บทที่ 6

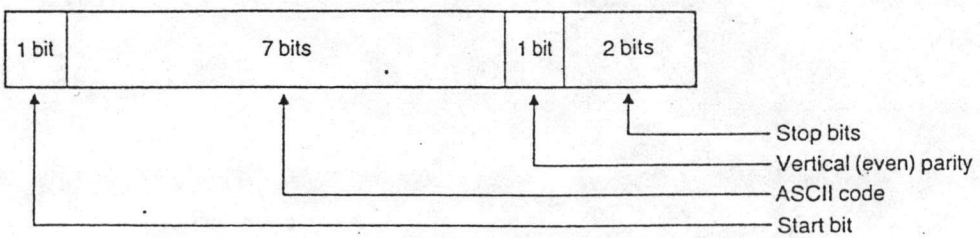
การส่งโปรแกรมไป PC

6.1 หลักการ

การส่งโปรแกรมรหัสภายในของ PC จากไมโครคอมพิวเตอร์ไปยัง PC จะส่งผ่านทางพอร์ทอนุกรม โดยใช้มาตรฐาน RS-232C เพราะทั้งไมโครคอมพิวเตอร์และ PC มีพอร์ตมาตรฐานนี้ ในการใช้งานจะต้องตั้งค่าพารามิเตอร์ที่ใช้ควบคุมการทำงานของพอร์ตเสียก่อน และทั้งบนไมโครคอมพิวเตอร์และ PC จะต้องตั้งค่าพารามิเตอร์เหล่านี้ให้ตรงกัน ในการสื่อสารในงานวิจัยนี้ไมโครคอมพิวเตอร์จะเป็นหลัก (Master) ที่จะก่อให้เกิดการสื่อสาร ส่วน PC จะรอรับคำสั่ง (Slave) จากไมโครคอมพิวเตอร์และปฏิบัติตาม

ในการออกแบบโปรแกรมการสื่อสารบนไมโครคอมพิวเตอร์ สามารถทำได้ใน 2 ลักษณะ ได้แก่ แบบแรกใช้เทคนิคการหยั่งเสียง (Polling) คือการจรวจสอบอยู่เสมอว่าพร้อมหรือยัง และแบบการขัดจังหวะ (Interrupt) คือ เมื่อไรที่พร้อมหรือต้องการจะคุยด้วยก็ให้มาเรียก ทั้งสองแบบมีข้อดีและข้อเสียแตกต่างกัน แบบหยั่งเสียงเสียเวลาในการเฝ้ารอ แต่ถ้าไม่มีงานอื่นจะทำอยู่แล้วก็ไม่น่าเป็นปัญหา แบบขัดจังหวะจะเหมาะสมกับระบบที่มีงานหลายๆงานที่ต้องทำพร้อมๆกัน ในการออกแบบจึงเลือกใช้เทคนิคการหยั่งเสียงกับการส่งข้อมูลเพราะถือว่าเป็นงานหลักที่จะต้องทำและเลือกใช้เทคนิคการขัดจังหวะกับการรับข้อมูลจาก PC

เมื่อมีการสั่งให้ส่งข้อมูล ระบบจะสร้างรูปแบบในการส่งสำหรับตัวอักษรแต่ละตัว ดังแสดงในรูปที่ 6.1 เริ่มต้นระบบจะแทรก Start bit เข้าไปที่ส่วนหัว และคำนวณค่า Parity (หากได้รับการสั่งไว้) ป้อนต่อท้ายจากข้อมูลสุดท้ายและจะปิดท้ายข้อมูลด้วย Stop bit



รูปที่ 6.1 : รูปแบบของข้อมูลในการสื่อสารแบบอนุกรม

6.1.1 การเตรียมการสำหรับไมโครคอมพิวเตอร์

ในขั้นต้นจะต้องตั้งค่าพารามิเตอร์ในการสื่อสารก่อน พารามิเตอร์เหล่านี้ไมโครคอมพิวเตอร์ได้จัดเก็บไว้ในตำแหน่งที่แน่นอน การอ้างอิงจะต้องอ้างอิงตามหมายเลขที่ระบุไว้ในคู่มือของไมโครคอมพิวเตอร์ ซึ่งจะขอนำมาอธิบายดังนี้[๒]

I/O Address	Register Selected	DLAB State
XF8	TX buffer	0 (write)
XF8	RX buffer	0 (read)
XF8	Divisor Latch LSB	1
XF9	Divisor Latch MSB	1
XF9	Interrupt Enable Register	0
XFA	Interrupt Identification Register	
XFB	Line Control Register	
XFC	Modem Control Register	
XFD	Line Status Register	
XFE	Modem Status Register	
XFF	Reserved	

ตารางที่ 6.1 : พารามิเตอร์ควบคุมการสื่อสารแบบอนุกรม

ตำแหน่ง 3F8

ใช้เป็นที่รับและส่งตัวอักษร 1 ตัวอักษร (8 บิต) ทั้งนี้จะต้องตั้งค่า DLAB (บิตที่ 7 ของ 3FB) ให้เป็น 0

ตำแหน่ง 3F9

หากตั้งค่า DLAB ให้เป็น 0 แล้ว ตำแหน่งนี้จะเก็บพารามิเตอร์สำหรับควบคุมการยอมให้เกิดการขัดจังหวะ (Interrupt enable register) ซึ่งมีรายละเอียดดังนี้

บิตที่ 4 ถึง 7 = 0

บิตที่ 3 : ยอมให้เกิดการขัดจังหวะจากโมเด็ม

บิตที่ 2 : ยอมให้เกิดการขัดจังหวะเมื่อการรับข้อมูลมีปัญหา

บิตที่ 1 : ยอมให้เกิดการขัดจังหวะเมื่อระบบพร้อมจะส่งข้อมูล

บิตที่ 0 : ยอมให้เกิดการขัดจังหวะเมื่อได้รับข้อมูล

เมื่อต้องการการควบคุมในลักษณะใดก็ให้ตั้งค่าบิตนั้นเป็น 1 ในงานวิจัยนี้จึงส่งข้อมูล 0000 0101 มาให้พอร์ตนี้ เพื่อยอมให้เกิดการขัดจังหวะเมื่อ PC ติดต่อมา และในกรณีเกิดปัญหาในการรับข้อมูล

---

1. IBM. PC/AT Technical Reference. Serial/Parellel Adapter. Florida : 1984. pp.1-24.

2. ธานินทร์ ถาวรศาสนวงศ์ และ ทินกร คุ้ก. การอินเตอร์เฟส IBM PC. กรุงเทพมหานคร : ฟิสิกส์เซ็นเตอร์การพิมพ์. หน้า 63-103.

ตำแหน่ง 3FA

เป็นตำแหน่งที่จะรายงานให้ทราบว่า ในการขัดจังหวะที่เกิดขึ้นนั้น เป็นผลมาจากสาเหตุอะไร ในการออกแบบจะให้ความสนใจกับ 2 บิตนี้เป็นพิเศษ

บิตที่ 0 : เนื่องจากเกิดปัญหาในการรับข้อมูล

บิตที่ 1 : ได้รับข้อมูลมา 1 ตัวอักษร

ตำแหน่ง 3FB

ใช้ควบคุมรูปแบบของข้อมูลในการสื่อสาร มีรายละเอียดดังนี้

บิต 0, 1 : บิต 1 บิต 0 ขนาดตัวอักษร (บิต)

0	0	5
0	1	6
1	0	7
1	1	8

บิต 2 : ความยาวของ Stop bit

0 : 1 Stop bit

1 : – 5 ตัวอักษร : 1.5 Stop bit

– 6,7,8 ตัวอักษร : 2 Stop bit

บิต 3 : การกระตุ้นการตรวจสอบ parity

บิต 4 : ประเภทของการตรวจสอบ parity

0 : even parity

1 : odd parity

บิต 5 : Stuck parity

บิต 6 : Set break

บิต 7 : DLAB (เกี่ยวพันกับค่าพารามิเตอร์อื่นๆ)

ในรูปที่ 6.1 คือรูปแบบของข้อมูลที่ PC เครื่องนี้กำหนดไว้ ดังนั้นจึงต้องส่งข้อมูล 0001110 ให้กับตำแหน่งนี้

ตำแหน่ง 3FD

แสดงสถานะต่างๆในการสื่อสาร ได้แก่

บิต 0 : มีข้อมูลเข้ามา

บิต 1 : มีข้อมูลใหม่มาทับข้อมูลครั้งก่อนซึ่งยังมิได้อ่าน

บิต 2 : ข้อมูลที่รับเข้ามา มี parity ไม่ถูกต้อง

บิต 3 : ขนาดของ Stop bit ไม่ตรงตามที่ตกลงกัน

บิต 4 : เกิดข้อมูลที่เป็น 0 มีความยาวมากกว่าเฟรม (Frame) ปกติ

บิต 5 : พร้อมจะส่ง

บิต 6 : ข้อมูลที่ได้รับถูกอ่านไปแล้ว

บิต 7 : = 0

ในการตั้งค่าพารามิเตอร์นอกจากจะสามารถอ้างอิงไปตามตำแหน่งต่างๆข้างต้นได้โดยตรงแล้ว ยังสามารถเรียกใช้โปรแกรมใน BIOS ซึ่งจะช่วยตั้งค่าพารามิเตอร์ต่างๆได้เช่นกัน โปรแกรมใน BIOS ที่จะช่วยจัดการงานนี้ต้องเรียกใช้ Interrupt14 ซึ่งจะต้องส่งค่าพารามิเตอร์ไปให้ดังนี้

รีจิสเตอร์ AH = 0 (Initialize communication port)

รีจิสเตอร์ AL = 111 11 1 10

Baud rate=9600, Even parity, 2 Stop bits, 7 Bits word length

รีจิสเตอร์ DX = 1 (Com 1)

ในการใช้งานแบบการขัดจังหวะนี้ จะต้องกระตุ้น IRQ (Interrupt Request) หมายเลขที่เลือกใช้ด้วย เนื่องจากเลือกใช้พอร์ตการสื่อสารหมายเลข 1 ซึ่งตรงกับ IRQ4 จึงต้องกระตุ้น IRQ4 ให้พร้อมที่จะทำงานด้วย โดยการตั้งค่า Interrupt mask ที่ OCW1 (Operation Control Word 1) ของ IC 8259 (port ($21)) ให้ถูกต้อง (IC 8259 ทำหน้าที่ควบคุมการขัดจังหวะของไมโครคอมพิวเตอร์) ซึ่งมีขั้นตอนดังนี้

- อ่านค่า Interrupt mask เดิมขึ้นมา (port ($21))
- เพิ่ม Interrupt mask สำหรับ IRQ4 ลงไป
- เขียนกลับไปที่เดิม

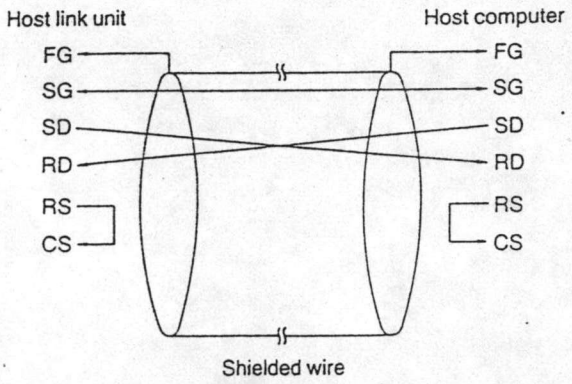
ขั้นตอนทั้ง 3 เขียนเป็นโปรแกรมได้ดังนี้ Port ($21) := Port($21) and $EF

6.1.2 การเตรียมการสำหรับ PC

ในการสื่อสารกับไมโครคอมพิวเตอร์นั้น PC จะมีโมดูลหนึ่งเรียกว่า Host link unit ซึ่งทำหน้าที่นี้ ดังนั้น จึงต้องตั้งค่าพารามิเตอร์ต่างๆที่ตัว Host link unit โดยตั้งที่สวิตช์ด้านหลัง นอกจากรูปแบบข้อมูลที่กำหนดไว้ดังรูป 6.1 แล้ว จะมีพารามิเตอร์อื่นๆ ที่ผู้ใช้ต้องตั้งดังนี้

- การสื่อสาร ใช้มาตรฐาน RS-232C
- Device no. 0 (Dip SW1)
- 9600 bps (Dip SW2)
- ยอมรับระดับคำสั่ง 1,2,3 (Dip SW2)

เพื่อให้ง่ายสำหรับผู้ใช้จึงได้เขียนโปรแกรมย่อย Setup เพื่อแสดงภาพเหมือน ซึ่งวาดเลียนแบบด้านหลังของ Host link unit ผู้ใช้จึงสามารถปรับสวิตช์ต่างๆตามที่ปรากฏได้ ในการเชื่อมต่อกับไมโครคอมพิวเตอร์ Host link unit ได้กำหนดรูปแบบการเชื่อมต่อของสายเคเบิลไว้ดังนี้ ดังแสดงในรูปที่ 6.2

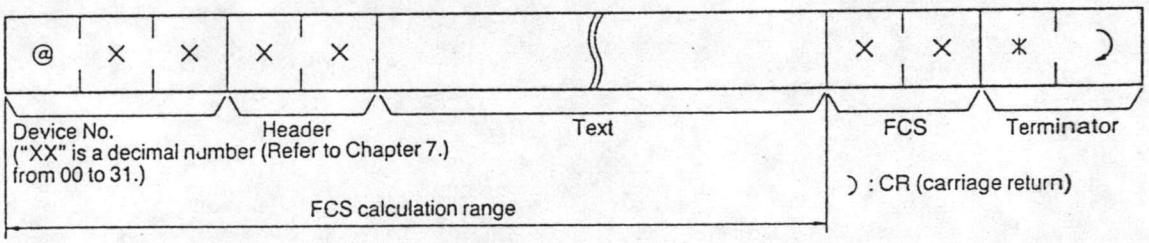


รูปที่ 6.2 : การเชื่อมต่อไมโครคอมพิวเตอร์กับ PC

6.2 การควบคุมการทำงาน

6.2.1 คำสั่งในการสื่อสารของ PC

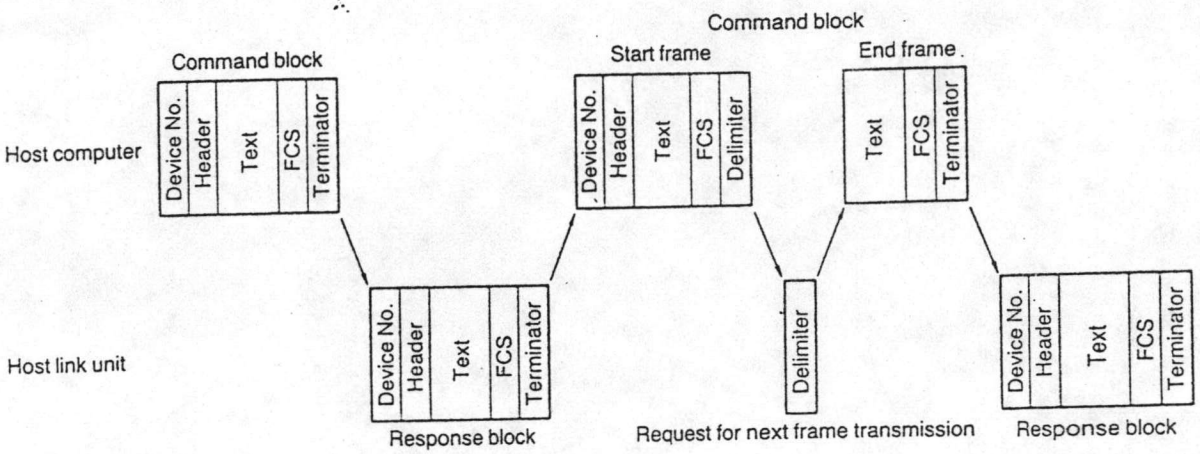
PC ได้เตรียมคำสั่งต่างๆไว้สำหรับใช้ควบคุมการทำงานของ PC ผ่านทางไมโครคอมพิวเตอร์มากมาย แต่คำสั่งที่จะนำมาใช้จะเป็นคำสั่งซึ่งมีผลต่อการส่งโปรแกรมไปยัง PC คำสั่งต่างๆจะมีรูปแบบดังแสดงในรูปที่ 6.3



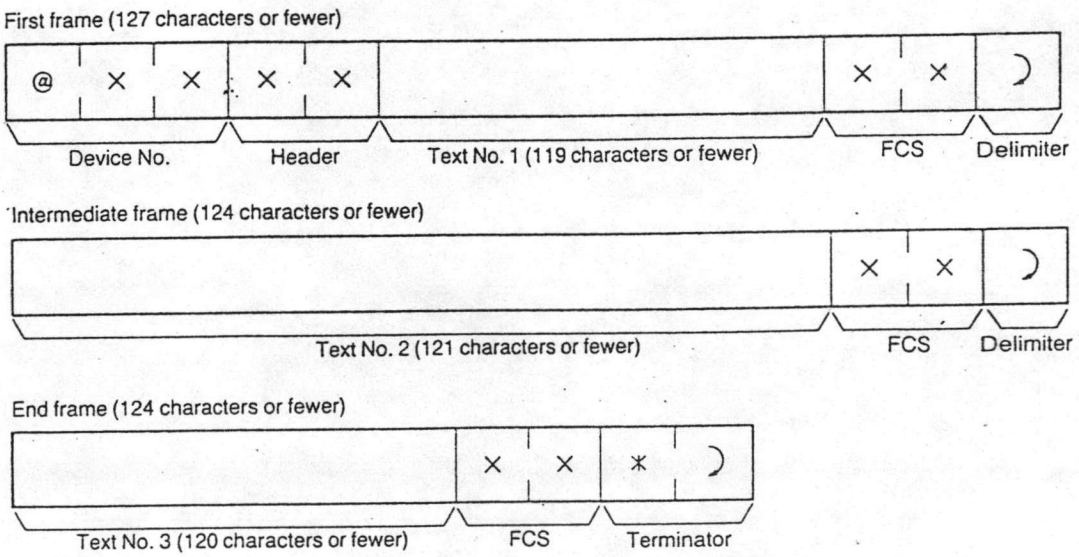
รูปที่ 6.3 : รูปแบบของคำสั่ง

คำสั่งทุกๆคำสั่งจะต้องขึ้นต้นด้วยตัวอักษร "@" และ 2 ไบต์ถัดไปจะต้องเป็น Device no. ซึ่งได้กำหนดไว้ให้เป็น 00 ในส่วน Header คือรหัสของคำสั่งที่ใช้ จากนั้นเป็นส่วนของข้อมูล และในส่วน FCS (Frame check sum) จะต้องคำนวณจากไบต์แรกจนถึงไบต์ก่อนหน้า FCS และปิดท้ายคำสั่งด้วย * และ รหัสของ Carriage return ทั้ง 2 ตัวอักษรนี้รวมเรียกว่า "Terminator" เฉพาะ Carriage Return เรียกว่า Delimiter

ในการสื่อสารจะมีรูปแบบของโปรโตคอล (Protocol) ดังแสดงในรูปที่ 6.4 จะเห็นว่าในกรณีที่ข้อมูลมีมากกว่าจะส่งได้ใน 1 เฟรม จะต้องตัดแบ่งเป็นท่อนๆและจัดส่งไปทีละท่อน PC จะรู้ได้ว่ายังมีข้อมูลต่ออีกโดยดูจากการใช้ Delimiter ปิดท้ายข้อมูลแทน Terminator PC ก็จะตอบรับรู้โดยส่ง Delimiter กลับไป จากนั้นระบบก็จะสามารถส่งข้อมูลต่อไปได้ทันทีโดยไม่ต้องส่งส่วนหัวไปด้วย และเมื่อส่งข้อมูลเฟรมสุดท้ายไปให้ PC จะต้องปิดท้ายด้วย Terminator ทั้งนี้ขนาดของเฟรมจะต้องปฏิบัติตามข้อกำหนดในรูปที่ 6.5



รูปที่ 6.4 : โปรโตคอล



รูปที่ 6.5 : ขนาดของเฟรม

คำสั่งต่างๆที่นำมาใช้ในการส่งโปรแกรมไปยัง PC มีดังนี้[3]

1. คำสั่งจัดกระบวนการเริ่มต้น (Initialize) : เป็นคำสั่งแรกที่ส่งไปให้ PC เพื่อจัดกระบวนการเริ่มต้นใหม่ มีรูปแบบพิเศษซึ่งแตกต่างไปจากคำสั่งอื่นๆดังแสดงในรูปที่ 6.6

Abort command format

@	Device No. ×10'	Device No. ×10°	X	Z	FCS	*	)

Initial command format

@	*	*	)

รูปที่ 6.6 : คำสั่ง Initialize และ Abort

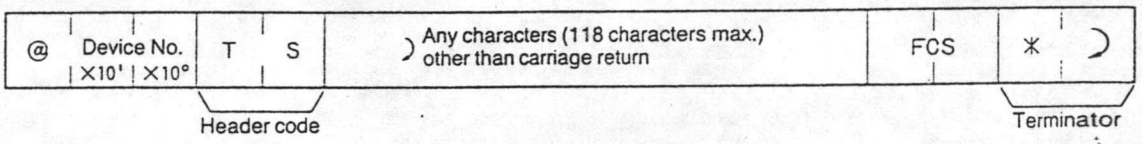
2. คำสั่งยกเลิกการทำงาน (Abort) : มีรูปแบบดังแสดงในรูปที่ 6.6

3. คำสั่งทดสอบการสื่อสาร (Test) : ใช้ทดสอบการสื่อสาร มีรูปแบบดังแสดงในรูปที่ 6.7 หาก PC เข้าใจก็จะส่งคำสั่งนี้กลับมา (Response format 1) หาก PC ไม่สามารถทำตามคำสั่งได้จะตอบกลับมาด้วย Response format 2 โดยบอกเหตุผลด้วย Completion code และหาก PC ไม่สามารถอ่านส่วนหัวของคำสั่งได้จะตอบด้วย Response format 3

---

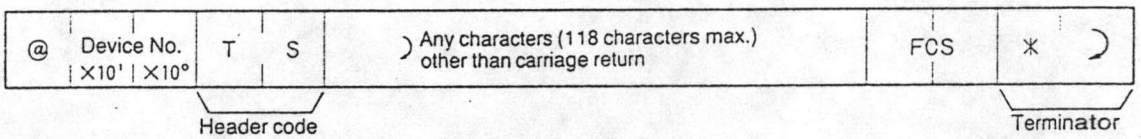
3. OMRON. OMRON user's manual. Host Link Unit (for SYSMAC C Series). Japan : OMRON Tateisi electronics co. pp. 35-69.

Command format



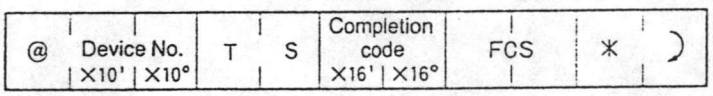
Response format 1

This is the normal response.



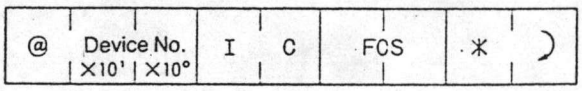
Response format 2

This response is returned from the host link unit when it cannot process the command. The error is identified by the completion code.



Response format 3

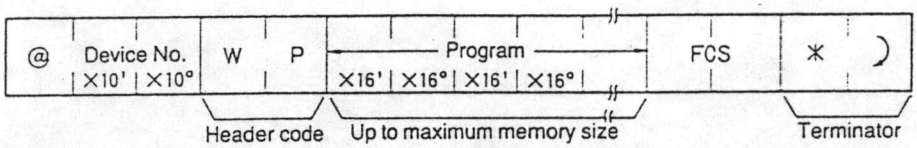
This response is returned if the host link unit cannot read the command's header code.



รูปที่ 6.7 : คำสั่ง Test และ Response format

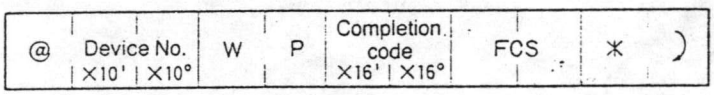
4. คำสั่งเขียนโปรแกรม (Write) : ใช้สำหรับส่งโปรแกรมรหัสภายในไปให้ PC มีรูปแบบดังแสดงในรูปที่ 6.8 หาก PC ไม่สามารถอ่านส่วน Header ได้ จะตอบด้วย Response format2 และหาก PC ได้รับและเข้าใจ หรือเกิดข้อผิดพลาดอื่นๆขึ้นทำให้ไม่สามารถนำโปรแกรมที่ส่งไปใช้ได้ ก็จะตอบด้วย Response format 1 โดยหากได้รับและเข้าใจ Completion code จะมีค่าเป็น 00 หากตอบด้วย Completion code อื่นๆ แสดงว่าเกิดปัญหา (จะกล่าวถึง Completion code ในภายหลัง)

Command format



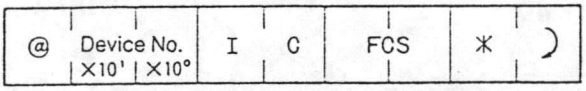
Response format 1

Completion code "00" indicates normal response. Any other end code indicates the host link unit cannot process the command due to the error indicated by the completion code.



Response format 2

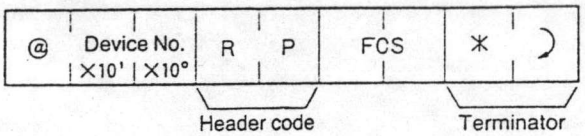
This response is returned from the host link unit when it cannot read the command's header code.



รูปที่ 6.8 : คำสั่ง Write และ Response format

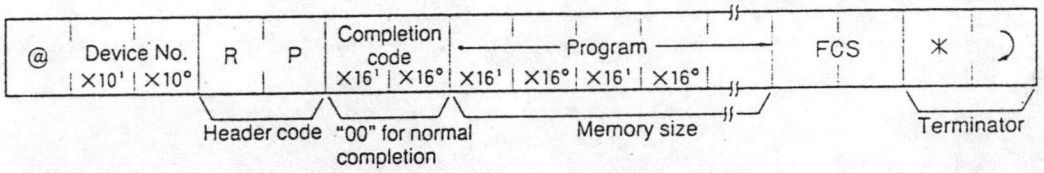
5. คำสั่งอ่านโปรแกรม (Read) : คำสั่งนี้นำมาใช้ทดสอบว่าการเขียนโปรแกรมทำได้ถูกต้องหรือไม่ มีรูปแบบดังแสดงในรูปที่ 6.9 หาก PC เข้าใจก็จะส่งโปรแกรมจากในหน่วยความจำขึ้นมาให้ไมโครคอมพิวเตอร์ตาม Response format1

Command format



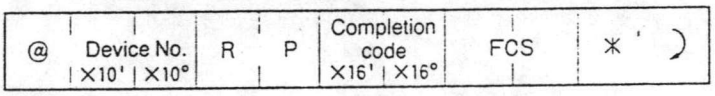
Response format 1

This is the normal response.



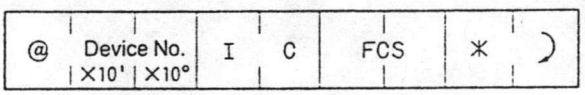
Response format 2

This response is returned from the host link unit when it cannot process the command. The error is identified by the completion code.



Response format 3

This response is returned if the host link unit cannot read the command's header code.



รูปที่ 6.9 : คำสั่ง Read และ Response format

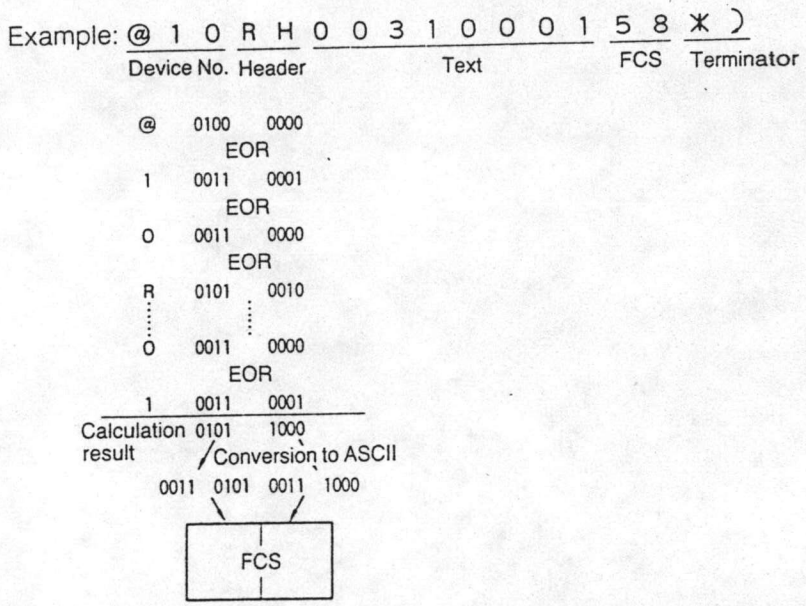
Completion code คือ รหัสที่ PC จะตอบกลับมา เพื่อบอกให้รู้ว่า PC รู้สึกอย่างไรกับคำสั่งที่ส่งไปให้ PC รูปที่ 6.10 แสดง Completion code ต่างๆ

$x16^1$	$x16^0$	
0	0	Normal completion
0	1	Not executable in RUN mode
0	2	Not executable in MONITOR mode
0	3	Not executable with PROM mounted
0	4	Address over (data overflow)
0	B	Not executable in PROGRAM mode
0	C	Not executable in DEBUG mode
0	D	Not executable in LOCAL mode
1	0	Parity error
1	1	Framing error
1	2	Overrun
1	3	FCS error
1	4	Format error (parameter length error)
1	5	Entry number data error (parameter error, data code error, data length error)
1	6	Instruction not found
1	8	Frame length error
1	9	Not executable (due to unexecutable error clear, non-registration of I/O table, etc.)
2	0	I/O table generation impossible (unrecognized remote I/O unit, channel over, duplication of optical transmitting I/O unit)
A	0	Abort due to parity error in transmit data under process
A	1	Abort due to framing error in transmit data under process
A	2	Abort due to overrun in transmit data under process
A	3	Abort due to FCS error in transmit data under process
A	4	Abort due to format error in transmit data under process
A	5	Abort due to entry number data error in transmit data under process
A	8	Abort due to frame length error in transmit data under process
B	0	Unexecutable due to program area capacity other than 16K bytes

รูปที่ 6.10 : Completion code

6.2.2 <u>การคำนวณค่า FCS</u>

FCS คือส่วนที่จะใช้ตรวจสอบความถูกต้องของข้อมูลว่าข้อมูลที่ได้รับผิดเพี้ยนไปจากข้อมูลที่ส่งมาหรือไม่ หากผิดเพี้ยนไป PC ก็จะตอบด้วย Completion code "13" การคำนวณ FCS จะต้องนำข้อมูล ที่ส่งไปให้ PC ตั้งแต่ไบต์แรก (@) จนถึงไบต์สุดท้าย (ก่อนหน้า FCS) มาเอ็กซ์คลูซีฟออร์ (Exclusive or) กัน จนครบทุกตัวทุกข้อมูล ก็จะได้ผลลัพธ์ค่าหนึ่งออกมาขนาด 8 บิต ให้แปลง 4 ไบต์แรกเป็นรหัส ASCII 1 ตัวอักษร และ 4 ไบต์หลังเป็นรหัส ASCII อีก 1 ตัวอักษร ก็จะได้ค่า FCS ขนาด 2 ไบต์ ดังแสดงในรูปที่ 6.11



รูปที่ 6.11 : การคำนวณ FCS

ได้เขียนฟังก์ชั่น FCS เพื่อคำนวณค่า FCS ดังแสดงในรูปที่ 6.12 ในลูป (Loop) ของ คำสั่ง For เป็นการอ่านคำสั่งขึ้นมาเอ็กซ์คลูซีฟออร์กันทีละตัวอักษรจนครบ ส่วนฟังก์ชั่น HexS จะเปลี่ยนผลลัพธ์ที่ได้ให้เป็นรหัส ASCII ขนาด 2 ไบต์

```
Function HexS(Dec:byte):string;
var Rem : integer;
    Hex : string;
Begin
        Hex:='';
        While Dec>0 do
              begin
                   Rem:=Dec mod 16;
                   Dec:=Dec div 16;
                   If Rem=0  then Hex:='0'+Hex else
                   If Rem=1  then Hex:='1'+Hex else
                   If Rem=2  then Hex:='2'+Hex else
                   If Rem=3  then Hex:='3'+Hex else
                   If Rem=4  then Hex:='4'+Hex else
                   If Rem=5  then Hex:='5'+Hex else
                   If Rem=6  then Hex:='6'+Hex else
                   If Rem=7  then Hex:='7'+Hex else
                   If Rem=8  then Hex:='8'+Hex else
                   If Rem=9  then Hex:='9'+Hex else
                   If Rem=10 then Hex:='A'+Hex else
                   If Rem=11 then Hex:='B'+Hex else
                   If Rem=12 then Hex:='C'+Hex else
                   If Rem=13 then Hex:='D'+Hex else
                   If Rem=14 then Hex:='E'+Hex else
                   If Rem=15 then Hex:='F'+Hex;
              end;
        If ord(Hex[0])=0 then Hex:='00';
        If ord(Hex[0])=1 then Hex:='0'+Hex;
        HexS:=Hex;
End;

Function FCS1(St:string):byte;
var Result,i : byte;
Begin
     Result:=ord(St[1]);
     For i:=2 to length(St) do Result:=Result xor ord(St[i]);
     FCS1:=Result;
End;
```

รูปที่ 6.12 : ฟังก์ชั่น FCS

6.2.3 โปรแกรมที่ออกแบบ

โครงสร้างหลัก

ในการส่งโปรแกรมรหัสภายในไปยัง PC ขั้นแรกจะต้องถามผู้ใช้ก่อนว่าจะให้ส่งโปรแกรมของชาร์ตใดไป จากนั้นจึงอ่านข้อมูลขึ้นมาและนำไปส่งให้กับ PC ใน 2 ขั้นตอนสุดท้ายนี้สามารถออกแบบได้หลายรูปแบบ เช่น อ่านข้อมูลขึ้นมาก่อนแล้วจึงค่อยส่งไปให้ PC หรือว่าอ่านไปส่งไปก็ได้ ในแบบแรกจะใช้หน่วยความจำมาก ซึ่งเป็นการเสี่ยงหากเครื่องไมโครคอมพิวเตอร์ที่ใช้มีหน่วยความจำน้อย จึงเลือกวิธีอ่านไปส่งไป และกำหนดให้อ่านครั้งละ 2 กิโลไบต์ (KBytes) มาเก็บไว้ในตัวแปร จากนั้นจึงส่งค่าในตัวแปรนี้ไปยัง PC ตัวแปรที่ใช้ (Prog) มีลักษณะดังนี้

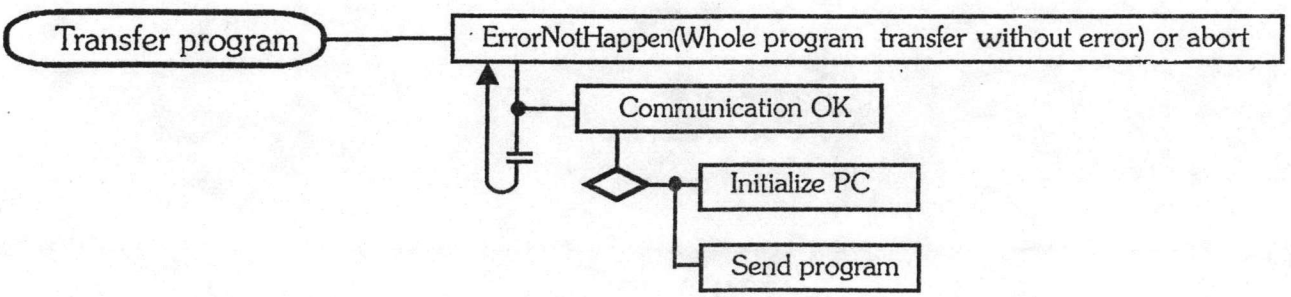
```
TxBuf  =  array(1..2048) of bytes;

Prog   :  TxBuf;
```

ขนาดของข้อมูลที่จะส่งไปยัง PC ในแต่ละครั้ง (Frame) จะมีขนาดจำกัดประมาณ 120 ตัวอักษร ดังนั้นในการอ่านแฟ้มข้อมูลขึ้นมา 1 ครั้ง (2048 ไบต์) ก็จะต้องจัดแบ่งข้อมูลให้เป็นเฟรมย่อยลงไปอีก ทั้งนี้ออกแบบให้เฟรมย่อยมีขนาด 100 ตัวอักษร เฟรมย่อยแต่ละเฟรมที่ไม่ใช่เฟรมสุดท้าย จะปิดท้ายเฟรมด้วย Delimiter และเฟรมสุดท้ายจะปิดท้ายด้วย Terminator

การส่งโปรแกรม

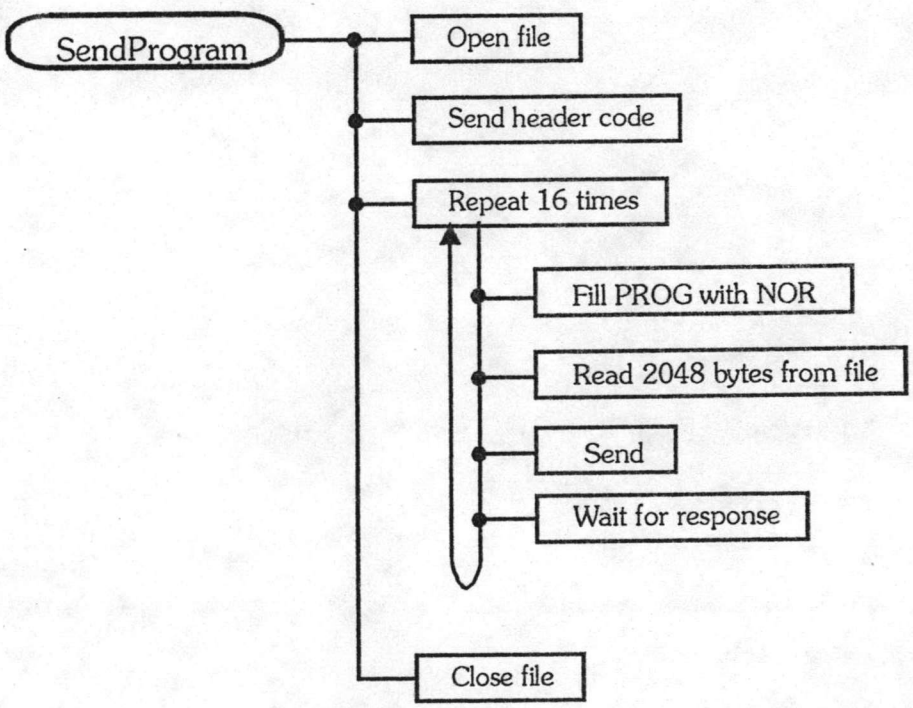
โปรแกรมย่อย Transfer ทำหน้าที่จัดการส่งโปรแกรมไปให้ PC รูปที่ 6.13 แสดง DSD ของโปรแกรมย่อย Transfer ในขั้นแรกจะตรวจสอบการสื่อสารก่อน โดยใช้คำสั่งทดสอบและทดลองเขียนโปรแกรมไปยัง PC หากไม่มีปัญหาใดก็จะส่งคำสั่งจัดกระบวนการเริ่มต้นใหม่ไปยัง PC (Send()) จากนั้นจะเรียกใช้โปรแกรมย่อย SendProgram เพื่อจัดส่งโปรแกรมไปยัง PC หากส่งโปรแกรมไปยัง PC ได้ถูกต้องครบถ้วน แฟลก error ก็จะไม่ถูกเซ็ท ระบบจะรอการตอบรับของ PC แต่ถ้าหากแฟลก error ถูกเซ็ท ระบบก็จะตรวจสอบหาข้อผิดพลาด โดยฟังก์ชั่น ErrorNotHappen



รูปที่ 6.13 : โปรแกรมย่อย Transfer

โปรแกรมย่อย SendProgram ทำหน้าที่อ่านข้อมูลจากแฟ้มข้อมูลที่ระบุและจัดส่งไปให้ PC แต่เนื่องจากโปรแกรมใหม่ที่ PC ได้รับ จะถูกนำไปวางทับโปรแกรมเก่า โดยไม่ได้เคลียร์หน่วยความจำก่อน ในบางครั้งจะทำให้การแสดงโปรแกรมของ PC ผิดพลาดได้ จึงออกแบบให้โปรแกรมย่อย SendProgram นี้ จัดการปรับแต่งให้ โปรแกรมที่จะจัดส่งมีขนาดเท่ากับหน่วยความจำของ PC นี้ (C-500, 32768 ไบต์, 2048x16) โดยในตอนต้น จะเป็นโปรแกรมที่จะจัดส่งและส่วนที่เหลือเป็นคำสั่ง "NOP (No operation)" ก็จะเสมือนการเคลียร์หน่วยความจำก่อนการส่งนั่นเอง

SendProgram มีขั้นตอนการทำงานดังแสดงในรูปที่ 6.14 ขั้นแรกจะเปิดแฟ้มข้อมูล จากนั้นจะส่ง Head ของคำสั่งไปให้ PC ก่อน เพราะในการส่งข้อมูลเดียวกันซึ่งแยกเป็นหลายๆเฟรม จะต้องส่ง Head เพียงครั้งเดียวเท่านั้น จากนั้นจะวนลูป 16 ครั้ง เพื่อให้ส่งโปรแกรมได้ขนาดเท่ากับ 16x2048 = 32768 ไบต์ ภายในลูปชั้นแรกจะบรรจุตัวแปร PROG ด้วยคำสั่ง NOP จากนั้นจึงอ่านข้อมูลจากแฟ้มขึ้นมาและเขียนทับลงไปบน PROG ไล่จากตำแหน่งแรก และเรียกใช้โปรแกรมย่อย Send เพื่อส่งโปรแกรมไปให้ PC ในช่วงสุดท้ายจะต้องรอการตอบจาก PC ก่อนที่จะไปทำงานในรอบใหม่ ในการรอนี้จะต้องรออย่างต่ำ 10 มิลลิวินาที เพราะหากเร็วกว่านี้ PC จะรับข้อมูล ไม่ทัน



รูปที่ 6.14 : โปรแกรมย่อย SendProgram

โปรแกรมย่อย Send จะส่งโปรแกรมไปยัง PC โดยจะนำมาตัดเป็นเฟรมย่อย เฟรมละ 100 ตัวอักษร นอกจากนั้นจะเรียกใช้ฟังก์ชั่น FCS เพื่อคำนวณค่า FCS และส่งต่อท้ายข้อมูลไป และปิดท้ายแฟรมด้วย Delimiter หรือ Terminator แล้วแต่กรณี โปรแกรมย่อย Send นี้จะเรียกใช้โปรแกรมย่อย Tx ซึ่งเป็นหน่วยเล็กที่สุดในการจัดส่งโปรแกรม กล่าวคือ โปรแกรม Tx นี้ จะตรวจสอบสถานะของตำแหน่ง 3FD บิตที่ 5 ว่าระบบพร้อมที่จะส่งข้อมูลหรือยัง ถ้าพร้อมแล้วก็จะส่งข้อมูลไปที่ตำแหน่ง 3F8 1 ตัวอักษร จากนั้นระบบก็จะนำข้อมูลไปแปลงให้เป็นรูปแบบดังในรูปที่ 6.1 และส่งไปให้ PC

ฟังก์ชั่น ErrorNotHappen จะตรวจสอบ Completion code ที่ PC ส่งตอบมาว่าโปรแกรมที่ได้รับถูกต้องครบถ้วนหรือไม่ ถ้าไม่ มีสาเหตุจากอะไร ซึ่งการวิเคราะห์จะตรวจสอบตามข้อมูลในรูปที่ 6.10 หลังจากรู้ปัญหาแล้วก็จะรายงานให้ผู้ใช้ทราบ และหากเป็นปัญหาที่เกิดเนื่องจากสภาพสายหรือข้อผิดพลาดในการส่ง ซึ่งเป็นปัญหาที่ไม่ได้เกิดจากผู้ใช้ ก็ได้ออกแบบให้โปรแกรมจัดการแก้ปัญหาเหล่านี้เอง ด้วยการสั่งให้ PC จัดกระบวนการตั้งต้นใหม่ จากนั้นส่งข้อมูลใหม่อีกครั้งหนึ่ง แต่ในคราวนี้ปรับเวลาในการรอระหว่างเฟรมเพิ่มขึ้นอีก 10 มิลลิวินาที และถ้าหากการส่งยังมีปัญหาอยู่ ก็จะทำเช่นเดิม จนกว่าจะสามารถส่งโปรแกรมไปได้ หรือช่วงเวลาการรอเพิ่มมากกว่า 50 มิลลิวินาที

การขัดจังหวะ

ได้เขียนโปรแกรม IRQ4Handle เพื่อจัดการการขัดจังหวะ เนื่องจากได้อนุญาตให้สามารถเกิดการขัดจังหวะได้จาก 2 สาเหตุ คือ ได้รับข้อมูล และเกิดข้อผิดพลาดขึ้น โปรแกรม IRQ4Handle นี้ จึงต้องตรวจสอบว่าการขัดจังหวะที่เกิดขึ้นนั้น เนื่องมาจากสาเหตุใด ทั้งนี้พิจารณาได้จากค่าในตำแหน่ง 3FA และเมื่อระบุได้ว่าเกิดจากสาเหตุใด จึงเรียกใช้โปรแกรมย่อยจัดการงานเหล่านี้อีกทีหนึ่ง ในกรณีที่เกิดข้อผิดพลาดขึ้น จะเรียกใช้โปรแกรมย่อย CommunicationError โปรแกรมย่อยนี้จะรายงานให้ผู้ใช้ทราบว่าเกิดปัญหาขึ้น และเรียกใช้โปรแกรมย่อย InitializeComminication เพื่อจัดกระบวนการเริ่มต้นใหม่ รูปที่ 6.13 แสดงโปรแกรมย่อย IRQ4Handle ในการเขียนโปรแกรมย่อยให้จัดการการขัดจังหวะจะต้องเปลี่ยน Compiler directive ให้เป็นแบบ Force far calls โดยระบุด้วยเครื่องหมาย {$F+} เพื่อบังคับให้มีการเรียกใช้โปรแกรมย่อยนี้ข้ามเซกเมนต์ (Segment) ได้ และในตอนเริ่มต้นของเนื้อโปรแกรมจะต้องปิดกั้นมิให้ยอมรับการเกิดการขัดจังหวะ เพื่อป้องกันมิให้เกิดการขัดจังหวะแบบรีเคอร์ซีฟ (Recursive interrupt) และในตอนสิ้นสุดการทำงานของโปรแกรมจะต้องอนุญาตให้ยอมรับการขัดจังหวะได้เหมือนเดิม

```
{$F+}
Procedure IRQ4Handle;
{$F-}
Interrupt;
var Event,C : byte;
Begin
     Inline($FA);       {Disable interrupt}
     Event:=Port[$3FA];
     case Event of
          $04    : ReceiveData;
          $06    : CommunicationError;
          end;
     Port[$20]:=$64;    {Reset 8259:IRQ4}
     Inline($FB);       {Enable interrupt}
End;
```

รูปที่ 6.15 : โปรแกรมจัดการการขัดจังหวะ IRQ4Handle

การรับข้อมูล

เมื่อวิเคราะห์การขัดจังหวะและทราบว่ามีข้อมูลใหม่เข้ามา ระบบก็จะเรียกใช้โปรแกรมย่อย ReceiveData ข้อมูลจะทยอยเคลื่อนเข้ามาทีละ 1 ตัวอักษร ทุกๆตัวอักษรที่ได้รับจะทำให้เกิดการขัดจังหวะ 1 ครั้ง เมื่อสิ้นสุดข้อความจะได้รับตัวอักษร Cariage Return โปรแกรมย่อย ReceiveData มีขั้นตอนการทำงานดังนี้

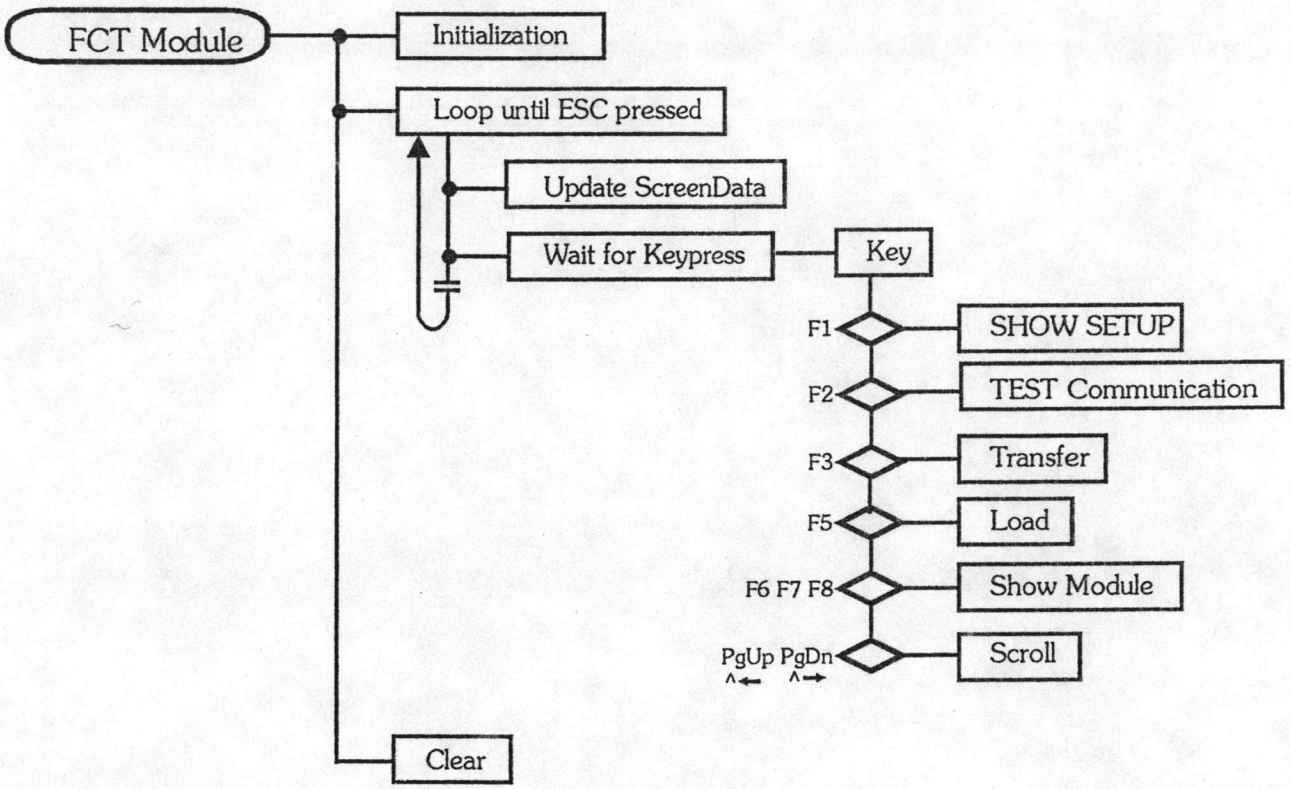
- อ่านตัวอักษรขึ้นมาจากตำแหน่ง 3F8
- นำไปสร้างเป็นข้อความ
- ถ้าหากเป็น Cariage return ($0D) แสดงว่าสิ้นสุดข้อความ ให้เซ็ทแฟลก DataReady เป็น "1" เพื่อแสดงว่าได้รับข้อความครบถ้วนแล้ว

```
Procedure ReceiveData;
var c:byte;
Begin
     c:=Port[$3F8];
     DataIn:=DataIn+chr(c);
     If (c=$0D) then
                    Begin
                         DataReady:=true;
                         Data:=DataIn;
                         DataIn:='';
                    End;
End;
```

รูปที่ 6.16 : โปรแกรมย่อย ReceiveData

6.3 โปรแกรมหลัก

โปรแกรมที่ทำหน้าที่จัดการงานภายในโมดูลการส่งโปรแกรมนี้ (Function Chart Transfer, FCT) มี ขั้นตอนการทำงานดังแสดงในรูปที่ 6.17 ในขั้นตอน Show setup จะแสดงภาพคล้ายของด้านหลังของ Host link unit เพื่อการตั้งค่าพารามิเตอร์ต่างๆ ขั้นตอน Test communication เป็นการทดสอบการสื่อสาร โดยการทดสอบแบ่งเป็น 2 ขั้นตอน ขั้นตอนแรกจะทดสอบด้วยคำสั่ง Test ของ PC และขั้นตอนที่ 2 จะทดสอบการเขียนโปรแกรมลง PC หากทดสอบทั้ง 2 ขั้นตอนผ่าน จะถือว่าสามารถสื่อสารถึงกันได้ ขั้นตอน Transfer เป็นการส่งโปรแกรมรหัสภายในไปยัง PC



รูปที่ 6.17 : DSD ของโมดูล FCT