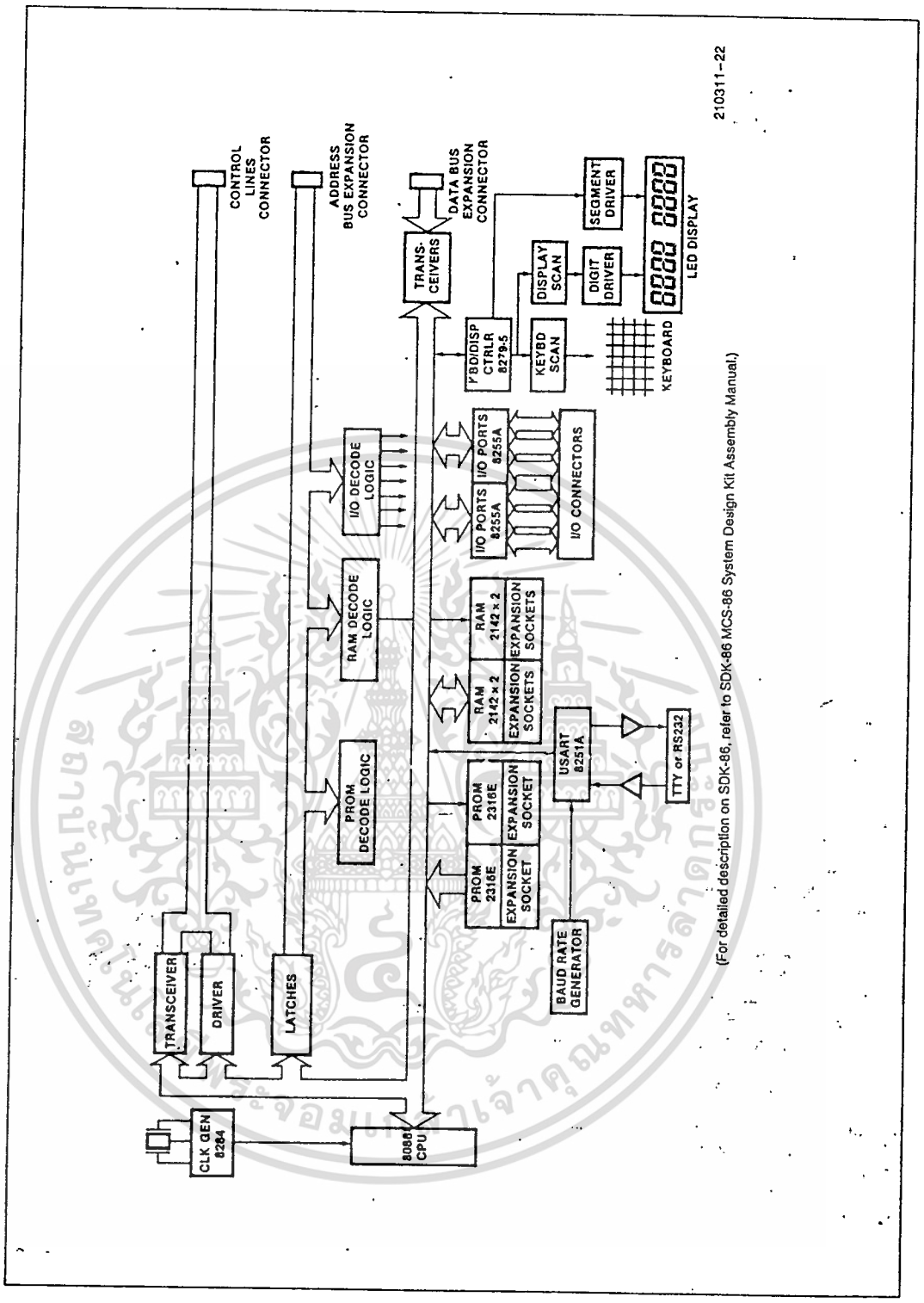
That's all there is to it. This hardware interface is completely general-purpose and supports all of the 8274 features except the DMA data transfer mode which requires an external DMA controller. Now let's look at the software interface.

SOFTWARE INTERFACE

In this example, it is assumed that the 8086 has better things to do rather than continuously run a serial channel. Presenting the software as a group of callable procedures lets the designer include them in the main body of another program. The interrupt-driven data transfers give the effect that the serial channels are handled in the background while the main program is executing in the foreground. There are five basic procedures: a serial channel initialization routine and buffer handling routines for the transmit and receive data buffers of each channel. Appendix D-1 shows the entire software listing. Listing line numbers are referenced as each major routing is discussed.

The channel initialization routine (INITIAL 8274), starting with line #203, simply sets each channel into a particular operating mode by loading the command registers of the 8274. In normal operation, once these registers are loaded, they are rarely changed. (Although this example assumes a simple asynchronous operating mode, the concept is easily extended for the byte- and bit-synchronous modes.)

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้



(For detailed description on SDK-86, refer to SDK-86 MCS-86 System Design Kit Assembly Manual.)

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

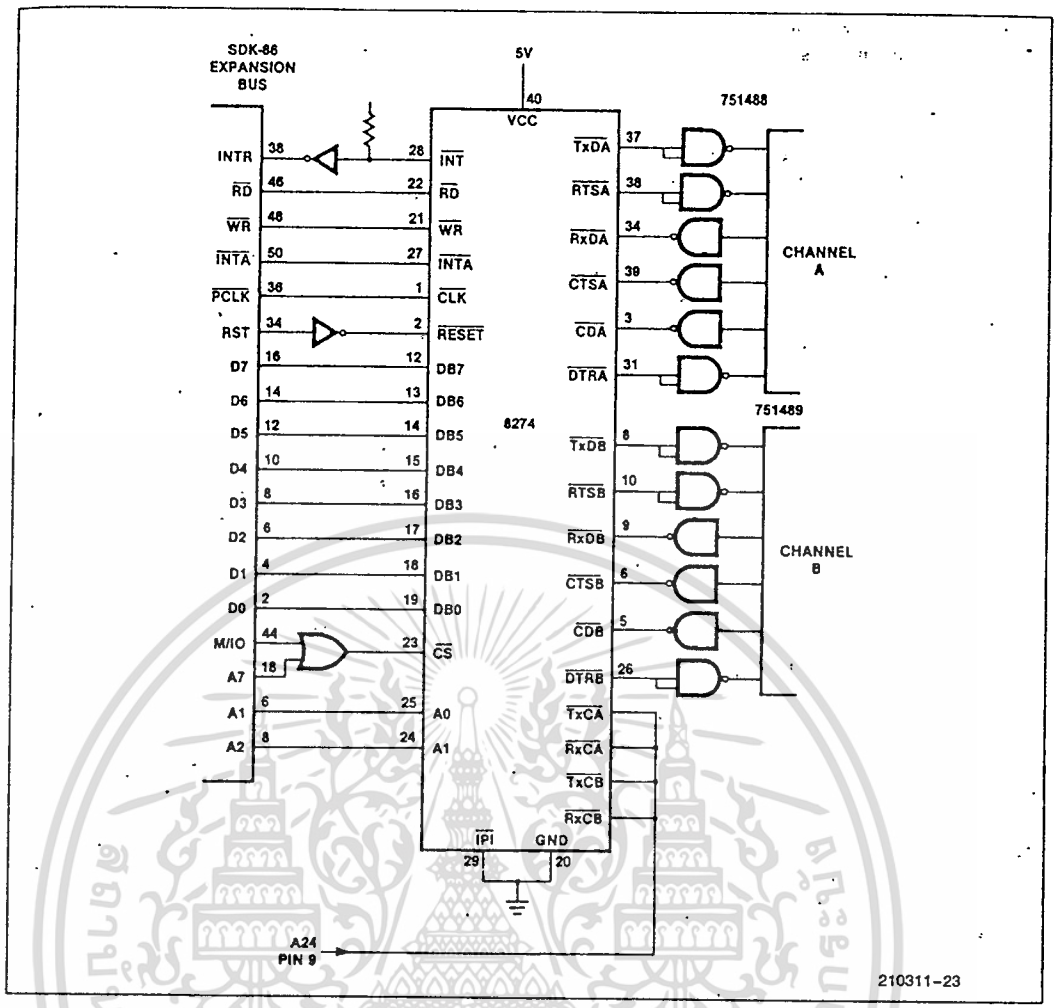


Figure D-1. 8274/SDK-86 Hardware Interface

The channel operating modes are contained in two tables starting with line #163. As the 8274 has only one command register per channel, the remaining seven registers are loaded indirectly through the WR0 (Write Register 0) register. The first byte of each table entry is the register pointer value which is loaded into WR0 and the second byte is the value for that particular register.

The indicated modes set the 8274 for asynchronous operation with data characters 8 bits long, no parity, and 2 stop bits. An X16 baud rate clock is assumed. Also selected is the "interrupt on all RX character" mode with a variable interrupt vector compatible with the 8086/8088. The transmitters are enabled and all model control lines are put in their active state.

In addition to initializing the 8274, this routine also sets up the appropriate interrupt vectors. The 8086 assumes the first 1K bytes of memory contain up to 256 separate interrupt vectors. On the SDK-86 the initial 2K bytes of memory is RAM and therefore must be initialized with the appropriate vectors. (In a prototype system, this initial memory is probably ROM, thus the vector set-up is not needed.) The 8274 supplies up to eight different interrupt vectors. These vectors are developed from internal conditions such as data requests, status changes, or error conditions for each channel. The initialization routine arbitrarily assumes that the initial 8274 vector corresponds to 8086 vector location 80H (memory location 200H). This choice is arbitrary since the 8274 initial vector location is programmable.

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

Finally, the initialization routine sets up the status and flag in RAM. The meaning and use of these locations are discussed later.

Following the initialization routine are those for the transmit commands (starting with line #268). These commands assume that the host CPU has initialized the publicly declared variables for the transmit buffer pointer, TX__POINTER__CHx, and the buffer length, TX__LENGTH__CHx. The transmit command routines simply clear the transmitter empty flag, TX EMPTY CHx, and load the first character of the buffer into the transmitter. It is necessary to load the first character in this manner since transmitter interrupts are generated only when the 8274's transmit data buffer becomes empty. It is the act of becoming empty which generates the interrupt not simply the buffer being empty, thus the transmitter needs one character to start.

The host CPU can monitor the transmitter empty flag, TX__EMPTY__CHx, in order to determine when transmission of the buffer is complete. Obviously, the CPU should only call the command routine after first checking that the empty flag is set.

After returning to the main program, all transmitter data transfers are handled via the transmitter-interrupt service routines starting at lines #360 and #443. These routines start by issuing an End-Of-Interrupt command to the 8274. (This command resets the internal-interrupt controller logic of the 8274 for this particular vector and opens the logic for other internal interrupt requests. The routines next check the length count. If the buffer is completely transmitted, the transmitter empty flag, TX__EMPTY__CHx, is set and a command is issued to the 8274 to reset its interrupt line. Assuming that the buffer is not completely transmitted, the next character is output to the transmitter. In either case, an interrupt return is executed to return to the main CPU program.

The receiver commands start at line #314. Like the transmit commands, it is assumed that the CPU has initialized the receive-buffer-pointer public variable, RX__POINTER__CHx. This variable points to the first location in an empty receive buffer. The command routines clear the receiver ready flag, RX__READY__CHx, and then set the receiver enable bit in the 8274 WR3 register. With the receiver now enabled, any received characters are placed in the receive buffer using interrupt-driven data transfers.

The received data service routines, starting at lines #402 and #485, simply place the received character in the buffer after first issuing the EOI command. The character is then compared to an ASCII CR. An ASCII CR causes the routine to set the receiver ready flag, RX__READY__CHx, and to disable the receiver. The CPU can interrogate this flag to determine when the buffer contains a new line of data. The receive buffer pointer, RX__POINTER__CHx, points to the last received character and the receive counter, RX__COUNTER__CHx, contains the length.

That completes our discussion of the command routines and their associated interrupt service routines. Although not used by the commands, two additional service routines are included for completeness. These routines handle the error and status-change interrupt vectors.

The error service routines, starting at lines #427 and #510, are vectored to if a special receive condition is detected by the 8274. These special receive conditions include parity, receiver overrun, and framing errors. When this vector is generated, the error condition is indicated in RR1 (Read Register 1). The error service routine issues an EOI command, reads RR1 and places it in the ERROR__MSG__CHx variable, and then issues a reset error command to the 8274. The CPU can monitor the error message location to detect error conditions. The designer, of course, can supply his own error service routine.

Similarly, the status-change routines (starting lines #386 and #469) are initiated by a change in the modem-control status lines CTS/, CD/, or SYNDET/. (Note that WR2 bit 0 controls whether the 8274 generates interrupts based upon changes in these lines. Our WR2 parameter is such that the 8274 is programmed to ignore changes for these inputs.) The service routines simply read RR0, place its contents in the STATUS__MSG__CHx variable and then issue a reset external status command. Read Register 0 contains the state of the modem inputs at the point of the last change.

Well, that's it. This application example has presented useful, albeit very simple, routines showing how the 8274 might be used to transmit and receive buffers using an asynchronous serial format. Extensions for byte- or bit-synchronous formats would require no hardware changes due to the highly programmable nature of the 8274's serial formats.

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```
MCS-86 MACRO ASSEMBLER   ASYNCB

LOC  OBJ              LINE    SOURCE

                        31
                        32          NAME    ASYNCB  ;MODULE NAME
                        33
                        34    ;PUBLIC DECLARATIONS FOR COMMAND ROUTINES
                        35
                        36          PUBLIC  INITIAL_8274    ;INITIALIZATION ROUTINE
                        37          PUBLIC  TX_COMMAND_CHB  ;TX BUFFER COMMAND CHANNEL B
                        38          PUBLIC  TX_COMMAND_CHA  ;TX BUFFER COMMAND CHANNEL A
                        39          PUBLIC  RX_COMMAND_CHB  ;RX BUFFER COMMAND CHANNEL B
                        40          PUBLIC  RX_COMMAND_CHA  ;RX BUFFER COMMAND CHANNEL A
                        41
                        42    ;PUBLIC DECLARATIONS FOR STATUS VARIABLES
                        43
                        44          PUBLIC  RX_READY_CHB    ;RX READY FLAG CHB
                        45          PUBLIC  RX_READY_CHA    ;RX READY FLAG CHA
                        46          PUBLIC  TX_EMPTY_CHB    ;TX EMPTY FLAG CHB
                        47          PUBLIC  TX_EMPTY_CHA    ;TX EMPTY FLAG CHA
                        48          PUBLIC  RX_COUNT_CHB    ;RX BUFFER COUNTER CHB
                        49          PUBLIC  RX_COUNT_CHA    ;RX BUFFER COUNTER CHA
                        50          PUBLIC  ERROR_MSG_CHB   ;ERROR FLAG CHB
                        51          PUBLIC  EPROR_MSG_CHA   ;ERROR FLAG CHA
                        52          PUBLIC  STATUS_MSG_CHB  ;STATUS FLAG CHB
                        53          PUBLIC  STATUS_MSG_CHA  ;STATUS FLAG CHA
                        54
                        55    ;PUBLIC DECLARATIONS FOR VARIABLES PASSED TO THE TRANSMIT
                        56    ;AND RECEIVE COMMANDS.
                        57
                        58          PUBLIC  TX_POINTER_CHB  ;TX BUFFER POINTER FOR CHB
                        59          PUBLIC  TX_LENGTH_CHB   ;TX LENGTH OF BUFFER FOR CHB
                        60          PUBLIC  TX_POINTER_CHA  ;TX BUFFER POINTER FOR CHA
                        61          PUBLIC  TX_LENGTH_CHA   ;TX LENGTH OF BUFFER FOR CHA
                        62          PUBLIC  RX_POINTER_CHB  ;RX BUFFER POINTER FOR CHB
                        63          PUBLIC  RX_POINTER_CHA  ;RX BUFFER POINTER FOR CHA
                        64
                        65    ;I/O PORT ASSIGNMENTS
                        66
                        67    ;CHANNEL A PORT ASSIGNMENTS
                        68
0000                    69    DATA_PORT_CHA           EQU     0                       ;DATA I/O PORT
0002                    70    COMMAND_PORT_CHA        EQU     2                       ;COMMAND PORT
0002                    71    STATUS_PORT_CHA         EQU     COMMAND_PORT_CHA        ;STATUS PORT
                        72
                        73    ;CHANNEL B PORT ASSIGNMENTS
                        74
0004                    75    DATA_PORT_CHB           EQU     4                       ;DATA I/O PORT
0006                    76    COMMAND_PORT_CHB        EQU     6                       ;COMMAND PORT
0006                    77    STATUS_PORT_CHB         EQU     COMMAND_PORT_CHB        ;STATUS PORT
                        78
                        79    ;MISC  SYSTEM EQUATES
                        80
0080                    81    CR_CHR          EQU     80H     ;ASCII CR CHARACTER CODE
0200                    82    INT_TABLE_BASE  EQU     200H    ;INT  VECTOR BASE ADDRESS
0500                    83    CODE_START      EQU     500H    ;START LOCATION FOR CODE
                        84
                        85 +1 $EJECT
                        86
                        87    ;RAM ASSSIGNMENTS FOR DATA SEGMENT
                        88
---                     89          DATA    SEGMENT
                        90
```

210311-25

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```
MCS-96 MACRO ASSEMBLER   ASYNCB


LOC  OBJ              LINE    SOURCE

                        91    ;VECTOR INTERRUPT TABLE - ASSUME INITIAL 8274 INTERRUPT
                        92    ;VECTOR IS NUMBER 00 (0200H)  FOR EACH VECTOR, THE TABLE
                        93    ;CONTAINS START LOCATION AND CODE SEGMENT REGISTER VALUE
                        94    ;THE TABLE IS LOADED FROM PROM
                        95
0200                    96            ORG     INT_TABLE_BASE
                        97
0200 0000               98    TX_VECTOR_CHB   DW      0       ;TX INTERRUPT VECTOR FOR CHB
0202 0000               99    TX_CS_CHB       DW      0
                       100
0204 0000              101    STS_VECTOR_CHB  DW      0       ;STATUS INTERRUPT VECTOR FOR CHB
0206 0000              102    STS_CS_CHB      DW      0
                       103
0208 0000              104    RX_VECTOR_CHB   DW      0       ;RX INTERRUPT VECTOR FOR CHB
020A 0000              105    RX_CS_CHB       DW      0
                       106
020C 0000              107    ERR_VECTOR_CHB  DW      0       ;ERROR INTERRUPT VECTOR FOR CHB
020E 0000              108    ERR_CS_CHB      DW      0
                       109
0210 0000              110    TX_VECTOR_CHA   DW      0       ;TX INTERRUPT VECTOR FOR CHA
0212 0000              111    TX_CS_CHA       DW      0
                       112
0214 0000              113    STS_VECTOR_CHA  DW      0       ;STATUS INTERRUPT VECTOR FOR CHA
0216 0000              114    STS_CS_CHA      DW      0
                       115
0218 0000              116    RX_VECTOR_CHA   DW      0       ;RX INTERRUPT VECTOR FOR CHA
021A 0000              117    RX_CS_CHA       DW      0
                       118
021C 0000              119    ERR_VECTOR_CHA  DW      0       ;ERROR INTERRUPT VECTOR FOR CHA
021E 0000              120    ERR_CS_CHA      DW      0
                       121
                       122    ;MISC RAM LOCATIONS FOR CHANNEL STATUS AND POINTERS
                       123
                       124    ;CHANNEL B POINTERS AND STATUS
                       125
0220 0000              126    TX_POINTER_CHB  DW      0       ;TX BUFFER POINTER FOR CHB
0222 0000              127    TX_LENGTH_CHB   DW      0       ;TX BUFFER LENGTH FOR CHB
0224 0000              128    RX_POINTER_CHB  DW      0       ;RX BUFFER POINTER FOR CHB
0226 0000              129    RX_COUNT_CHB    DW      0       ;RX LENGTH COUNTER FOR CHB
0228 00                130    TX_EMPTY_CHB    DB      0       ;TX DONE FLAG
0229 00                131    RX_READY_CHB    DB      0       ;READY FLAG (1 IF CR_CHR RECEIVED, ELSE 0)
022A 00                132    STATUS_MSG_CHB  DB      0       ;STATUS CHANGE MESSAGE
022B 00                133    ERROR_MSG_CHB   DB      0       ;ERROR STATUS LOCATION (0 IF NO ERROR)
                       134
                       135    ;CHANNEL A POINTERS AND STATUS
                       136
022C 0000              137    TX_POINTER_CHA  DW      0       ;TX BUFFER POINTER FOR CHA
022E 0000              138    TX_LENGTH_CHA   DW      0       ;TX BUFFER LENGTH FOR CHA
0230 0000              139    RX_POINTER_CHA  DW      0       ;RX BUFFER POINTER FOR CHA
0232 0000              140    RX_COUNT_CHA    DW      0       ;RX LENGTH COUNTER FOR CHA
0234 00                141    TX_EMPTY_CHA    DB      0       ;TX DONE FLAG
0235 00                142    RX_READY_CHA    DB      0       ;READY FLAG (1 IF CR_CHR RECEIVED, ELSE 0)
0236 00                143    STATUS_MSG_CHA  DB      0       ;STATUS CHANGE MESSAGE
0237 00                144    ERROR_MSG_CHA   DB      0       ;ERROR STATUS LOCATION (0 IF NO ERROR)
                       145
                       146            DATA    ENDS
                       147
                       148 $1  $EJECT
```

210311-26

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```
MCS-86 MACRO ASSEMBLER   ASYNC8

LOC  OBJ              LINE    SOURCE

                      149
----                  150           ABC     SEGMENT
                      151           ASSUME  CS:ABC,DS:DATA,SS:DATA
0500                  152           ORG     CODE_START
                      153
                      154     ;*****************************************************
                      155     ;*                                                   *
                      156     ;*         PARAMETERS FOR CHANNEL INITIALIZATION     *
                      157     ;*                                                   *
                      158     ;*****************************************************
                      159
                      160     ;CHANNEL B PARAMETERS
                      161
                      162           ;WR1 - INTERRUPT ON ALL RX CHR. VARIABLE INT VECTOR. TX INT ENABLE
0500 01               162     CMDSTRB DB      1.16H
0501 16
                      164           ;WR2 - INTERRUPT VECTOR
0502 02               165           DB      2,(INT_TABLE_BASE/4)
0503 80
                      166           ;WR3 - RX 8 BITS/CHR. RX DISABLE
0504 03               167           DB      3.0C0H
0505 C0
                      168           ;WR4 - X16 CLOCK, 2 STOP BITS. NO PARITY
0506 04               169           DB      4,4CH
0507 4C
                      170           ;WR5 - DTR ACTIVE. TX 8 BITS/CHR. TX ENABLE. RTS ACTIVE
0508 05               171           DB      5.0EAH
0509 EA
                      172           ;WR6 AND WR7 NOT REQUIRED FOR ASYNC
050A 00               173           DB      0,0
050B 00
                      174
                      175     ;CHANNEL A PARAMETERS
                      176
                      177           ;WR1 - INTERRUPT ON ALL RX CHR. TX INT ENABLE
050C 01               178     CMDSTRA DB      1,12H
050D 12
                      179           ;WR2 - VECTORED INTERRUPT FOR 8086
050E 02               180           DB      2,38H
050F 38
                      181           ;WR3 - RX 8 BITS/CHR, RX DISABLE
0510 03               182           DB      3.0C0H
0511 C0
                      183           ;WR4 - X16 CLOCK, 2 STOP BITS. NO PARITY
0512 04               184           DB      4,4CH
0513 4C
                      185           ;WR5 - DTR ACTIVE. TX 8 BITS/CHR. TX ENABLE. RTS ACTIVE
0514 05               186           DB      5.0EAH
0515 EA
                      187           ;WR6 AND WR7 NOT REQUIRED FOR ASYNC
0516 00               188           DB      0,0
0517 00
                      189
                      190 +1  $EJECT
```

210311-27

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```
MCS-86 MACRO ASSEMBLER   ASYNC8

LOC  OBJ              LINE    SOURCE

                       191
                       192     ;START OF COMMAND ROUTINES
                       193
                       194     ;*******************************************************
                       195     ;*                                                     *
                       196     ;*      INITIALIZATION COMMAND FOR THE 8274 - THE 8274 *
                       197     ;*      IS SETUP ACCORDING TO THE PARAMETERS STORED IN *
                       198     ;*      FROM ABOVE STARTING AT CHSTAB FOR CHANNEL B AND*
                       199     ;*      CHSTAA FOR CHANNEL A                           *
                       200     ;*                                                     *
                       201     ;*******************************************************
                       202
0518                   203     INITIAL_8274:
                       204             ;COPY INTERRUPT VECTOR IF AND CS VALUES FROM PROM TO RAM
0518 C70608020806      205             MOV     TX_VECTOR_CHB, OFFSET XMTINB    ;TX DATA VECTOR CHB
051E 8C0E0202          206             MOV     TX_CS_CHB, CS
0522 C70604823506      207             MOV     STS_VECTOR_CHB, OFFSET STAINB   ;STATUS VECTOR CHB
0528 8C0E0602          208             MOV     STS_CS_CHB, CS
052C C70608824906      209             MOV     RX_VECTOR_CHB, OFFSET RCVINB    ;RX DATA VECTOR CHB
0532 8C0E0A02          210             MOV     RX_CS_CHB, CS
0536 C7060C827706      211             MOV     ERR_VECTOR_CHB, OFFSET ERRINB   ;ERROR VECTOR CHB
053C 8C0E0A02          212             MOV     RX_CS_CHB, CS
0540 C706100206D6      213             MOV     TX_VECTOR_CHA, OFFSET XMTINA    ;TX DATA VECTOR CHA
0546 8C0E1202          214             MOV     TX_CS_CHA, CS
054A C70614028906      215             MOV     STS_VECTOR_CHA, OFFSET STAINA   ;STATUS VECTOR CHA
0550 8C0E1602          216             MOV     STS_CS_CHA, CS
0554 C706182CD006      217             MOV     RX_VECTOR_CHA, OFFSET RCVINA    ;RX DATA VECTOR CHA
055A 8C0E1A02          218             MOV     RX_CS_CHA, CS
055E C7061C82FB06      219             MOV     ERR_VECTOR_CHA, OFFSET ERRINA   ;ERROR VECTOR CHA
0564 8C0E1E02          220             MOV     ERR_CS_CHA, CS
                       221
                       222     ;COPY SETUP TABLE PARAMETERS INTO 8274
                       223
0568 BF0005            224             MOV     DI, OFFSET CHSTAB       ;INITIALIZE CHB
056B BA0600            225             MOV     DX, COMMAND_PORT_CHB
056E E82E00            226             CALL    SETUP                   ;COPY CHB PARAMETERS
0571 BF0C05            227             MOV     DI, OFFSET CHSTAA       ;INITIALIZE CHA
0574 BA0200            228             MOV     DX, COMMAND_PORT_CHA
0577 E82500            229             CALL    SETUP                   ;COPY CHA PARAMETERS
                       230
                       231     ;INITIALIZE STATUS BYTES AND FLAGS
                       232
057A B80000            233             MOV     AX, 0
057D A22002            234             MOV     ERROR_MSG_CHB, AL       ;CLEAR ERROR FLAG CHB
0580 A23702            235             MOV     ERROR_MSG_CHA, AL       ;CLEAR ERROR FLAG CHA
0583 A22A02            236             MOV     STATUS_MSG_CHB, AL      ;CLEAR STATUS FLAG CHB
0586 A23602            237             MOV     STATUS_MSG_CHA, AL      ;CLEAR STATUS FLAG CHA
0589 A32602            238             MOV     RX_COUNT_CHB, AX        ;CLEAR RX COUNTER CHB
058C A33202            239             MOV     RX_COUNT_CHA, AX        ;CLEAR RX COUNTER CHA
058F B001              240             MOV     AL, 1
0591 A22902            241             MOV     RX_READY_CHB, AL        ;SET RX DONE FLAG CHB
0594 A23502            242             MOV     RX_READY_CHA, AL        ;SET RX DONE FLAG CHA
0597 A22802            243             MOV     TX_EMPTY_CHB, AL        ;SET TX DONE FLAG CHB
059A A23402            244             MOV     TX_EMPTY_CHA, AL        ;SET TX DONE FLAG CHA
059D FB                245             STI                             ;ENABLE INTERRUPTS
059E C3                246             RET                             ;RETURN - DONE WITH SETUP
                       247
059F 8A05              248     SETUP:  MOV     AL, [DI]                ;PARAMETER COPYING ROUTINE
05A1 3C00              249             CMP     AL, 0
05A3 7404              250             JE      DONE

                                                                        210311-28
```


เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```
LOC  OBJ              LINE    SOURCE

05A5 EE                251            OUT     DX, AL                  ;OUTPUT PARAMETER
05A6 47                252            INC     DI                      ;POINT AT NEXT PARAMETER
05A7 EBF6              253            JMP     SETUP                   ;GO LOOK IT
05A9 C3                254    DONE    RET                             ;DONE - SO RETURN
                       255
                       256 $1  $EJECT
                       257
                       258    ;*****************************************************************
                       259    ;*                                                               *
                       260    ;*    TX CHANNEL B COMMAND ROUTINE - ROUTINE IS CALLED TO        *
                       261    ;*    TRANSMIT A BUFFER.  THE BUFFER STARTING ADDRESS,           *
                       262    ;*    TX_POINTER_CHB, AND THE BUFFER LENGTH, TX_LENGTH_CHB,      *
                       263    ;*    MUST BE INITIALIZED BY THE CALLING PROGRAM                 *
                       264    ;*    BOTH ITEMS ARE WORD VARIABLES                              *
                       265    ;*                                                               *
                       266    ;*****************************************************************
                       267
05AA                   268    TX_COMMAND_CHB:
05AA 50                269            PUSH    AX              ;SAVE REGISTERS
05AB 57                270            PUSH    DI
05AC 52                271            PUSH    DX
05AD C606230200        272            MOV     TX_EMPTY_CHB, 0 ;CLEAR EMPTY FLAG
0582 BA0400            273            MOV     DX, DATA_PORT_CHB       ;SETUP PORT POINTER
0585 883E2002          274            MOV     DI, TX_POINTER_CHB      ;GET TX BUFFER POINTER CHB
0589 8A05              275            MOV     AL, [DI]                ;GET FIRST CHARACTER TO TX
058B EE                276            OUT     DX, AL                  ;OUTPUT IT TO 8274 TO GET IT STARTED
058C 5A                277            POP     DX
058D 5F                278            POP     DI
058E 58                279            POP     AX
058F C3                280            RET                             ;RETURN
                       281
                       282    ;*****************************************************************
                       283    ;*                                                               *
                       284    ;*    TX CHANNEL A COMMAND ROUTINE - ROUTINE IS CALLED TO        *
                       285    ;*    TRANSMIT A BUFFER.  THE BUFFER STARTING ADDRESS,           *
                       286    ;*    TX_POINTER_CHA, AND THE BUFFER LENGTH, TX_LENGTH_CHA,      *
                       287    ;*    MUST BE INITIALIZED BY THE CALLING PROGRAM                 *
                       288    ;*    BOTH ITEMS ARE WORD VARIABLES                              *
                       289    ;*                                                               *
                       290    ;*****************************************************************
                       291
05C0                   292    TX_COMMAND_CHA:
05C0 50                293            PUSH    AX              ;SAVE REGISTERS
05C1 57                294            PUSH    DI
05C2 52                295            PUSH    DX
05C3 C606F48200        296            MOV     TX_EMPTY_CHA, 0 ;CLEAR EMPTY FLAG
05C8 BA0000            297            MOV     DX, DATA_PORT_CHA       ;SETUP PORT POINTER
05CB 883E2C82          298            MOV     DI, TX_POINTER_CHA      ;GET TX BUFFER POINTER CHA
05CF 8A05              299            MOV     AL, [DI]                ;GET FIRST CHARACTER TO TX
05D1 EE                300            OUT     DX, AL                  ;OUTPUT IT TO 8274 TO GET IT STARTED
05D2 5A                301            POP     DX
05D3 5F                302            POP     DI
05D4 58                303            POP     AX
05D5 C3                304            RET                             ;RETURN
                       305
                       306    ;*****************************************************************
                       307    ;*                                                               *
                       308    ;*    RX COMMAND FOR CHANNEL B - THE CALLING ROUTINE MUST        *
                       309    ;*    INITIALIZE RX_POINTER_CHB TO POINT AT THE RECEIVE          *
                       310    ;*    BUFFER BEFORE CALLING THIS ROUTINE                         *
```

210311-29

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```
MCS-86 MACRO ASSEMBLER   ASYNCB


LOC  OBJ              LINE    SOURCE

                       311    ;*                                                  *
                       312    ;****************************************************
                       313
0526                   314    RX_COMMAND_CHB
0526 50                315           PUSH    AX              ;SAVE REGISTERS
0527 52                316           PUSH    DX
0528 C606290200        317           MOV     RX_READY_CHB, 0 ;CLEAR RX READY FLAG
052D C706260B0000      318           MOV     RX_COUNT_CHB, 0 ;CLEAR RX COUNTER
0533 BA0B00            319           MOV     DX, COMMAND_PORT_CHB    ;POINT AT COMMAND PORT
0536 B003              320           MOV     AL, 3           ;SET UP FOR WR3
0538 EE                321           OUT     DX, AL
0539 B0C1              322           MOV     AL, 0C1H        ;WR3 = 6 BITS/CHR, ENABLE RX
053B EE                323           OUT     DX, AL
053C 5A                324           POP     DX
053D 58                325           POP     AX
053E C3                326           RET                     ;RETURN
                       327
                       328    ;****************************************************
                       329    ;*                                                  *
                       330    ;*    RX COMMAND FOR CHANNEL A - THE CALLING ROUTINE MUST     *
                       331    ;*    INITIALIZE RX_POINTER_CHA TO POINT AT THE RECEIVE       *
                       332    ;*    BUFFER BEFORE CALLING THIS ROUTINE                      *
                       333    ;*                                                  *
                       334    ;****************************************************
                       335
053F                   336    RX_COMMAND_CHA
053F 50                337           PUSH    AX              ;SAVE REGISTERS
0540 52                338           PUSH    DX
0541 C606350200        339           MOV     RX_READY_CHA, 0 ;CLEAR RX READY FLAG
0546 C706320B0000      340           MOV     RX_COUNT_CHA, 0 ;CLEAR RX COUNTER
054C BA0200            341           MOV     DX, COMMAND_PORT_CHA    ;POINT AT COMMAND PORT
054F B003              342           MOV     AL, 3           ;SET UP FOR WR3
0551 EE                343           OUT     DX, AL
0552 B0C1              344           MOV     AL, 0C1H        ;WR3 = 6 BITS/CHR, ENABLE RX
0554 EE                345           OUT     DX, AL
0555 5A                346           POP     DX
0556 58                347           POP     AX
0557 C3                348           RET                     ;RETURN
                       349
                       350 *1 $EJECT
                       351
                       352    ;****************************************************
                       353    ;*                                                  *
                       354    ;*          START OF INTERRUPT SERVICE ROUTINES     *
                       355    ;*                                                  *
                       356    ;****************************************************
                       357
                       358    ;CHANNEL B TRANSMIT DATA SERVICE ROUTINE
                       359
0558 52                360    XXTINB: PUSH    DX              ;SAVE REGISTERS
0559 57                361           PUSH    DI
055A 50                362           PUSH    AX
055B E80201            363           CALL    EOI             ;SEND EOI COMMAND TO 8274
055E FF062002          364           INC     TX_POINTER_CHB  ;POINT TO NEXT CHARACTER
0562 FF0E2202          365           DEC     TX_LENGTH_CHB   ;DEC LENGTH COUNTER
0566 740E              366           JE      XIB             ;TEST IF DONE
0568 BA0400            367           MOV     DX, DATA_PORT_CHB       ;NOT DONE - GET NEXT CHARACTER
056B 8B3E2002          368           MOV     DI, TX_POINTER_CHB
056F 8A05              369           MOV     AL, [DI]        ;PUT CHARACTER IN AL
0571 EE                370           OUT     DX, AL          ;OUTPUT IT TO 8274
```

210311-30

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```
MCS-86 MACRO ASSEMBLER   ASYNCB

LOC  OBJ              LINE    SOURCE

0622 58               371             POP     AX              ;RESTORE REGISTERS
0623 5F               372             POP     DI
0624 5A               373             POP     DX
0625 CF               374             IRET                    ;RETURN TO FOREGROUND
0626 BA0600           375     XIB:    MOV     DX, COMMAND_PORT_CHB    ;ALL CHARACTERS HAVE BEEN SENT
0629 B028             376             MOV     AL, 28H         ;RESET TRANSMITTER INTERRUPT PENDING
062B EE               377             OUT     DX, AL
062C C606280201       378             MOV     TX_EMPTY_CHB, 1 ;DONE - SO SET TX EMPTY FLAG CHB
0631 58               379             POP     AX              ;RESTORE REGISTERS
0632 5F               380             POP     DI
0633 5A               381             POP     DX
0634 CF               382             IRET                    ;RETURN TO FOREGROUND
                      383
                      384     ;CHANNEL B STATUS CHANGE SERVICE ROUTINE
                      385
0635 52               386     STAINB: PUSH    DX              ;SAVE REGISTERS
0636 57               387             PUSH    DI
0637 50               388             PUSH    AX
0638 E80500           389             CALL    EOI             ;SEND EOI COMMAND TO 8274
063B BA0600           390             MOV     DX, COMMAND_PORT_CHB
063E EC               391             IN      AL, DX          ;READ RR0
063F A22A02           392             MOV     STATUS_MSG_CHB, AL      ;PUT RR0 IN STATUS MESSAGE
0642 B010             393             MOV     AL, 10H         ;SEND RESET STATUS INT COMMAND TO 8274
0644 EE               394             OUT     DX, AL
0645 58               395             POP     AX              ;RESTORE REGISTERS
0646 5F               396             POP     DI
0647 5A               397             POP     DX
0648 CF               398             IRET
                      399
                      400     ;CHANNEL B RECEIVED DATA SERVICE ROUTINE
                      401
0649 52               402     RCVINB: PUSH    DX              ;SAVE REGISTERS
064A 57               403             PUSH    DI
064B 50               404             PUSH    AX
064C E8C100           405             CALL    EOI             ;SEND EOI COMMAND TO 8274
064F 8B3E2402         406             MOV     DI, RX_POINTER_CHB      ;GET RX CHB BUFFER POINTER
0653 BA0400           407             MOV     DX, DATA_PORT_CHB
0656 EC               408             IN      AL, DX          ;READ CHARACTER
0657 8805             409             MOV     [DI], AL        ;STORE IN BUFFER
0659 FF062402         410             INC     RX_POINTER_CHB  ;BUMP THE BUFFER POINTER
065D FF062602         411             INC     RX_COUNT_CHB    ;BUMP THE COUNTER
0661 3C0D             412             CMP     AL, CR_CHP      ;TEST IF LAST CHARACTER TO BE RECEIVED
0663 75AE             413             JNE     RIB
0665 C606290201       414             MOV     RX_READY_CHB, 1 ;YES, SET READY FLAG
066A BA0600           415             MOV     DX, COMMAND_PORT_CHB    ;POINT AT COMMAND PORT
066D B002             416             MOV     AL, 2           ;POINT AT WR2
066F EE               417             OUT     DX, AL
0670 B0C0             418             MOV     AL, 0C0H        ;DISABLE RX
0672 EE               419             OUT     DX, AL
0673 58               420     RIB:    POP     AX              ;EITHER WAY, RESTORE REGISTERS
0674 5F               421             POP     DI
0675 5A               422             POP     DX
0676 CF               423             IRET                    ;RETURN TO FOREGROUND
                      424
                      425     ;CHANNEL B ERROR SERVICE ROUTINE
                      426
0677 52               427     ERRINB: PUSH    DX              ;SAVE REGISTERS
0678 50               428             PUSH    AX
0679 E88400           429             CALL    EOI             ;SEND EOI COMMAND TO 8274
067C BA0600           430             MOV     DX, COMMAND_PORT_CHB
```

210311-31

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```
MCS-86 MACRO ASSEMBLER   ASYNCB


LOC  OBJ                LINE    SOURCE

067F B001                431            MOV     AL, 1           ;POINT AT RR1
0681 EE                  432            OUT     DX, AL
0682 EC                  433            IN      AL, DX          ;READ RR1
0683 A22D82              434            MOV     ERROR_MSG_CHB, AL       ;SAVE IT IN ERROR FLAG
0686 B030                435            MOV     AL, 30H         ;SEND RESET ERROR COMMAND TO 8274
0688 EE                  436            OUT     DX, AL
0689 58                  437            POP     AX              ;RESTORE REGISTERS
068A 5A                  438            POP     DX
068B CF                  439            IRET                    ;RETURN TO FOREGROUND
                         440
                         441    ;CHANNEL A TRANSMIT DATA SERVICE ROUTINE
                         442
068C 52                  443    XMTINA: PUSH    DX              ;SAVE REGISTERS
068D 57                  444            PUSH    DI
068E 50                  445            PUSH    AX
068F E87E00              446            CALL    EOI             ;SEND EOI COMMAND TO 8274
0692 FF062C82            447            INC     TX_POINTER_CHA  ;POINT TO NEXT CHARACTER
0696 FF0E2E82            448            DEC     TX_LENGTH_CHA   ;DEC LENGTH COUNTER
069A 740E                449            JE      XIA             ;TEST IF DONE
069C BA0000              450            MOV     DX, DATA_PORT_CHA       ;NOT DONE - GET NEXT CHARACTER
069F 8B3E2C82            451            MOV     DI, TX_POINTER_CHA
06A3 8A05                452            MOV     AL, [DI]        ;PUT CHARACTER IN AL
06A5 EE                  453            OUT     DX, AL          ;OUTPUT IT TO 8274
06A6 58                  454            POP     AX              ;RESTORE REGISTERS
06A7 5F                  455            POP     DI
06A8 5A                  456            POP     DX
06A9 CF                  457            IRET                    ;RETURN TO FOREGROUND
06AA BA0200              458    XIA:    MOV     DX, COMMAND_PORT_CHA    ;ALL CHARACTERS HAVE BEEN SENT
06AD B028                459            MOV     AL, 28H         ;RESET TRANSMITTER INTERRUPT PENDING
06AF EE                  460            OUT     DX, AL
06B0 C606348201          461            MOV     TX_EMPTY_CHA, 1 ;DONE - SO SET TX EMPTY FLAG CHB
06B5 58                  462            POP     AX              ;RESTORE REGISTERS
06B6 5F                  463            POP     DI
06B7 5A                  464            POP     DX
06B8 CF                  465            IRET                    ;RETURN TO FOREGROUND
                         466
                         467    ;CHANNEL A STATUS CHANGE SERVICE ROUTINE
                         468
06B9 52                  469    STAINA: PUSH    DX              ;SAVE REGISTERS
06BA 57                  470            PUSH    DI
06BB 50                  471            PUSH    AX
06BC E85100              472            CALL    EOI             ;SEND EOI COMMAND TO 8274
06BF BA0200              473            MOV     DX, COMMAND_PORT_CHA
06C2 EC                  474            IN      AL, DX          ;READ RR0
06C3 A23682              475            MOV     STATUS_MSG_CHA, AL      ;PUT RR0 IN STATUS MESSAGE
06C6 B010                476            MOV     AL, 10H         ;SEND RESET STATUS INT COMMAND TO 8274
06C8 EE                  477            OUT     DX, AL
06C9 58                  478            POP     AX              ;RESTORE REGISTERS
06CA 5F                  479            POP     DI
06CB 5A                  480            POP     DX
06CC CF                  481            IRET
                         482
                         483    ;CHANNEL A RECEIVED DATA SERVICE ROUTINE
                         484
06CD 52                  485    RCVINA: PUSH    DX              ;SAVE REGISTERS
06CE 57                  486            PUSH    DI
06CF 50                  487            PUSH    AX
06D0 E83D00              488            CALL    EOI             ;SEND EOI COMMAND TO 8274
06D3 8B3E3082            489            MOV     DI, RX_POINTER_CHA      ;GET RX CHA BUFFER POINTER
06D7 BA0000              490            MOV     DX, DATA_PORT_CHA
```

210311-32

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```
MCS-86 MACRO ASSEMBLER   ASYNCB


LOC  OBJ                 LINE    SOURCE

06DA EC                  491             IN      AL, DX          ;READ CHARACTER
06DB 8805                492             MOV     [DI], AL        ;STORE IN BUFFER
06DD FF063802            493             INC     RX_POINTER_CHA  ;BUMP THE BUFFER POINTER
06E1 FF063202            494             INC     RX_COUNT_CHA    ;BUMP THE COUNTER
06E5 3C0D                495             CMP     AL, CR_CHR      ;TEST IF LAST CHARACTER TO BE RECEIVED?
06E7 750E                496             JNE     RIA
06E9 C606350201          497             MOV     RX_READY_CHA, 1 ;YES, SET READY FLAG
06EE BA0200              498             MOV     DX, COMMAND_PORT_CHA    ;POINT AT COMMAND PORT
06F1 B003                499             MOV     AL, 3           ;POINT AT WR3
06F3 EE                  500             OUT     DX, AL
06F4 B0C8                501             MOV     AL, 0C8H        ;DISABLE RX
06F6 EE                  502             OUT     DX, AL
06F7 58                  503     RIA:    POP     AX              ;EITHER WAY, RESTORE REGISTERS
06F8 5F                  504             POP     DI
06F9 5A                  505             POP     DX
06FA CF                  506             IRET                    ;RETURN TO FOREGROUND
                         507
                         508     ;CHANNEL A ERROR SERVICE ROUTINE
                         509
06FB 52                  510     ERRINA: PUSH    DX              ;SAVE REGISTERS
06FC 50                  511             PUSH    AX
06FD E81000              512             CALL    EOI             ;SEND EOI COMMAND TO 8274
0700 BA0200              513             MOV     DX, COMMAND_PORT_CHA
0703 B001                514             MOV     AL, 1           ;POINT AT RR1
0705 EE                  515             OUT     DX, AL
0706 EC                  516             IN      AL, DX          ;READ RR1
0707 A23702              517             MOV     ERROR_MSG_CHA, AL       ;SAVE IT IN ERROR FLAG
070A B038                518             MOV     AL, 38H         ;SEND RESET ERROR COMMAND TO 8274
070C EE                  519             OUT     DX, AL
070D 58                  520             POP     AX              ;RESTORE REGISTERS
070E 5A                  521             POP     DX
070F CF                  522             IRET                    ;RETURN TO FOREGROUND
                         523
                         524     ;END-OF-INTERRUPT ROUTINE - SENDS EOI COMMAND TO 8274.
                         525     ;  THIS COMMAND MUST ALWAYS TO ISSUED ON CHANNEL A
                         526
0710 50                  527     EOI:    PUSH    AX              ;SAVE REGISTERS
0711 52                  528             PUSH    DX
0712 BA0200              529             MOV     DX, COMMAND_PORT_CHA    ;ALWAYS FOR CHANNEL A !!!
0715 B038                530             MOV     AL, 38H
0717 EE                  531             OUT     DX, AL
0718 5A                  532             POP     DX
0719 58                  533             POP     AX
071A C3                  534             RET
                         535
                         536     ;END OF CODE ROUTINE
                         537
                         538             ABC     ENDS
                         539             END

ASSEMBLY COMPLETE, NO ERRORS FOUND
```

210311-33

REFERENCES

1. 8274 Multiprotocol Serial Controller (MPSC) Data Sheet, Intel Corporation, California, 1980.
2. Basics of Data Communication, Electronics Book Series, McGraw-Hill, New York, 1976.
3. Telecommunications and the Computer, J. Martin, Prentice-Hall, New Jersey, 1976.
4. Technical Aspects of Data Communications, J. McNamara, DEC Press, Massachusetts, 1977.
5. Miscellaneous Data Communications Standards—EIA RS-232-C, EIA RS-422, EIA RS-423, EIA Standard Sales, Washington, D.C.

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้