รีโมทสั่งอาหาร

ORDERING FOOD REMOTE CONTROL

โดย

นาย ประเสริฐ ลดาวัลย์วิวัฒน์

นาย สุรศักดิ์ งามมิตรสมบูรณ์

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตร์บัณฑิต

สาขาอิเล็กทรอนิกส์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2537

ปริญญานิพนธ์ปีการศึกษา 2537

ภาควิชา อิเล็กทรอนิกส์

คณะ วิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

เรื่อง รีโมทสั่งอาหาร (ORDERING FOOD REMOTE COMTROL)

ผู้จัดทำ

นาย ประเสริฐ ลดาวัลย์วิวัฒน์ รหัส 34104210

นาย สุรศักดิ์ งามมิตรสมบูรณ์ รหัส 34108454

.....อาจารย์ที่ปรึกษา

( )

รีโมทสั่งอาหาร

(ORDERING FOOD REMOTE CONTROL)

โดย นาย ประเสริฐ ลดาวัลย์วิวัฒน์
นาย สุรศักดิ์ งามมิตรสมบูรณ์

อาจารย์ที่ปรึกษา อาจารย์ กิตติพล ชิตสกุล

บทคัดย่อ

โครงงานนี้เป็นการพัฒนามาจากโครงงานการส่งข้อมูลผ่านทางอินฟราเรดซึ่งเป็นการจำลองระบบสั่งอาหารผ่านเครื่องส่งระยะไกล โดยโครงงานนี้ได้พัฒนาทั้ง Hard ware และ Software ของเครื่องส่งอินฟราเรด โดยลดการกินกระแสของเครื่องส่งลงอย่างมากเพื่อให้สามารถนำไปใช้งานได้ และยังได้ปรับปรุงการทำงานของภาครับ โดยได้เปลี่ยนแปลงรูปแบบการประมวลผลข้อมูลให้มีความยืดหยุ่นและง่ายต่อการใช้งาน

ABSTRACT

This project presents a new partial design and construction of remote food ordering for using in a restaurant. Based on the project of data transmitting by infrarad, we proposed a new designed hardware providing lower energy consumption and small size menu logging model. We also develope a recieved control software on PC providing more capacity , flexibility and easy to use.

สารบัญ

บทที่ 1

บทนำ

ระบบการสั่งอาหารในภัตราคารในรูปแบบทั่วๆไปจะใช้บริกรจดหรือจำหากจำนวนอาหารไม่มากนักในกรณีเมื่อมีจำนวนโต๊ะอาหารมากขึ้นหรือจำนวนอาหารที่จะสั่งมากขึ้นจะต้องใช้บริกรจำนวนมาก และใช้เวลามากขึ้นด้วย จากปัญหาดังกล่าว จึงมีแนวคิดว่าหากมีระบบสั่งอาหาร ซึ่งสามารถสั่งจากโต๊ะโดยผู้ใช้บริการหรือบริกร ไปยังห้องครัว และแคชเชียร์ได้โดยตรง จะทำให้การบริการสะดวกและรวดเร็วยิ่งขึ้น

จากแนวคิดดังกล่าวจึงได้พัฒนาต้นแบบระบบการสั่งอาหารซึ่งจะประกอบด้วยโมดูลเก็บข้อมูลเมนูที่สร้างไว้ประกอบด้วย แป้นกดรหัสอาหาร ,เลขที่โต๊ะ ฯลฯ จากนั้นจะส่งผ่านทางแสงอินฟราเรดไปยังโมดูลรับ ซึ่งต่อไปยังห้องครัว และโต๊ะคิดเงิน ซึ่งมีเครื่องไมโครคอมพิวเตอร์ และเครื่องพิมพ์คอยพิมพ์รายการอาหารสำหรับห้องครัว และจัดพิมพ์ใบเสร็จเมื่อลูกค้าต้องการ ดังนั้นงานหลักของการพัฒนานี้จะมีสองส่วนใหญ่ ๆ คือ การพัฒนาโมดูลรับเมนูอาหารเป็นเครื่องมือขนาดเล็ก และการพัฒนา Software สำหรับการรับข้อมูลและคำนวนและพิมพ์ใบเสร็จ

ในส่วนของเมนูรับและเก็บเมนูใช้ไมโครคอนโทรลเลอร์ควบคุม เป็นลักษณะ data logger ง่ายๆ ในเบื้องต้นนี้ จะออกแบบไว้สำหรับให้บริกรเป็นผู้กดแป้นใส่รหัสอาหารและส่งข้อมูลเองซึ่งสามารถตรวจสอบรายการอาหารได้โดยดูบนจอแสดงผลแบบ LCD ในส่วนของ Software บน PC นั้นพัฒนาโดยใช้ภาษา C

บทที่2

ความรู้เบื้องต้นเกี่ยวกับไมโครคอนโทรลเลอร์และการสื่อสารข้อมูล

8031 เป็นไมโครคอนโทรลเลอร์แบบชิพเดี่ยว ซึ่งอยู่ในตระกูล MCS-51 ลักษณะโครงสร้างของ 8031 แสดงอยู่ในรูปที่ 1 ซึ่งสถาปัตยกรรมของ 8031 สร้างขึ้นด้วย HMOS

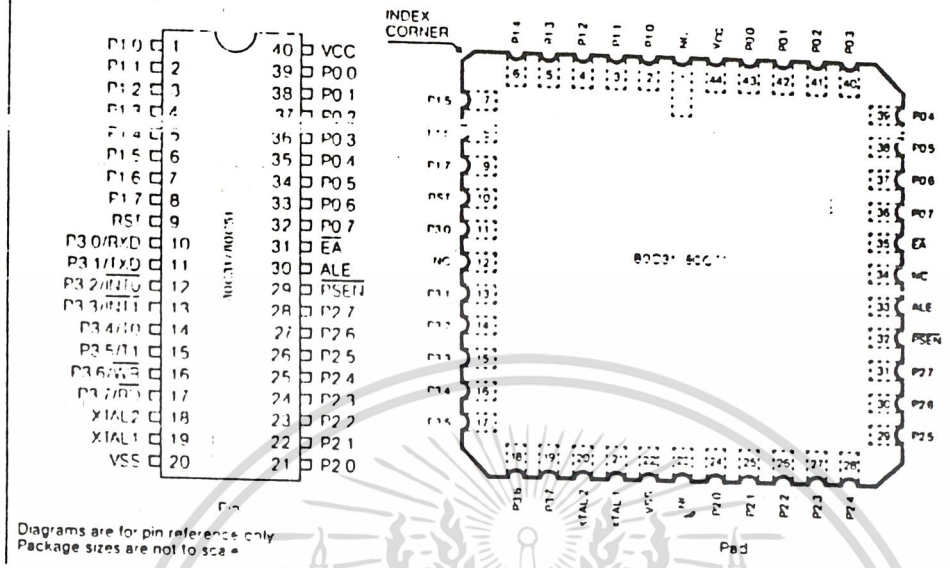
ลักษณะของ 8031

- เป็น CPU แบบ 8 บิท
- มีวงจรออสซิลเลเตอร์และ CLOCK อยู่ในตัว
- มีขาสัญญาณเข้าและออก (I/O) 32 ขา
- แยกหน่วยความจำของข้อมูลได้ 62K และหน่วยความจำของโปรแกรมอีก 64K
- มี TIMER และ COUNTER แบบ 16 บิท ถึง 2 ตัว
- สัญญาณอินเทอร์รัพท์ 6 แหล่ง 5 VECTOR ซึ่งแบ่งลำดับความสำคัญออกเป็น 2 ระดับ
- การทำงานแบบ FULL DUPLEX ในขณะส่งข้อมูลแบบอนุกรม
- มีการประมวลผลแบบบูลีน (AND, OR, XOR) ฯลฯ

การจัดการหน่วยความจำของ 8031

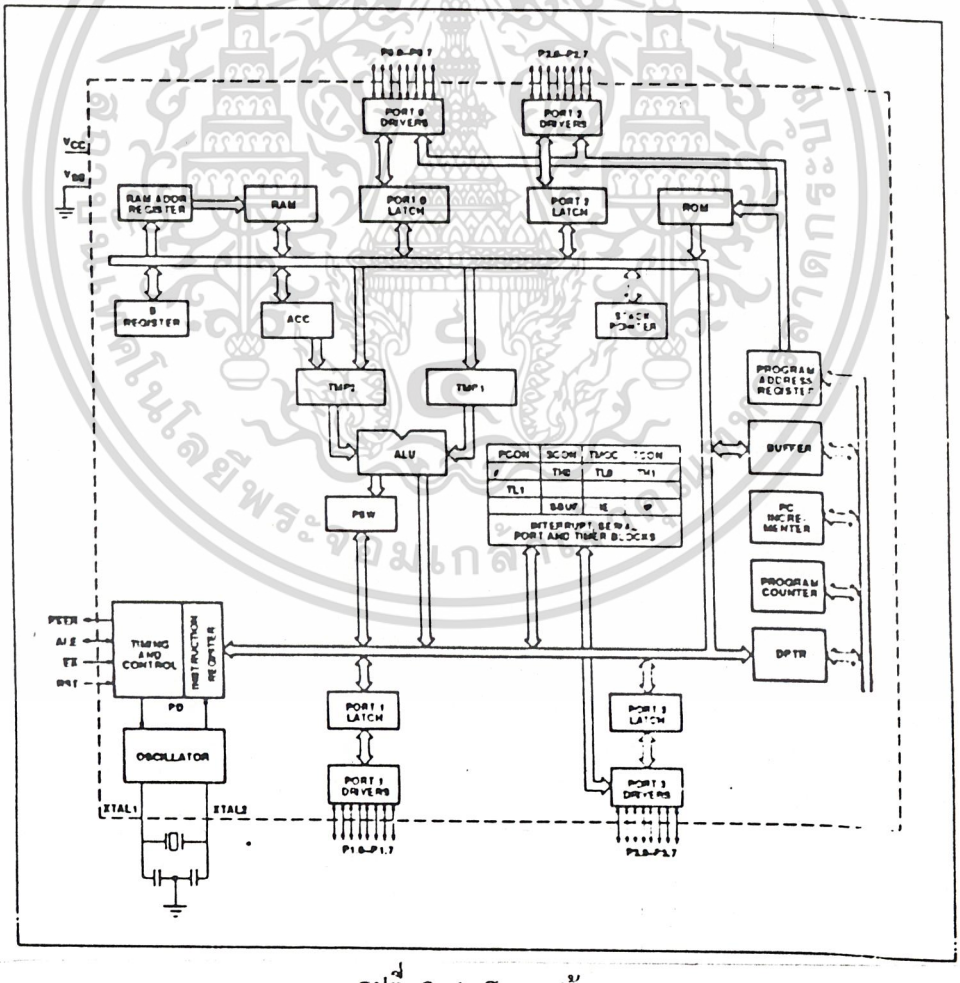
8031 แบ่งหน่วยความจำสำหรับโปรแกรมและข้อมูลอย่างละ 64K และยังมีหน่วยความจำประเภทแรมอีก 128 ไบท์ ซึ่งอยู่ในตัวของ 8031 และบน 8031 ยังประกอบด้วยรีจีสเตอร์ที่ทำหน้าที่พิเศษ "SFRs" (SPECIAL FUNCTION REGISTOR) ซึ่งแสดงในตารางที่ 1

Figure 2. Configurations



Diagrams are for pin reference only
Package sizes are not to scale

BLOCK DIAGRAM



รูปที่ 2.1 โครงสร้างของ 8031

*ตารางที่ 1.

SYMBOL	NAME	ADDRESS
* ACC	ACCUMULATOR	0E0H
* B	B REGISTOR	0F0H
* PSW	PROGRAM STATUS WORD	0D0H
SP	STACK POINTER	81H
DPTR	DATA POINTER (CONSIST OF DPH,PPL)	83H
DPL		82H
* P0	PORT 0	80H
* P1	PORT 1	90H
* P2	PORT 2	0A0H
* P3	PORT 3	0B0H
* IP	INTERUPT PIORITY CONTROL	0B8H
* IE	INTERUPT ENABLE CONTROL	0A8H
TMOD	TIMER/COUNTER MODE CONTROL	89H
+*T2CON	TIMER/COUNTER CONTROL	0C8H
TCON	TIMER/COUNTER 2 CONTROL	88H
TH0	TIMER0 (HIGH BYTE)	8CH
TL0	TIMER0 (LOW BYTE)	8AH
TH1	TIMER1 (HIGH BYTE)	8DH
TL1	TIMER1 (LOW BYTE)	8BH
+ TH2	TIMER2 (HIGH BYTE)	0CDH
+ TL2	TIMER2 (LOW BYTE)	0CCH
+ RCAP2H	TIMER/COUNTER2 CAPTURE (HI)	0CBH
+ RCAP2L	TIMER/COUNTER2 CAPTURE REG (LO)	0CAH
* SCON	SERIAL CONTROL	98H
SBUF	SERIAL DATA BUFER	99H
PCON	POWER CONTROL	87H

หมายเหตุ * SFRs ที่มีเครื่องหมายดอกจันทร์อยู่ข้างหน้า หมายความว่า สามารถเข้าถึงได้โดยการอ่านแอดเดรสแบบไบท์ และแบบบิท คือสามารถจะเข้าถึงบิทใดบิทหนึ่งใน SFRs ได้โดยตรง

+ SFRs ที่มีเครื่องหมายบวกแสดงว่ามีอยู่ใน 8032/8052 เท่านั้น

(MSB)							(LSB)
CY	AC	F0	RS1	RSO	0V	-	P

SYMBOL	POSITION	NAME AND SIGNIFICANCE
CY	PSW.7	CARRY FLAG
AC	PSW.6	AUXILLARY CARRY FLAG (FOR BCD OPERATION)
F0	PSW.5	FLAG 0 (FOR GENERAL PURPOSES)
RS1	PSW.4	REGISTOR BANK SELECT CONTROL BITS 1&0 SET/CLEARED BY SOLFWARE TO DETERMINE
RS0	PSW.3	WORKING REGISTOR BANK (SEE NOTE)
OV	PSW.2	OVERFLOW FLAG
	PSW.1	(RESERVED)
P	PSW.0	PARITY FLAG. SET/CLEARED BY HARDWARE EACH INSTRUCTION TO INDICATE AN ODD/EVEN NUMBER OF "ONE" BITS INTHE ACCUMULATOR., ie. EVEN PARITY.

NOTE - THE CONSISTS OF (RS1, RS0) ENABLE THE WORKING REGISTOR BANKS AS FOLLOWS.

(0.0) - BANK 0	(00H-07H)
(0.1) - BANK 1	(08H-0FH)
(1.0) - BANK 2	(10H-17H)
(1.1) - BANK 3	(18H-1FH)

รูปที่ 2.2 PROGRAM STATUS WORD

STACK POINTER

SP เป็นรีจีสเตอร์ขนาด 8 บิท เมื่อใช้คำสั่ง PUSH และ CALL SP จะเพิ่มขึ้นก่อนที่จะเก็บข้อมูลหรือแอดเดรส ซึ่ง STACK จะอยู่ที่ไหนก็ได้ในหน่วยความจำ (RAM) บน 8031 ภายหลังการ RESET แสตกจะมาอยู่ที่ 07H นั่นคือ แสตกจะเริ่มที่ 80H

DATA POINTER

DPTR ประกอบด้วย DPH และ DPL ซึ่งจะรวมกันเป็นรีจีสเตอร์ขนาด 16 บิท การเข้าถึง DPTR สามารถทำได้ทั้งแบบ 16 บิท และแบบ 8 บิท หน้าที่ของ DPTR ถ้าใช้แบบ 16 บิท จะทำหน้าที่เป็นตัวชี้ข้อมูลที่อยู่ในหน่วยความจำภายนอก

PORT 0-3

พอร์ต 0, พอร์ต 1, พอร์ต 2, และพอร์ต 3 เป็น SFR ตัวหนึ่งที่สามารถแลตช์ข้อมูลได้สามารถใช้เป็นอินพุทพอร์ต และเอ๊าท์พุทพอร์ตได้ทั้ง 3 พอร์ต

SERIAL DATA BUFFER

แบ่งเป็นรีจีสเตอร์บัฟเฟอร์ในทางส่งและบัฟเฟอร์ในทางรับ เมื่อมีการส่งข้อมูลภายในให้ SBUF ข้อมูลจะถูกส่งมาที่ บัฟเฟอร์ในทางส่ง ที่ซึ่งจะใช้ทำการส่งข้อมูลอนุกรม การส่งข้อมูลให้ SBUF จะเป็นการเริ่มต้นการส่งด้วย) และถ้าอ่านข้อมูลจาก SBUF ข้อมูลจะถูกอ่านจากบัฟเฟอร์ในทางรับ

TIMER REGISTER

รีจีสเตอร์คู่ (TH0,TL0) และ (TH1,TL1) เป็นตัวจับและตัวนับขนาด 16 บิทจะกล่าวรายละเอียดภายหลัง

CONTROL REGISTERS

รีจีสเตอร์หน้าที่พิเศษ IP, IE, TMOD, TCON, T2CON, SCON และ PCON ประกอบด้วยบิทควบคุมและบิทสถานะสำหรับระบบการอินเตอร์รัพท์

โครงสร้างและการทำงานของพอร์ต

8031 มี I/O พอร์ต อยู่ 4 พอร์ต โดยแต่ละพอร์ตจะเป็นพอร์ตแบบ 2 ทิศทาง มีการแลตช์ข้อมูลได้ (SFR.P0-P3) รวมทั้งมีวงจรบัฟทางด้านเอ๊าท์พุท และบัฟเฟอร์ทางด้านอินพุทพอร์ต 0 และ พอร์ต 2 ใช้สำหรับติดต่อกับหน่วยความจำภายนอก ในการใช้ 8031 ติดต่อกับหน่วยความจำภายนอก

พอร์ต 0 จะให้เอ๊าท์พุทเป็น LOW BYTE ของแอดเดรสของหน่วยความจำภายนอกและจะทำการ MULTIPLEX กับข้อมูลที่จะเขียนหรืออ่าน สรุปคือ P0 จะเป็นทั้ง ADDRESS และ DATA ส่วนพอร์ต 2 จะให้แอดเดรสไบท์สูง (MSB) ของหน่วยความจำภายนอก

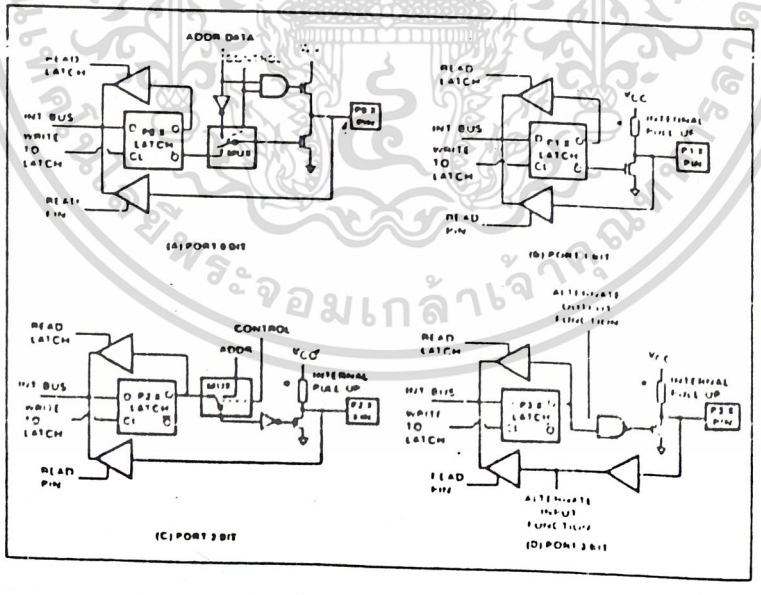
ขาของพอร์ต 3 ทั้งหมดกับอีก 2 บิทของพอร์ต 1 (8052) จะทำงานหลายหน้าที่

* P1.0	T2 (TIMER/COUNTER 2 EXTERNAL INPUT)
* P1.1	T2EX (TIMER/COUNTER 2 CAPTURE/RELOAD TRIGGER)
P3.0	RXD (SERIAL INPUT PORT)
P3.1	TXD (SERIAL OUTPUT PORT)
P3.2	INT0 (EXTERNAL INTERUPT)
P3.3	INT1 (EXTERNAL INTERUPT)
P3.4	T0 (TIMER/COUNTER 0 EXTERNAL INPUT)
P3.5	T1 (TIMER/COUNTER 1 EXTERNAL INPUT)
P3.6	WR (EXTERNAL DATA MEMORY WRITE STROBE)
P3.7	RD (EXTERNAL DATA MEMORY READ STROBE)

หมายเหตุ *p1.0 และ P1.1 สงวนไว้สำหรับ 8052

โครงสร้างของพอร์ต

ในรูปที่ 2.3 แสดงโครงสร้างของ I/O ของแต่ละพอร์ตโดยแสดงเพียงพอร์ตละบิทส่วนของพอร์ตแลตช์ใช้ D-FLIP FLOP ซึ่งจะรับข้อมูลมาจากบัสภายในโดยสัญญาณ "WRITE TO LATCH" ของ CPU ขา Q ของ D-FLIP FLOP จะถูกป้อนกลับไปที่บัสภายในเพื่อตอบสนองต่อสัญญาณ "READ LATCH" จาก CPU เมื่อ CPU ต้องการอ่านพอร์ต ส่วนสัญญาณที่ภายนอกของพอร์ตจะถูกต่อเข้าขาข้อมูลภายใน และขาพอร์ตภายนอกนี้จะถูกอ่านโดยตอบสนองต่อสัญญาณ "READ PIN" ซึ่งอ่านมาจาก CPU คำสั่งบางคำสั่งจะอ่านพอร์ตโดยอ่านจาก "READ LATCH" (ดูรูป 2.3 ประกอบ) ส่วนคำสั่งอื่น ๆ จะอ่านใช้สัญญาณ "READ PIN" ดูรายละเอียดคำสั่งที่อ่านข้อมูลจาก READ LATCH



รูปที่ 2.3 โครงสร้างแต่ละบิทของพอร์ต

ANL	(LOGIC AND, e.g., ANL P1, A)
ORI	(LOGIC OR, e.g., ORL P2, A)
XRL	(LOGIC EX-OR e.g., XRL P3, A)
JBC	(JUMP IF BIT = 1 AND CLEAR BIT, e.g., JBC P1.1, LABEL
CPL	(COMPLEMENT BIT, e.g., CPL P3.0)
INC	(INCREMENT, e.g., INC P2)
DEC	(DECREMENT, e.g., DEC P2)
DJNZ	(DECREMENT AND JUMP IF NOT ZERO, e.g., DJNZ P3, LABEL
MOV PX.Y,C	(MOVE CARRY BIT TO BIT Y OF PORT X)
CLR PX.Y	(CLEAR BIT Y OF PORT X)
SET PX.Y	(SET BIT Y OF PORT X)

เหตุที่บางคำสั่งต้องอ่านพอร์ตจาก "READ LATCH" เพราะในบางกรณี CPU อาจเข้าใจสัญญาณที่ขาของพอร์ตผิดพลาด ตัวอย่างเช่น PORT 1 ขาที่ 1 ถูกต่ออยู่กับขา BASE ของ TRANSISTOR และให้สัญญาณทางออกเป็น HIGH แก่ขา BASE ของทรานซิสเตอร์ ในขณะที่ทรานซิสเตอร์นำกระแสจะทำให้แรงดันที่ขา BASE เหลือโดยประมาณ 0.6 โวลท์ ซึ่งขา BASE ถูกต่ออยู่กับพอร์ตเมื่อ CPU ทำการอ่านพอร์ตจะทำให้เข้าใจสัญญาณที่ขาของพอร์ตผิดไปทั้ง ๆ ที่ขณะนั้นขาของพอร์ตเป็น HIGH อยู่

จากรูปที่ 2.3 จะเห็นว่าพอร์ต 0 นอกจากจะเป็น I/O พอร์ตแล้วยังเป็นได้ทั้งแอดเดรสบัสและบัสข้อมูล ส่วนพอร์ต 2 เป็นทั้ง I/O พอร์ตและแอดเดรส ซึ่งควบคุมโดยสัญญาณภายในชิพในขณะทำการติดต่อกับหน่วยความจำภายนอก SFR ของพอร์ต 2 จะไม่เปลี่ยนค่าหลังจากให้แอดเดรสแล้ว แต่ SFR ของพอร์ต 0 จะเปลี่ยนเป็น HIGH ทั้ง 8 บิท เพื่อที่จะรับข้อมูลที่จะส่งมาทางบัสข้อมูลของหน่วยความจำ (เนื่องจาก SFR สามารถแลตช์ข้อมูลได้ถ้าไม่ทำให้เป็น HIGH อาจเกิดการผิดพลาดในการอ่านข้อมูลจากหน่วยความจำภายนอก)

พอร์ต 1, พอร์ต 2 และพอร์ต 3 มีการพูลอัพภายใน ส่วนในพอร์ต 0 จะเป็นแบบ OPEN DRAIN OUTPUT แต่ในกรณีที่ใช้พอร์ต 1,2,3 เป็นอินพุท

จะต้องทำให้แต่ละบิทเป็น HIGH ก่อนเพื่อทำให้ FET ที่อยู่ทางด้านทางออกหยุดนำกระแสและรักษาระดับสัญญาณเป็น HIGH จากความต้านทานพูลอัพภายใน (ดูรูป 2.3 ประกอบ)

ข้อแตกต่างของพอร์ต 0 ก็คือ ไม่มีการพูลอัพภายใน (ดูรูป (2.3) ประกอบ) FET ตัวที่ทำหน้าที่พูลอัพ (ตัวบน) จะใช้เพียงให้เอ๊าท์พุทเป็น 1 ในขณะที่ติดต่อกับหน่วยความจำภายนอกในกรณีอื่น ๆ FET ตัวนี้จะถูกทำให้คัทออฟ ฉะนั้นพอร์ต 0 ที่ใช้เป็นเอาต์พุทพอร์ตจะเป็น OPENDRAIN

การเขียน 1(FFH) ไปที่พอร์ต 0 จะทำให้ FET ทั้ง 2 ตัวหยุดทำงานผลก็คือทำให้ขาพอร์ต 0 ลอยและสามารถใช้เป็นขาอินพุทแบบความต้านทานหน่วยสูง (HI-z INPUT) การรีเซ็ท 8031 จะทำให้ทุกพอร์ตมีระดับสัญญาณเป็น HIGH

การติดต่อกับหน่วยความจำภายนอก

การติดต่อกับหน่วยความจำภายนอกกระทำได้ 2 แบบ คือ ติดต่อกับโปรแกรมภายนอกกับติดต่อกับหน่วยความจำภายนอกที่เก็บข้อมูล การติดต่อกับโปรแกรมภายนอกจะใช้สัญญาณ PESN (PROGRAM STORE ENABLE) เป็นสัญญาณอ่านโปรแกรมภายนอก ส่วนการติดต่อกับหน่วยความจำภายนอกที่เก็บข้อมูลจะใช้ RD(P3.7) และ WR(P3.6) เหมือนกับ CPU ทั่วไป

การติดต่อกับข้อมูลภายนอกจะใช้ได้ทั้ง 16 บิท (MOVX @DPTR) หรือ เป็นแบบ 8 บิท (MOVX @RI) เมื่อไรก็ตามที่ต้องใช้แอดเดรสขนาด 16 บิท แอดเดรสไบท์สูง (A8-A15) จะออกทางพอร์ต 2

เมื่อติดต่อกับหน่วยความจำของข้อมูลภายนอกแบบ 8 บิท แอดเดรสไบท์สูง (A8-A15) จะออกทางพอร์ต 2

เมื่อติดต่อกับหน่วยความจำของข้อมูลภายนอกแบบ 8 บิท (MOVX A, @RI) จะไม่มีผลกระทบกับ SFR ของพอร์ต 2 คือ SFR ของพอร์ต 2 จะยังคงแลตช์ข้อมูลเดิมเอาไว้ ประโยชน์คือ สามารถใช้พอร์ต 2 กำหนดหน้า (PAGE) ของหน่วยความจำ

การที่จะต่อกับหน่วยความจำภายนอกได้ขึ้นอยู่กับสภาวะอยู่ 2 ประการ

1. เมื่อต่อขา EA ของ 8031 ลงกราวด์
2. เมื่อไรก็ตามที่โปรแกรมเค้าท์เตอร์ (PC) มีค่ามากกว่า OFFFH

สัญญาณ PSEN

เมื่อมีการแฟชคำสั่งจากโปรแกรมภายนอก สัญญาณ PSEN จะแอคทีฟ 2 ครั้งทุก ๆ 1 แมชีนไซเคิล (ยกเว้นคำสั่ง MOVX)

สัญญาณ ALE

จะเป็นขาโสตรปในการแลตช์แอดเดรสไบท์ต่ำ เมื่อทำการติดต่อกับหน่วยความจำภายนอก ALE จะแอคทีฟ 2 ครั้งทุก ๆ แมชีนไซเคิล ยกเว้นขณะทำคำสั่งที่ติดต่อกับข้อมูลในหน่วยความจำภายนอก (MOVX) ฉะนั้นในกรณีที่ระบบไม่ได้ใช้หน่วยความจำภายนอก (MOVX) การแอคทีฟของ ALE จะคงที่ในอัตรา 1/6 ของความถี่ออสซิลเลเตอร์ เพื่อใช้เป็น clock ให้กับวงจรภายนอกได้ การใช้งานบางกรณีต้องการที่จะใช้ทั้งโปรแกรมและข้อมูลที่อยู่ในเพจเดียวกัน (64 K) เราสามารถทำได้โดยการรวมสัญญาณ PSEN และ RD โดยใช้ AND GATE

การรีเซต

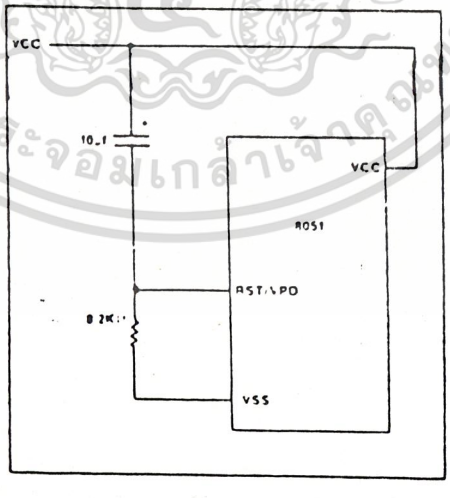
สัญญาณรีเซ็ตเป็นสัญญาณอินพุททางขา 9 การรีเซ็ทจะสมบูรณ์ต้องรักษาระดับ HIGH อย่างน้อยที่สุด 2 แมชีนไซเคิล (24 คาบเวลาของออสซิลเลเตอร์) การรีเซ็ทภายในตัว CPU จะเริ่มในระหว่างไซเคิลที่ 2 นับตั้งแต่ขา RST เป็น HIGH ผลของการรีเซ็ทจะมีผลกับรีจิสเตอร์ ดังต่อไปนี้

การรีเซตเมื่อเปิดเครื่อง

เมื่อจ่ายไฟเข้าระบบควรจะมีการรีเซท CPU ก่อนเพื่อรอให้ทั้งระบบ

อยู่ในสภาวะพร้อมที่จะทำงานซึ่งทำได้โดยต่อ C ขนาด 10uF จาก VCC มาที่ขา 9(RST) และจากขา 9 ต่อ R ขนาด 8.2K ลงกราวด์

REGISTOR	CONTENT
PC	0000H
ACC	00H
B	00H
PSW	00H
SP	07H
DPTR	0000H
P0-P3	FFH
IP	(XX000000)
IE	(0X000000)
TMOD	00H
TCON	00H
TH0	00H
TL0	00H
TH1	00H
TL1	00H
SCON	00H
SBUF	00H
PCON	00H



รูปที่ 2.4 POWER ON RESET CIRCUIT

IDLE MODE

8031 มีคำสั่งเซ็ท PCON.0 ซึ่ง CPU ทำคำสั่งนี้จะเป็นคำสุดท้ายก่อนจะเข้าสู่ IDLE โหมด (ไม่ทำงาน) IDLE โหมดนี้สัญญาณนาฬิกาภายในจะหยุดส่งให้ CPU ถูกเก็บไว้และรีจีสเตอร์ต่าง ๆ ยังคงรักษาข้อมูลเดิมไว้ในระหว่างอยู่ใน IDLE โหมด การที่จะทำให้ CPU กลับสู่โหมดปกติทำได้ 2 อย่าง คือทำการอินเตอร์รัพท์จากภายนอกเมื่อ CPU ถูกอินเตอร์รัพท์จะให้บริการอินเตอร์รัพท์เมื่อจบบริการอินเตอร์รัพท์ CPU จะกลับมาทำคำสั่งต่อไปของโปรแกรมซึ่งต่อจากคำสั่งที่ทำ IDLE โหมด การประยุกต์ใช้งานเราอาจใช้ประโยชน์จาก GF1, GF0 ซึ่งเป็นแฟลกที่ใช้ในจุดประสงค์ทั่ว ๆ ไปโดยการเซ็ทบิทใดบิทหนึ่งหรือทั้งสองก่อนคำสั่ง IDLE โหมด และเมื่อถูกอินเตอร์รัพท์ในโปรแกรมบริการอินเตอร์รัพท์ตรวจดูว่าเป็นการอินเตอร์รัพท์หลังจากทำ คำสั่ง IDLE โหมดหรือไม่เพื่อที่จะให้บริการที่ถูกต้อง อีกวิธีหนึ่งที่จะออกจาก IDLE โหมดได้คือ ทำการรีเซ็ท CPU ทางฮาร์ดเซ็ท

รายละเอียดของขาต่าง ๆ

VCC : ต่อกับไฟเลี้ยงของระบบ (5VDC)

VSS : กราวด์

PORT0 : พอร์ต 0 เป็นอินพุท/เอาท์พุทพอร์ทแบบ 8 บิท (OPEN DRAIN) สามารถรับกระแส SINK จาก LSTTLK ได้ 8 ตัว ถ้าเราเขียน 1 ไปที่พอร์ต 0 จะทำให้พอร์ต 0 เป็น H1-Z อินพุท พอร์ต 0 สามารถ MULTIPLEX ระหว่างบัสข้อมูลกับแอดเดรสไบท์ต่ำเมื่อใช้ติดต่อกับหน่วยความจำภายนอก

PORT1 : พอร์ต 1 เป็น 8 บิทสองทิศทางมีการพูลอัพภายใน, พอร์ต 1 นี้ขับ TTL(LS) ได้ 4 ตัว เมื่อเขียนค่า 1H ไปที่พอร์ต 1 ขาของพอร์ต 1

จะเป็น HIGH โดยมีการพูลอัพภายในและสามารถใช้เป็นอินพุทพอร์ตได้

PORT2 : เป็นอินพุท/เอ๊าท์พุทพอร์ตแบบ 2 ทิศทางมีการพูลอัพภายในสามารถขับ TTL(LS) ได้ 4 ตัว และพอร์ต 2 นี้จะให้แอดเดรสไบท์สูงในขณะติดต่อกับหน่วยความจำภายนอก

PORT3 : พอร์ต 3 เป็นอินพุท/เอ๊าพุทพอร์ตแบบ 2 ทิศทางมีการพูลอัพภายในพอร์ต 3 ขับ TT:(LS) ได้ 4 ตัวและยังสามารถใช้ทำงานในลักษณะพิเศษดังรายละเอียดข้างล่าง

PORT PIN	ALTERNATE FUNCTION
P3.0	RXD (SERIAL INPUT PORT)
P3.1	TXD (SERIAL OUTUT PORT)
P3.2	INTO (EXTERNAL INTERUPTO)
P3.3	INT1 (WXTERNAL INTERUPT1)
P3.4	TO(TIMER 0 EXTERNAL INPUT)
P3.5	T1 (TIMER 1 EXTERNAL INPUT)
P3.6	WR (EXTERNAL DATA MEMORY WRITE STROBE)
P3.7	RD (EXTERNAL DATA MEMORY READ STORBE)

RST : ขารีเซ็ทสัญญาณ HIGH ที่ขารีเซ็ทนี้นาน 2 แมชีนไซเคิลจะเป็นการรีเซ็ท CPU

ALE/PROG : ADDRESS LATCH ENABLE พัลส์สำหรับแลตช์แอดเดรสไบท์ต่ำในระหว่างการติดต่อกับหน่วยความจำภายนอก ALE นี้ยังจ่ายความถี่คงที่ 1/6 ของความถี่ออสซิลเลเตอร์

(ถ้าไม่ติดต่อกับหน่วยความจำภายนอก) และ ALE อีกหน้าที่หนึ่งคือเป็นอินพุทรับพัลส์ (LOW) ในระหว่างโปรแกรม EPROM (8451)

PSEN : เป็นขาสัญญาณ READ STROBE ในขณะอ่านโปรแกรมจากภายนอก PSEN จะแอคทีฟ 2 ครั้งต่อ 1 แมชีนไซเคิล

EA/UPP : ต่อ LOW จะทำคำสั่งในโปรแกรมภายนอก ถ้าต่อ HIGH จะใช้โปรแกรมภายใน และใช้เป็นอินพุทรับ 21V. ในการโปรแกรม 8751H

XTAL1 : เป็นขาอินพุทของอินเวอร์เตอร์ของภาคขยายความถี่

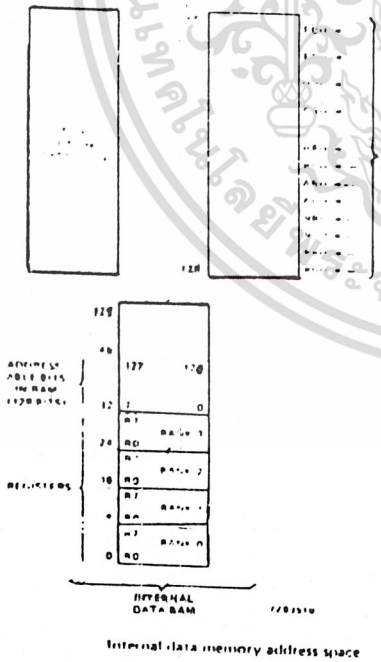
XTAL2 : เป็นขาเอ๊าท์พุทของอินเวอร์เตอร์ของภาคขยายความถี่

<u>การจัดหน่วยความจำ</u>

โปรแกรมเมมโมรี : ไมโครคอนโทรเลอร์ตระกูล MCS-51 จะแบ่งหน่วยความจำเป็นส่วนของโปรแกรมเมมโมรีและส่วนของข้อมูล (DATA) อย่างละ 64K ถ้า EA ต่อ HIGH จะเป็นการรันโปรแกรม ที่อยู่ภายในตัวโดยแอดเดรสจะไม่เกิน OFFH และถ้า EA ต่อ LOW จะเป็นการรันโปรแกรมภายนอกที่แอดเดรส 00 ถึง 23H จะเป็นส่วนเกินของการอินเตอร์รัพท์

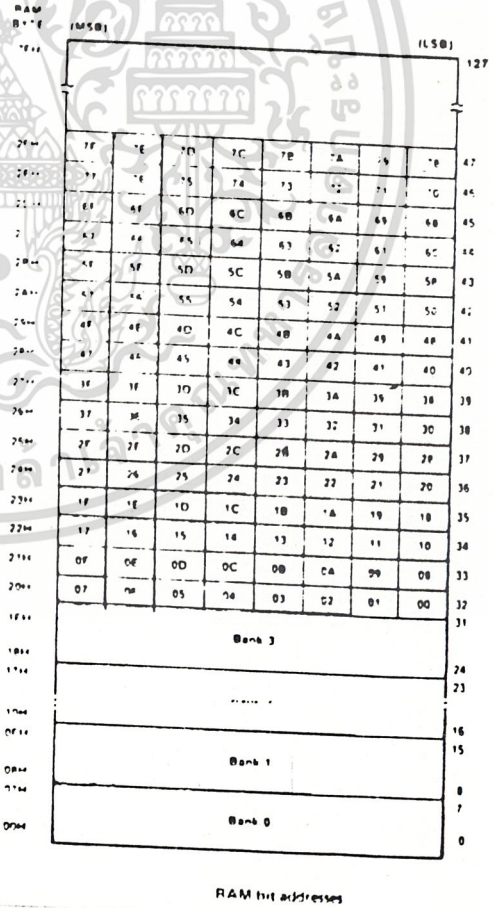
หน่วยความจำเก็บข้อมูล

แอดเดรสของข้อมูลจะประกอบด้วยหน่วยความจำภายในและภายนอก โดยที่หน่วยความจำสำหรับข้อมูลภายนอกติดต่อได้โดยคำสั่ง MOVX และมี DPTR เป็นตัวชี้หน่วยวามจำภายในถูกแบ่งเป็น 3 ส่วน คือ แอดเดรสด้านต่ำ 128 ไบท์ (RAM) และ SFR(APECIAL FUNCTION REGISTER) ซึ่งแท้ที่จริงก็คือ RAM อีก 128 ไบท์นั่นเอง แต่การเข้าถึงหน่วยความจำใน SFR นี้แตกต่างจาก 128 ไบท์ล่างหน่วยความจำทางด้านต่ำ (LOWER RAM 0-31H) ยังแบ่งเป็น 4 BANK โดยที่สามารถเข้าถึงได้ครั้งละ 1 BANK เท่านั้นแต่ละ BANK เลือกโดยเซ็ทบิท R0)R1 ใน PSW หน่วยความจำอีก 16 ไบท์ ตำแหน่ง 20H ถึง 47H ประกอบด้วยหน่วยความจำที่สามารถเข้าถึงแบบบิทได้อีก 128 บิท



Internal data memory address space

รูปที่ 2.5



RAM bit addresses

รูปที่ 2.6

ADDRESSING MODES

8031 แบ่งการเข้าถึงหน่วยความจำได้ 5 แบบ ดังรูปที่ 17

1.REGISTER ADDRESSING

- R0-R7
- ACC, B, CY(BIT), DPTR

2.DIRECT ADDRESSING

- LOWER 128 BYTES OF INTERNAL RAM
- SPECIAL FUNCTION REGISTER

3.REGISTER INDIRECT ADDRESSING

- INTERNAL RAM (@RI,@RO,SP)
- EXTERNAL DATA MEMORY (@R1,RO,@DPTR)

4.IMMEDIATE ADDRESSING

- PROGRAM MEMORY

5.BASE-REGISTER PLUS INDEX-REGISTER INDIRECT ADDRESS

- PROGRAM MEMORY (@DPTR+A,@PC+A)

REGISTER ADDRESSING

เป็นการติดต่อกับรีจีสเตอร์ทั้ง 8 ตัวในแต่ละ BANK โดยใช้บิทล่างของ OP-CODE เป็นตัวกำหนดรีจีสเตอร์ที่จะทำการติดต่อด้วย ACC, B, DPTR และ CY และการประมวลผลทางบูลีนถือว่าอยู่ในโหมด REGISTER ADDRESING ตัวอย่างเช่น MOV A, R0

DIRECT ADDRESS

เป็นเพียงวิธีเดียวที่จะเข้าถึง SFR(SPECIAL FUNCTION REGISTER) ได้ติดต่อกับ RAM 128 ไบท์ล่างก็ใช้โหมดนี้ด้วย ตัวอย่างเช่น MOV A, SBUF

REGISTER-INDIRECT ADDRESSING

ในโหมดนี้ใช้ค่าที่อยู่ในรีจีสเตอร์ R0 หรือ R1 (ใน BANK ที่เลือกไว้) เป็นตัวชี้ไปยังตำแหน่งต่าง ๆ ภายใน 256 ไบท์ (RAM 128 ไบท์ล่าง หรือ 256 ไบท์ล่างของหน่วยความจำภายนอกข้อสังเกต คือ SFR ไม่สามารถติดต่อได้ด้วยวิธี REGISTER-INDIRECT นี้ ส่วนการติดต่อกับหน่วยความจำข้อมูลภายนอก 64K นั้นใช้ตัวชี้ขนาด 16 บิท(DPTR) คำสั่ง PUSH และ POP ก็ทำงานในโหมดนี้ด้วย (ใช้ SP เป็นตัวชี้) ตัวอย่าง MOV A,@R0

IMMEDIATE ADDRESSING

โหมดนี้อนุญาติให้ใช้ค่าคงที่เป็นส่วนหนึ่งของ OP-CODE

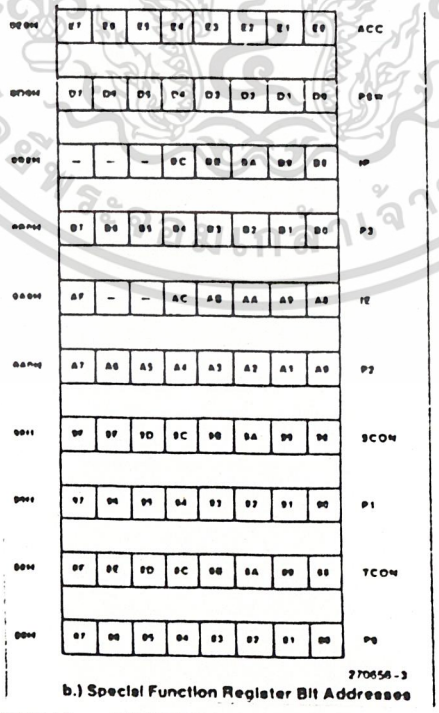
ตัวอย่าง MOV A,#01H

MOV DPTR,#1234H

BASE-REGISTER PLUS INDEX REGISTER-INDIRECT ADDRESSING

ในโหมดนี้จะใช้ค่าในแอคคิวมูเลเตอร์ A บวกกับเบสรีจีสเตอร์ เช่น DPT หรือ PC ประโยชน์ของโหมดนี้คือใช้ในการหาค่าที่ต้องการจากตาราง(TABLE)

ตัวอย่าง MOVC A,@A+PC



รูปที่ 2.7 SPCEIAL, FUNCTION REGISTOR BIT ADDRESS

ชุดคำสั่งของ 8031

8031 มีคำสั่งทั้งหมด 111 ประกอบด้วยคำสั่งไบท์เดียว 49 คำสั่ง คำสั่ง 2 ไบท์ 45 คำสั่ง คำสั่ง 3 ไบท์ 17 คำสั่ง เราแบ่งคำสั่งตามหน้าที่การทำงานได้เป็น 4 แบบ

1. DATA TRANSFER
2. ARITHMETIC
3. LOGIC
4. CONTROL

1.1 DATA TRANSFER ยังแบ่งเป็น

- GENERAL PUPOSE
- ACCUMULATOR-SPECFIC
- ADDRESS-OBJECT

ทั้งหมดนี้ไม่มีคำสั่งใดที่มีผลกระทบต่อ PSW ยกเว้นการ POP หรือ MOV โดยตรงไปที่ PSW

GENERAL PURPOSE TRANSFER

- MOV จะกระทำเป็นบิทหรือไบท์ก็ได้โดยข้อมูลจะย้ายจากต้นทาง (SOURCE) มายังปลายทาง

- PUSH คำสั่งนี้จะเพิ่มค่า SP ขึ้นไปอีก 1 ค่าก่อนที่จะนำข้อมูลจากต้นทางไปเก็บไว้ในตำแหน่งที่ชี้โดย SP

- POP คำสั่งย้ายข้อมูลจากตำแหน่งที่ชี้โดย SP มายังปลายทางและ SP จะลดค่าลง 1

ACCUMULATOR SPECIFIC TRANSFER

- XCH(EXCHANGE) แลกเปลี่ยนข้อมูลของต้นทางกับแอคคิวมูเลเตอร์

- XCHD แลกเปลี่ยนค่า 4 บิทล่างของไบท์ข้อมูลกับข้อมูล 4 บิทล่าง

ของแอคคิวมูเลเตอร์

- MOVEX ทำการย้ายข้อมูลระหว่างแอคคิวมูเลเตอร์กับหน่วยความจำข้อมูลภายในออก โดยที่ตำแหน่งของหน่วยความจำภายนอกกำหนดได้โดยค่าของ DPTR(16 บิท) หรือรีจีสเตอร์ R0,R1 (8 บิท)

- MOVC ย้ายข้อมูล 1 ไบท์จากหน่วยความจำของโปรแกรม (ในกรณีแยกหน่วยความจำของข้อมูลต่างหาก) มาไว้ที่ A โดยข้อมูลใน A ก่อนทำคำสั่งนี้จะรวมกับ PC หรือ DPTR เป็นตัวชี้ที่อยู่ของข้อมูลที่ต้องการ

ตัวอย่าง

```
REL-PC:  INC A
         MOVC A,@A+PC
         RET
         DB 66H
         DB 77H
         DB 88H
         DB 89H
```

ถ้า SUBROUTINE นี้ถูกเรียกโดยที่ A มีค่า = 01H เมื่อออกจากรูทีนจะได้ค่า 77H อยู่ใน A

ARITHMETIC

8031 มีคำสั่งสนับสนุนการคำนวณเบื้องต้น เช่น บวก ลบ คูณ หาร แบบ 8 บิท โดยไม่คิดเครื่องหมาย

การบวก (ADDING)

- INC(INCREMENT) เพิ่มค่าในโอเปอร์แรนด์ขึ้นอีก 1
- ADD บวกค่าของโอเปอร์แรนด์คันทางกับแอคคิวมูเลเตอร์และเก็บค่าไว้ในแอคคิวมูเลเตอร์

- DA(DECIMAL-ADD-ADJUST FOR BCE ADDITION) เป็นการปรับแต่งผลรวมของการบวกเลข BCD และคืนผลลัพธ์นั้นให้รีจีสเตอร์ A แฟลก CY จะถูกเซ็ท ถ้าผลลัพธ์ที่ได้จากคำสั่ง DA มากกว่า 99 การทำงานคร่าว ๆ ของคำสั่งถ้าค่าที่จะทำการปรับแต่ง 4 บิทล้างของ A มากกว่า 9 หรือแฟลก AC=1 จะทำการบวกด้วย 06H เช่นเดียวกัน ถ้า 4 บิทบนมากกว่า 9 ก็จะบวกด้วย 06H

การลบ (SUBTRACTION)

- SUBB(SUBTRACT WITH BORROW) ลบค่าในโอเปอร์แรนด์ตัวแรกด้วยโอเปอร์แรนด์ตัวที่ 2 และลบด้วย 1 ถ้า CY ถูก SET และคืนผลลัพธ์ให้แอคคิวมูเลเตอร์ A

- DEC(DECREMENT) ลดค่าในแอคคิวมูเลเตอร์ลง 1

การคูณ (MULTIPLICATION)

- MUL เป็นการคูณกันระหว่างรีจีสเตอร์ A กับ B โดยที่ A จะรับค่าไบท์ต่ำและ B รับค่าไบท์สูงของผลการคูณ OV จะถูกเคลียร์ถ้าข้อมูลครึ่งบนสุดของผลลัพธ์เป็น 0 ส่วน CY จะเคลียร์เสมอ

การหาร (DIVISION)

- DIV เป็นการหารระหว่างรีจีสเตอร์ A โดยรีจีสเตอร์ B และให้ผลลัพธ์เลขจำนวนเต็มในรีจีสเตอร์ A ส่วนเศษของการหารจะอยู่ในรีจีสเตอร์ B การหารด้วยศูนย์จะไม่ให้ค่าผลลัพธ์ใน A ส่วนเศษของการหารจะอยู่ในรีจีสเตอร์ B การหารด้วยศูนย์จะไม่ให้ค่าผลลัพธ์ใน A และ B และ OV แฟลกจะถูกเซ็ท การทำคำสั่งหารนี้จะมีผลต่อแฟลกต่าง ๆ ดังนี้

- CY จะถูกเซ็ทถ้าผลของการหารทำให้เกิดตัวทดขึ้นจากบิทสูงสุด

- AC จะถูกเซ็ทถ้าผลของการหารทำให้เกิดตัวทดจาก 4 บิทล้างหรือเกิดการยืมบิทบนโดยบิทล่าง

- OV จะถูกเซ็ทถ้าจำนวนใด ๆ ถูกหารด้วย 0 ในกรณีอื่น OV จะถูกเคลียร์ OV ถูกใช้ในการคำนวณแบบ TWO'S COMPLEMENT เพราะว่า OV จะถูกเซ็ทเมื่อมีการใช้จำนวนที่มีเครื่องหมายและไม่สามารถแสดงผลใน 8 บิทได้

- P(PARITY) ถ้าผลลัพธ์ทำให้เกิดพาริตี้คี่ P จะถูกเซ็ท

LOGIC

8031 สามารถปฏิบัติการทาง LOGIC ได้ทั้งแบบบิทและไบท์

SINGLE-OPERAND OPERATION

- CLR เคลียร์รีจีสเตอร์ A ให้เป็น 0 หรือเคลียร์บิทใดบิทหนึ่งในรีจีสเตอร์หรือหน่วยความจำที่สามารถเข้าถึงแบบบิทให้เป็น 0 ได้

- SETB เซ็ทบิทต่าง ๆ ถ้าสามารถเข้าถึงแบบบิทได้เป็น 1

- CPL คอมพลีเมนท์ค่าให้รีจีสเตอร์ A หรือคอมพลีเมนท์โดยตรงกับบิทใดบิทหนึ่งที่สามารถติดต่อแบบบิทได้

- RL,RLC,RR,RRC,SWAP เป็นกลุ่มคำสั่งหมุน (ROTATE) ค่าในรีจีสเตอร์ A SWAP ใช้เปลี่ยนค่ากันระหว่าง 4 บิทล่างกับ 4 บิทบนของรีจีสเตอร์

TWO OPERAND OPERATIONS

- ANL คำสั่ง LOGIC "AND" แบบบิทหรือไบท์ด้วยรีจีสเตอร์ 2 ตัว โดยผลลัพธ์เก็บไว้ในรีจีสเตอร์ตัวแรก

- ORL คำสั่ง LOGIC "OR" แบบบิทหรือไบท์โดยรีจีสเตอร์ 2 ตัว

- XOR คำสั่ง LOGIC "XOR"

UNCONDITIONAL CALLS, RETURN AND JUMP

- ACALL และ LACLL เก็บค่า PC ของคำสั่งต่อไปลงในแสตคและเปลี่ยนการควบคุมระบบให้กับแอดเดรสปลายทางของคำสั่ง CALL

- LCALL เป็นคำสั่ง 3 ไบท์ ที่สามารถกระโดดไปยังตำแหน่งใด ๆ ภายใน 64K ได้

- ACALL เป็นคำสั่ง 2 ไบท์ ใช้เมื่อการ CALL ไปยังตำแหน่งที่ไม่ไกลเกิน 2K โดยการนำค่าของ A0-A10 ของตำแหน่งที่จะกระโดดไปมาเข้าโค๊ดในคำสั่ง ACALL

A10 A9 A8 1	0001	A7 A6 A5 A4	A3 A2 A1 A0

ข้อควรระวัง ถ้าคำสั่ง AJMP อยู่ใน 2 ไบท์สุดท้ายของ PAGE(2K) เมื่อทำคำสั่งนี้ PAGE จะเปลี่ยนไปเพราะว่า PC จะต้องเพิ่มขึ้นอีก 2 ไบท์ (AJMP=2 ไบท์ ก่อนที่จะทำคำสั่งทำให้การกระโดดไม่ตรงตำแหน่งที่ต้องการ เพราะในคำสั่ง AJMP จะเกี่ยวพันกัน A11-A15 ด้วยในกรณีนี้ A11 จะเปลี่ยนไป

- RET กลับสู่โปรแกรมหลักหลังจากการ CALL โดยการ POP ไบท์สูงให้ PC แล้วลดค่า SP ลง 1 POP ไบท์ต่ำให้ PC และลดค่า SP ลงอีก 1

- AJMP LJMP และ SJMP คำสั่ง AJMP และ LJMP คล้ายกัน ACALL และ LCALL ส่วน SJMP(SHORT JUMP จะกระโดดได้เพียง 256 ไบท์ โดยมีจุดศูนย์กลางอยู่ที่คำสั่งต่อจาก SJMP(-128 TO+127)

- JMP @A+DPTR ใช้รีจิสเตอร์ A เป็นตัว OFFSET(0-255) โดยการบวกค่าในรีจิสเตอร์ A กับ DPTR

DONDITIONAL JUMP

เป็นการกระโดดแบบมีเงื่อนไขโดยระยะทางในการกระโดดอยู่ในช่วง 256 ไบท์ โดยเดินหน้าและถอยหลังจากจุดศูนย์กลางอยู่ที่คำสั่งที่ต่อจากการกระโดดนั้น (-128 TO +127)

- JZ กระโดดถ้าแอคคิวมูเลเตอร์เป็นศูนย์
- JNZ กระโดดถ้าแอคคิวมูเลเตอร์ไปเท่ากับศูนย์
- JC กระโดดถ้าแฟลกตัวทดถูกรีเซท
- JNC กระโดดถ้าแฟลกตัวทดไม่ถูกรีเซท
- JB กระโดดถ้าบิทที่กำหนดถูกรีเซท
- JNB กระโดดถ้าบิทที่กำหนดไม่ถูกรีเซท
- JBC กระโดดถ้าบิทที่กำหนดถูกรีเซ็ทและทำการเคลียร์บิทที่กำหนดนั้นด้วย
- CJNE ทำการเปรียบเทียบโอเปอร์แรนด์ตัวแรกกับตัวที่ 2 และจะกระโดดถ้าโอเปอร์แรนด์ทั้งสองมีค่าไม่เท่ากัน แฟลก CY จะถูกรีเซ็ทถ้าโอเปอร์แรนด์ตัวแรกมีค่าน้อยกว่าตัวที่ 2
- DJNZ ลดค่าในโอเปอร์แรนด์และกระโดดถ้าผลของการลดค่านั้นไม่เท่ากับศูนย์

การใช้ DOT MATRIX LCD MODULE

อุปกรณ์ในปัจจุบันนี้ในส่วนแสดงผลนั้นจะใช้ LCD เสียเป็นส่วนใหญ่ ไม่ว่าจะเป็นเครื่องเล่น VIDIO, เครื่องถ่ายเอกสาร, เครื่องมือวัดคุมต่างๆ เครื่องคอมพิวเตอร์ เราพอจะแบ่ง DOT MATRIX LCD MODULE นี้ออกได้เป็นพวกๆดังนี้ :-

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

โดยในแต่ละแบบนี้ก็จะมีส่วนประกอบใหญ่ๆแบ่งได้เป็น

1. DOT MATRIX LCD เป็นตัวแสดงผลให้เรามองเห็นในลักษณะการปิดและเปิดตัวเองกับแสงก็คือ ส่วนของที่เป็นตัวกระจกบรรจุผลึก

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

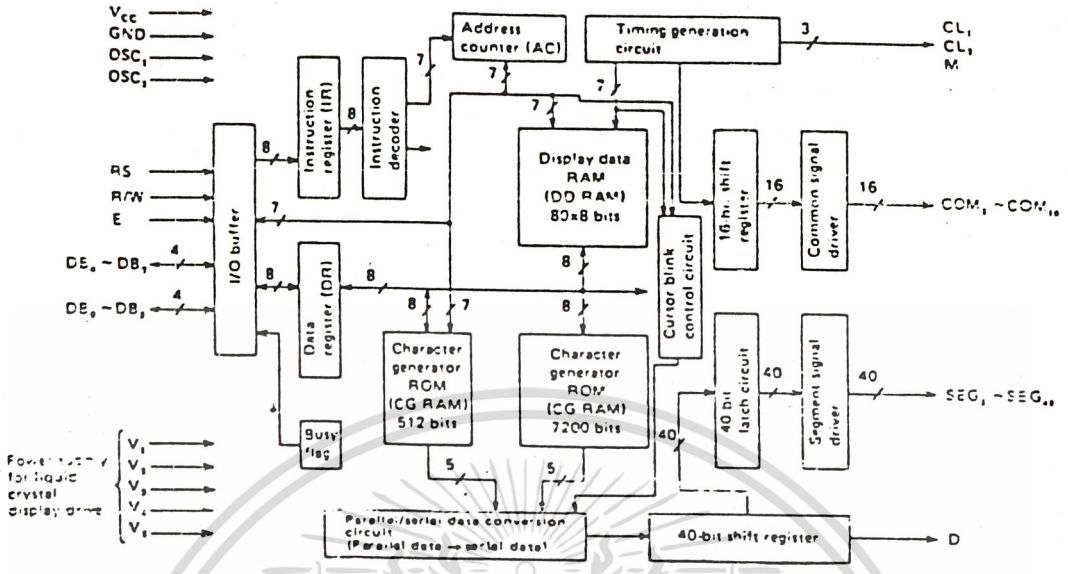
3. CONTROLLERเป็นตัวรับข้อมูลจากอุปกรณ์ภานนอกมาและจัดการควบคุม LCD MODULEDDD ให้ทำงานแสดงผลต่างๆ เช่น การลบจอภาพ,การเกิดตัวอักษร เป็นต้น โดยมีเบอร์ IC ที่นิยมใช้กันคือ HD44780 ซึ่งจะใช้ในแบบ CHARACTER LCD MODULE เป็นส่วนใหญ่ เบอร์ IC HD61830 จะใช้ในแบบ GRAPHIC LCD MODULE

ในการศึกษาการทำงานและใช้งาน LCD MODULE นั้นไม่ใช่เรื่องยากเลยถ้าเราสามารถทำความเข้าใจในส่วนของ CONTROLLER ได้ก็เพียงพอแล้วและโดยมาก LCD MODULE ในแต่ละบริษัทแล้วจะใช้ตัว CONTROLLER ที่มีหลักการทำงานเหมือนๆกันเป็นส่วนใหญ่และใน LCD MODULE แต่ละขนาดจำนวนตัวอักษรหรือจำนวนบรรทัดก็มีหลักการทำงานแบบเดียวกันทั้งหมด IC

ที่นิยมมากที่สุดตัวหนึ่งที่เป็น CONTROLLER LCD ก็คือ เบอร์ HD44780 โดยรูปแบบการทำงานของมันได้เป็นมาตรฐานให้กับ CONTROLLER LCD ตัวอื่นๆ ด้วย

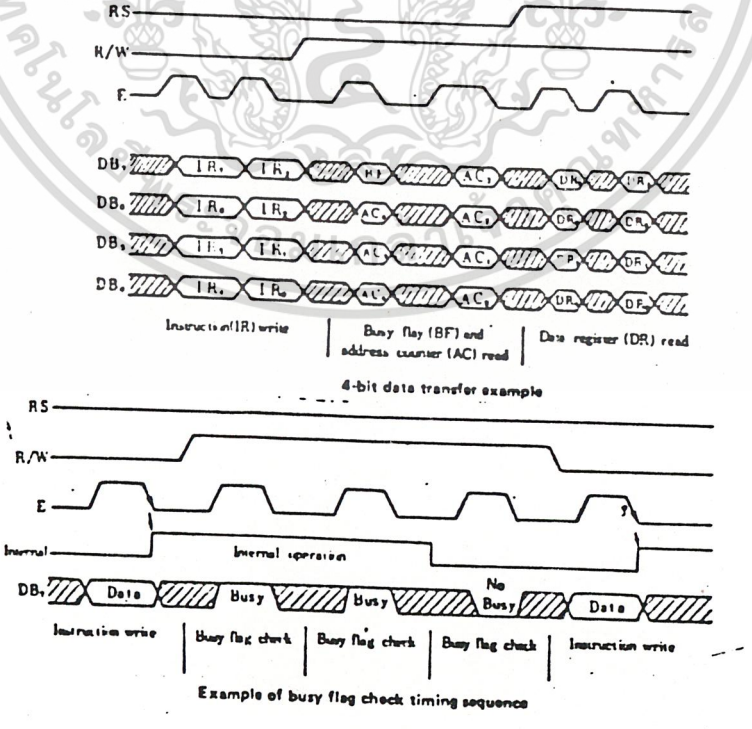
HD44780 เป็นไอซี LSI ตัวหนึ่งใช้ควบคุม LCD โดยแสดงผลในรูปตัวอักษรหรือสัญลักษณ์ต่างๆตัวมันเองสามารถต่อใช้แบบ 4 BIT หรือ 8 BIT ก็ได้ โดยถ้าเราต่อแบบ 4 BIT จะต่อใช้งานที่ DB7-DB4

Block diagram of HD44780 interior

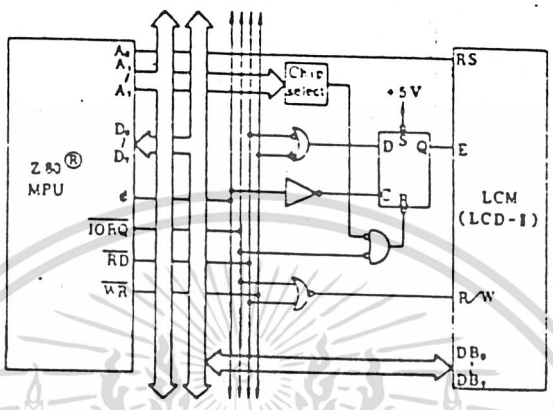


รูปที่ 2.8 Block diagram of HD47780 interior

เท่านั้นโดยข้อมูลครั้งแรกที่ส่งนั้น HD44780 จะถือเป็นข้อมูล 4BIT บนและข้อมูลที่ส่งมาต่อมานั้นเป็นข้อมูล 4 BIT ล่าง

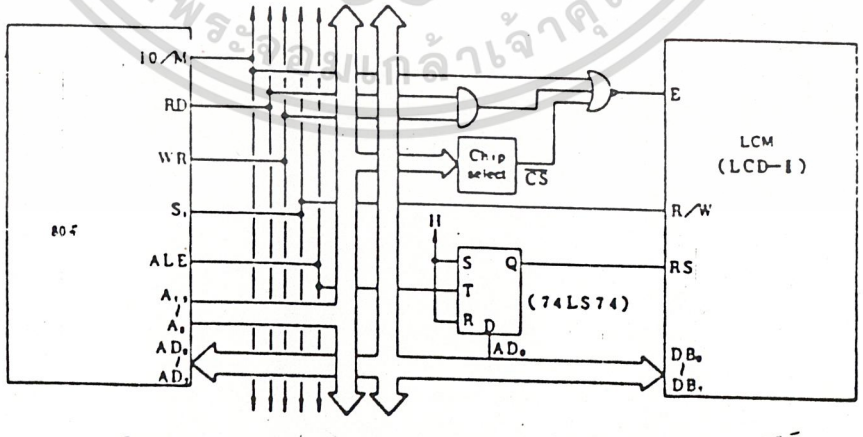


รูปที่ 2.9 Example of busy flag check timing sequence



รูปที่ 2.10 Example of interfacing to Z80 MPU

เราสามารถต่อ LCD MODULE (HD44780 เป็น CONTROLER) เข้ากับระบบไมโครได้ หลายรูปแบบดังรูปที่ 2.11 และ 2.12



Example of connection with LCM being used as a part of memories on the determined address.

รูปที่ 2.11 Example of connection with LCM

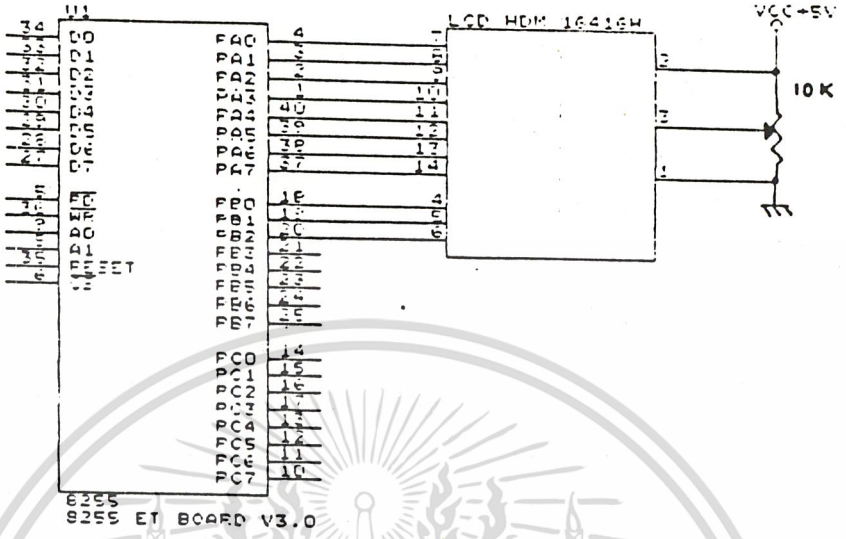
ตารางคำสั่ง HD44780

Instruction	Code										Description	Execution time (when fosc is 250 kHz) Note 1	Execution time (when fosc is 160 kHz) Note 2
	RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0			
Clear display	0	0	0	0	0	0	0	0	0	1	Clears all display and returns the cursor to the home position (Address 0).	82 μs ~ 1.64 ms	120 μs ~ 4.9 ms
Return home	0	0	0	0	0	0	0	0	1	*	Returns the cursor to the home position (Address 0). Also returns the display being shifted to the original position. DD RAM contents remain unchanged.	40 μs ~ 1.6 ms	120 μs ~ 4.8 ms
Entry mode set	0	0	0	0	0	0	0	1	I/D	S	Sets the cursor move direction and specifies or not to shift the display. These operations are performed during data write and read.	40 μs	120 μs
Display ON/ OFF control	0	0	0	0	0	0	1	D	C	B	Sets ON/OFF of all display (D), cursor ON/OFF (C), and blink of cursor position character (B).	40 μs	120 μs
Cursor and display shift	C	0	0	0	0	1	S/C	R/L	*	*	[illegible] and shifts the cursor without changing DD RAM contents	40 μs	120 μs
Function set	0	0	0	0	1	DL	N	F	*	*	Sets interface data length (DL) number of display lines (L) and character font (F).	40 μs	120 μs
Set CG RAM address.	0	0	0	1	ACG						Sets the CG RAM address. CG RAM data is sent and received after this setting.	40 μs	120 μs
Set DD RAM address	0	0	1	ADD							Sets the DD RAM address. DD RAM data is sent and received after this setting.	40 μs	120 μs
Read busy flag & address	0	1	BF	AC							Reads Busy flag (BF) indicating internal operation is being performed and reads address counter contents.	1 μs	1 μs
Write data to CG or DD RAM	1	0	Write Data								Writes data into DD RAM or CG RAM.	40 μs	120 μs
Read data to CG or DD RAM	1	1	Read Data								Reads data from DD RAM or CG RAM.	40 μs	120 μs
	I/D = 1: Increment (+1) I/D = 0: Decrement (−1) S = 1: Accompanies display shift. S/C = 1: Display shift S/C = 0: Cursor move R/L = 1: Shift to the right. R/L = 0: Shift to the left. DL = 1: 8 bits DL = 0: 4 bits N = 1: 2 lines N = 0: 1 line F = 1: 5 × 10 dots F = 0: 5 × 7 dots BF = 1: Internally operating BF = 0: Can accept instruction										DD RAM: Display data RAM CG RAM: Character generator RAM ACG: CG RAM address ADD: DD RAM address Corresponds to cursor address. AC: Address counter used for both of DD and CG RAM address.	Execution time changes when frequency changes. (Example) When fosc is 270 kHz: $40\ \mu s \times \frac{250}{270} = 37\ \mu s$	

*No effect

Notes 1. Applied to models driven by 1/8 duty or 1/11 duty.
2. Applied to models driven by 1/16 duty.

รูปที่ 2.12



รูปที่ 2.15 ตัวอย่างการต่อใช้งานจริงกับ ET-BOARD V3.0

จากวงจรเป็นการต่อ 8255 ให้เข้าใช้กับ LCD โดยเราจะจำลองสัญญาณต่างๆขึ้นมา โดยการใช้ PORT A และ PORT B โดย PORT A นั้นเราให้เป็น DATA PORT และ PORT B นั้นเราให้เป็นสัญญาณควบคุมไปใช้

เมื่อเราเริ่มเปิดไฟป้อนให้ HD44780 นั้นก็จะทำการ RESET ตัวมันเองโดยจะใช้เวลาประมาณ 10 ms หลังจากไฟ VDD ถึง 4.5 VOLT แล้วโดยตจะ SET ตัวเองดังนี้ :

1. DISPLAY CLEAR จะทำการลบข้อมูลจอภาพ LCD

2. FUNCTION SET โดยจะ SET ค่าภายใน

DL = 1 :เป็นการ SET ให้การติดต่อแบบ 8 BIT

N = 0 :SET เป็น 1 บรรทัดการแสดงผล

M = 0 :5X7 DOT ต่อหนึ่งตัวอักษณ

3. DISPLAY ON/OFF D = 0 : DISPLAY OFF

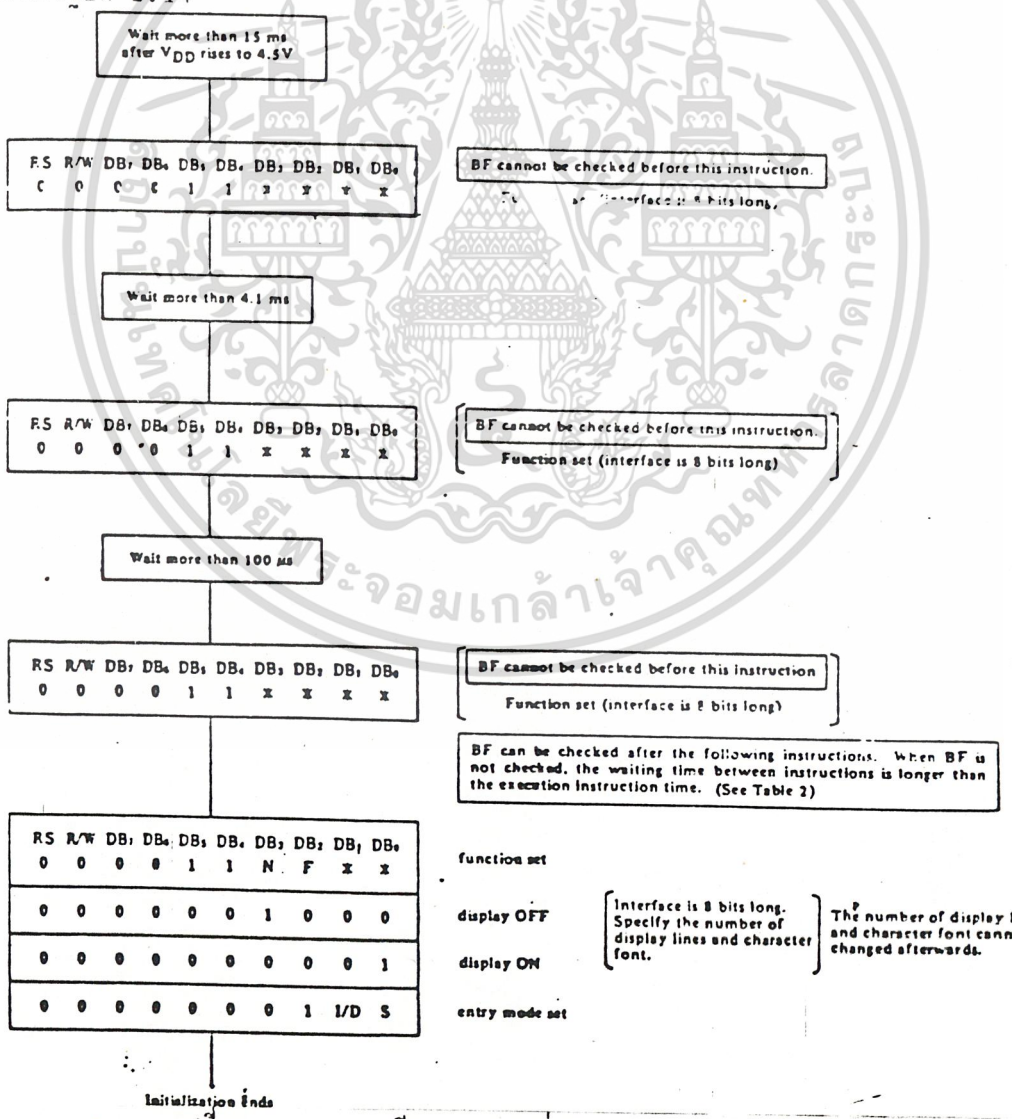
C = 0 : CURSOR OFF

B = 0 : BLINK OFF

4. ENTRY MODE SET I/D = 1 : +1(เพิ่มค่า COUNTER ขึ้น1)

S = 0 : NO SHIFT

เมื่อเราเริ่มเปิดเครื่องทำงานแล้วก็จะสั่งคำสั่งควบคุมให้มันเริ่มทำงานดังรูปที่ 2.14



รูปที่ 2.14 การเตรียมสถานะเริ่มต้นให้กับ LCD

ตารางคำสั่ง HD44780

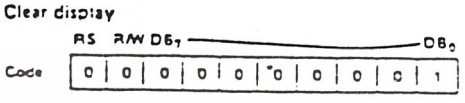
Instruction	Code										Description	Execution time (when fosc is 250 kHz) Note 1	Execution time (when fosc is 160 kHz) Note 2
	RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0			
Clear display	0	0	0	0	0	0	0	0	0	1	Clears all display and returns the cursor to the home position (Address 0).	82 μs ~ 1.64 ms	120 μs ~ 4.9 ms
Return home	0	0	0	0	0	0	0	0	1	*	Returns the cursor to the home position (Address 0). Also returns the display being shifted to the original position. DD RAM contents remain unchanged.	40 μs ~ 1.6 ms	120 μs ~ 4.8 ms
Entry mode set	0	0	0	0	0	0	0	1	I/D	S	Sets the cursor move direction and specifies or not to shift the display. These operations are performed during data write and read.	40 μs	120 μs
Display ON/ OFF control	0	0	0	0	0	0	1	D	C	B	Sets ON/OFF of all display (D), cursor ON/OFF (C), and blink of cursor position character (B).	40 μs	120 μs
Cursor and display shift	0	0	0	0	0	1	S/C	R/L	*	*	Moves the cursor and shifts the display without changing DD RAM contents.	40 μs	120 μs
Function set	0	0	0	0	1	DL	N	F	*	*	Sets interface data length (DL), number of display lines (L) and character font (F).	40 μs	120 μs
Set CG RAM address.	0	0	0	1	A_{CG}						Sets the CG RAM address. CG RAM data is sent and received after this setting.	40 μs	120 μs
Set DD RAM address	0	0	1	A_{DD}							Sets the DD RAM address. DD RAM data is sent and received after this setting.	40 μs	120 μs
Read busy flag & address	0	1	BF	AC							Reads Busy flag (BF) indicating internal operation is being performed and reads address counter contents.	1 μs	1 μs
Write data to CG or DD RAM	1	0	Write Data								Writes data into DD RAM or CG RAM.	40 μs	120 μs
Read data to CG or DD RAM	1	1	Read Data								Reads data from DD RAM or CG RAM.	40 μs	120 μs
	I/D = 1: Increment (+1) I/D = 0: Decrement (−1) S = 1: Accompanies display shift. S/C = 1: Display shift S/C = 0: Cursor move R/L = 1: Shift to the right. R/L = 0: Shift to the left. DL = 1: 8 bits DL = 0: 4 bits N = 1: 2 lines N = 0: 1 line F = 1: 5 × 10 dots F = 0: 5 × 7 dots BF = 1: Internally operating BF = 0: Can accept instruction										DD RAM: Display data RAM CG RAM: Character generator RAM A_{CG}: CG RAM address A_{DD}: DD RAM address Corresponds to cursor address. AC: Address counter used for both of DD and CG RAM address.	Execution time changes when frequency changes. (Example) When fosc is 270 kHz: $40\ \mu s \times \frac{250}{270} = 37\ \mu s$	

*No effect

Notes 1. Applied to models driven by 1/8 duty or 1/11 duty.
2. Applied to models driven by 1/16 duty.

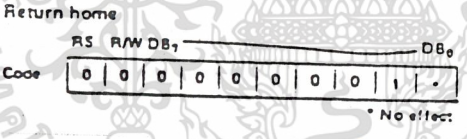
รายละเอียดของคำสั่ง HD44780

1. CLEAR DISPLAY



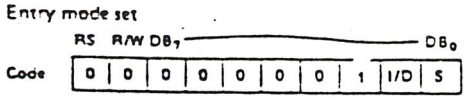
คำสั่งนี้จะเป็นการเขียนช่องว่างหรือ SPACE (ASCII 20H) เข้าไปใน DD RAM ทั้งหมดและทำการ SET DD RAM ADDRESSER เป็นศูนย์ ตัว CURSOR จะกลับไปอยู่ตำแหน่งบนสุดซ้ายมือของจอภาพ SET I/D = 1,S ไม่มีการเปลี่ยน

2. RETURN HOME



คำสั่งนี้จะทำการ SET DD RAM ADDRESSER เป็นศูนย์ ตัว CURSOR จะกลับไปอยู่ตำแหน่งบนสุดซ้ายมือของจอภาพข้อมูลในจอภาพไม่เปลี่ยน

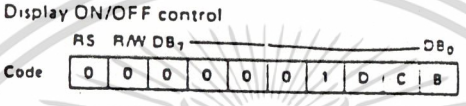
3. ENTRY MODE SET



BIT I/D : โดยจะเป็นตัวกำหนดให้ว่าเมื่อเขียนหรืออ่านข้อมูลแล้ว จะทำให้ DD RAM ADDRESS เพิ่มขึ้นหนึ่งหรือลดลงหนึ่งโดย 1= เพิ่ม 0 = ลดลงหนึ่ง

BIT S : เป็นตัวกำหนดแสดงผลโดยถ้า S = 1 จะเป็นการใส่ข้อมูลแล้วตัว CURSOR อยู่กับที่ข้อมูลจะถูกดันไปทางซ้าย ถ้าถ้า S = 0 ข้อมูลจะอยู่กับตัว CURSOR จะถูกดันไปทางขวามือ

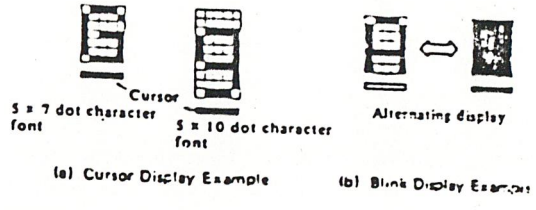
4. DISPLAY ON/OFF CONTROL



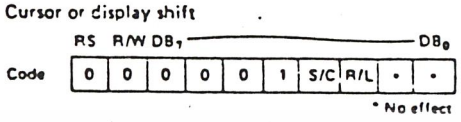
BIT D : เป็น BIT ให้เปิดปิดหน้าหน้าจอภาพโดยถ้า D = 1 จะ ON และ D = 0 จะ OFF

BIT C : จะให้แสดง CURSOR ให้ BIT C = 1 และถ้าไม่ต้องการแสดง CURSOR BIT C = 0 โดยตัว CURSOR จะอยู่ LINE ที่ 8 ในแบบ 5X7 DOT และจะอยู่ LINE ที่ 11 ในแบบ 5X10 DOT

BIT B : เป็น BIT SET การกระพริบของ CURSOR โดย B = 1 มีการกระพริบ B = 0 ไม่มีการกระพริบ โดยมีระยะเวลาการกระพริบประมาณ 379.2 ms



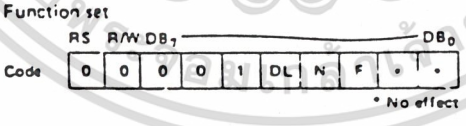
5. CURSOR OR DISPLAY SHIFT



เป็นคำสั่งกำหนดให้ตำแหน่ง CURSOR หรือข้อมูลไปเกิดทางซ้ายหรือขวาโดยไม่ต้องใช้คำสั่งเขียนหรืออ่าน โดย

S/C	R/L	
0	0	ทำการย้าย CURSOR ไปจากตำแหน่งเดิมไปซ้ายมือ 1 ตำแหน่ง
0	1	ทำการย้าย CURSOR ไปจากตำแหน่งเดิมไปขวามือ 1 ตำแหน่ง
1	0	เป็นการดันตัวอักษรที่เกิดไปทางซ้าย
1	1	เป็นการดันตัวอักษรที่เกิดไปทางขวามือ

6. FUNCTION SET

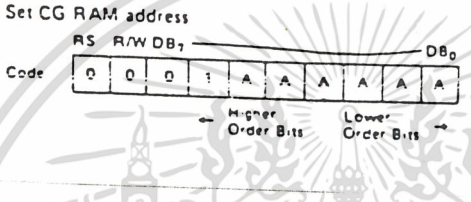


BIT DL : เป็นการ SET การติดต่อว่าจะให้เป็นแบบ 8 BIT หรือ 4 BIT โดยถ้าต้องการติดต่อ 4 BIT DL = 0 และ 8 BIT DL = 1

N : เป็นการ SET บรรทัดการแสดงผล N = 0 แสดง 1 บรรทัด N = 1 แสดง 2 บรรทัดในกรณีมากกว่า 2 บรรทัด ก็ให้ SET N = 1

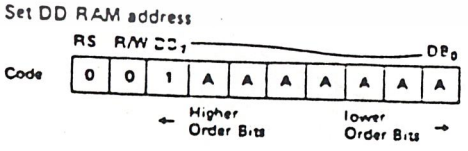
F : เป็นการ SET ขนาด DOT การแสดงผล 5x7 หรือ 5x10 โดย F = 0 เป็นแบบ 5x7 และ F = 1 เป็นแบบ 5x10

7. SET CG RAM ADDRESS



ใน HD44780 นั้นจะมีหน่วยความจำอยู่ 2 ชนิด คือ DISPLAY DATA RAM (DD RAM) จำนวน 80x8 BIT และ CHARACTER GENERATOR ROM CG RAM จำนวน 512 BIT และ 7200 BIT คำสั่งนี้จะเป็นการ SET ADDRESS ก่อนเขียนหรืออ่านข้อมูลจาก CG RAM ด้วย

8. SET DD RAM ADDRESS



เป็นคำสั่ง SET ค่า ADDRESS ใน DD RAM ในการเขียนหรืออ่านค่าจาก DD RAM (DD RAM คือ ส่วนที่จะแสดงผลหน้าจอ LCD) โดยจำนวน ADDRESS ที่จะเกิดขึ้นบนจอ LCD จะอยู่กับ SET ค่า N ด้วย

ถ้า N = 0 (1 บรรทัด) ADDRESS จะอยู่ 00H-4FH

ถ้า N = 1 (2 บรรทัด) ADDRESS จะอยู่ 00H-27H สำหรับบรรทัดที่ 1 และ 40H-67H สำหรับบรรทัดที่ 2

ตัวอย่างการจัด ADDRESS ของ DD RAM หน้า LCD แบบ 16 ตัวอักษร 4 บรรทัด และ 20 ตัวอักษร 2 บรรทัด HDM-16416H, HDM-20216H

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	
1-line	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	← display position
2-line	40	41	42	43	44	45	46	47	48	49	4A	4B	4C	4D	4E	4F	← DD RAM address
3-line	10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D	1E	1F	
4-line	50	51	52	53	54	55	56	57	58	59	5A	5B	5C	5D	5E	5F	

HDM-16416H

HDM-16416H

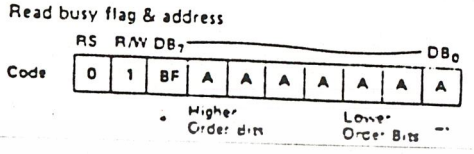
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	
1-line	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	10	11	12	13	← display position
2-line	40	41	42	43	44	45	46	47	48	49	4A	4B	4C	4D	4E	4F	50	51	52	53	← DD RAM address
3-line	14	15	16	17	18	19	1A	1B	1C	1D	1E	1F	20	21	22	23	24	25	26	27	
4-line	54	55	56	57	58	59	5A	5B	5C	5D	5E	5F	60	61	62	63	64	65	66	67	

(Note) Shift display is as same as 2-line type.

HDM-20216H

HDM-20216H

9. REAS BUSY FLAG AND ADDRESS



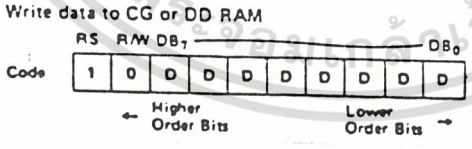
เป็นคำสั่งอ่านค่า BUSY FLAG ซึ่งจะเป็นตัวบอกว่าตัว HD44780 นั้นอยู่ในขบวนการทำงานภายในอยู่หรืออยู่ในสภาพหรือพร้อมจะรับข้อมูล โดย

BF = 1 อยู่ในขบวนการทำงานภายในไม่พร้อมจะรับข้อมูลหรือคำสั่ง

BF = 0 พร้อมจะรับข้อมูลหรือคำสั่งได้

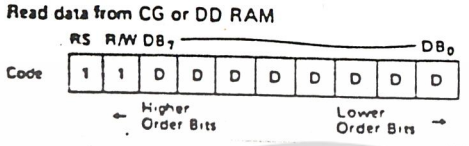
และนอกจากนี้ยังเป็นคำสั่งอ่านค่าข้อมูล ADDRESS ของ CG RAM หรือ DD RAM ด้วย

10. WRITE DATA TO CG หรือ DD RAM



เป็นคำสั่งเขียนข้อมูลเข้าไปใน CG หรือ DD RAM โดยเมื่อเขียนข้อมูลและ ADDRESS จะเพิ่มหรือลดโดยอัตโนมัติตามคำสั่งที่ SET ใน ENTRY MODE ข้อกำหนดที่จะว่าเป็นการเขียนข้อมูลของ CG RAM หรือ DD RAM ทำได้โดยการ SET ADDRESS ของ CG RAM หรือ DD RAM ขึ้นมาก่อนจะเขียนข้อมูล

11. READ DATA FROM CG OR DD RAM



เป็นคำสั่งอ่านค่าข้อมูลจาก CG RAM หรือ DD RAM โดยก่อนอ่านค่าจาก DD RAM หรือ CG RAM นั้นควรจะใช้คำสั่ง SET ADDRESS ก่อนเพื่อให้รู้ว่าข้อมูลที่อ่านได้นั้นเป็น DD หรือ CG RAM

จากตารางทำงานจะเห็นว่าการใช้งาน LCD MODULE นั้นง่ายเพียงแต่เราส่งคำสั่งเริ่มแรกและ SET ความต้องการในขนาดตัวอักษร, CURSOR หลังจากนั้นเราก็สามารถเขียนตัวอักษรเข้าไปใน DD RAM ตามตารางตัวอักษรที่ให้มานั้นก็จะเกิดอักษรในจอภาพ LCD เรายังสามารถกำหนดตำแหน่งอักษรที่ที่จะให้เกิดบนจอได้โดยการ SET DD RAM ADDRESS ตามตารางที่ให้มาในหัวข้อ SET DD RAM ADDRESS ขอให้ทดสอบทำความเข้าใจกับตัวโปรแกรมที่ใช้กับ ET-BOARS V3.0 นี้ที่ให้มาจะเห็นว่าจะมีส่วนเริ่มต้นก็คือ ส่วนการ INITAL LCD เพื่อกำหนดหน้าที่การทำงานต่างๆ

นอกจากนี้ LCD MODULE (HD44780) นี้จะยังมีส่วนหนึ่งของ CHARACTER GENERATOR ที่เราสามารถเขียนข้อมูลในการเกิดตัวอักษราขึ้นได้เอง จากตารางตัวอักษร 5X7 DOT นั้นจะเห็นว่า คือ ตำแหน่งในตาราง OOH ถึง 07H ส่วนตำแหน่ง 08H-OFH จะเป็นตำแหน่งเดียวกับ OOH-07H จะเห็นว่าจะมี CHARACTER GENERATOR 8 ตัวที่เราสามารถเขียนข้อมูลกำหนดเองได้และถ้าเป็นแบบ 5X10 DOT จะเขียนได้ 4 ตัวอักษร ซึ่งจากข้อพิเศษนี้ทำให้เราสามารถเาขียนตัวอักษรสัญลักษณ์หรืออักษรภาษาไทยได้

การเขียนข้อมูล CHARACTER GENERATOR

เราสามามารถเขียนข้อมูลได้โดยกำหนด ADDRESS ของ CG RAM ก่อนโดยเขียนได้ 64 ตำแหน่ง BIT 5-BIT 0 และเมื่อกำหนด ADDRESS แล้วก็จะทำการาเขียนข้อมูลลงใน CG RAM โดยเป็นลักษณะ BIT ต่อ BIT บนจอ 1 ตัวอักษร คือ 5X7 DOT นั้น จะใช้ข้อมูล BIT 4 ถึง BIT 0 ต่อ 1 BYTE เท่านั้น 1 ตัวอักษรจะใช้ข้อมูล 8 BYTE ด้วยกันให้ดูจากตารางประกอบไปด้วยและเมื่อเขียนข้อมูลลงใน CG RAM แล้วเวลาเราจะใช้งานก็ให้เขียนข้อมูลใน DD RAM คือ ข้อมูลตำแหน่งในตาราง CHARACTER ที่ตำแหน่ง OOH-07H

ตัวอย่างโปรแกรมการเขียนข้อมูลตัวหนังสือภาษาไทยเป็นตัว (อ) (ท) และตัวของ ( ) เข้าไปใน CG RAM ตำแหน่งที่ OOH, 01H และ 02H และนำมาแสดงผลทางจอ LCD โดยใช้ 2 บรรทัดในการแสดงผล

สรุป การใช้งาน LCD MODULE นั้นที่สำคัญคือ ต้องเข้าใจในตัว CONTROLLER ของ LCD MUODULE นั้นโดย CONTROLLER ทุกๆบริษัทจะมีการทำงานที่เหมือนๆกันเป็นส่วนใหญ่

ขาต่างๆในการต่อใช้งาน HD44780

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

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

3. E(ENABLE SIGNAL) เป็นขากำหนดสภาพการรับเขียนอ่านข้อมูล

4. DB0-DB7 เป็นขารับส่งข้อมูลจากตัว IC

5. VDD ไฟเลี้ยงตัววงจร

6. VSS เป็นขา GND

7. VO เป็นขารับ VOLTAGE ในการขับ LCD ให้สว่างหรือมืด

The relation between the operation and the combination of RS, R/W

RS	RW	E	OPERATION
0	0		Write instruction code
0	1		Read busy flag and address counter
1	0		Write data
1	1		Read data

When performing data and instruction code by 4 bit, transfer RS, R/W every time.

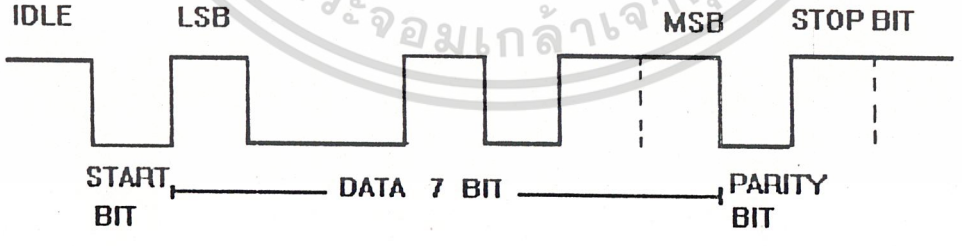
รูปที่ 2.15

Serial Data Communication

ในการส่งข้อมูลจะใช้ Program Mode 1 :" Standard 8-Bit Uart Mode รูปแบบการส่งจะส่งข้อมูล 8 บิท เป็ ASCII Code 7 Bit และ Parity 1 Bit หรือจะละ Parity Bit ทิ้งก็ได้ก็จะกลายเป็นการส่งข้อมูล 8 บิท ในการส่งจะเป็นแบบ Asynchornous Mode Data Transmission ซึ่งต้องใช้ Start Bit และ Stop Bit เพื่อให้วงจรทางด้านรับ Detect Bit เริ่มต้นและบิทสุดท้ายได้ เพราะฉะนั้นข้อมูลที่ต้องส่งไปจริงทั้งหมดมี 10 บิท ประกอบด้วย ASCII Code 7 Bit Parity 1 Bit Start และ Stop Bit อีก 2 Bit

การส่งจะเริ่มเมื่อมี Data ถูกเขียนเข้าไปยัง SBUF และการส่ง Byte ต่อไปจะต้องแน่ใจว่า ข้อมูลชุดก่อนหน้าถูกส่งไปเรียบร้อยแล้ว ซึ่งเราสามารถทราบได้โดยการตรวจดู บิท T_1 ใน SFR SCON ว่าอยู่ในสถานะ High หรือไม่ การเคลื่อนย้ายข้อมูลจะเริ่มจาก Start Bit ซึ่งจะทำให้เกิดการเปลี่ยนแปลงของสถานะของขา TXD ซึ่งจะเปลี่ยนจาก High เป็น Low จะเป็นสัญญาณบอกให้ Circuit ทางด้านรับทราบว่าจะมีข้อมูลอีก 8 บิท และตามด้วย Stop Bit อีก 1 บิท โดยข้อมูลบิทต่ำ (SLB) จะถูกส่งไปก่อนสุดท้ายจะเป็น (MSB) และ Stop Bit จะเป็นบิทสุดท้ายที่จะส่งไป และเป็นสัญญาณบอกให้ทางด้านรับทราบว่าสิ้นสุดการส่งข้อมูล 1 บิท และถ้ามีข้อมูล Byte ต่อไปตามมาทันทีทันใด Start Bit ของข้อมูลชุดหลังนี้จะถูกส่งออกไปทางขวา TXD หลังจาก Stop Bit ของข้อมูล Byte ก่อนหน้าถูกส่งไปแล้ว 1 Period ของ Stop Bit ของข้อมูลชุดก่อน แต่ในทางตรงข้ามถ้ายังไม่มีข้อมูลส่งตามมาขา TXD จะคงสถานะ High ไว้เรียกสถานะนี้ว่า IDLE

ความกว้างของ Pulse ที่ส่งไปจะถูกควบคุมโดย Baud Rate Clock ที่ใช้ ซึ่ง Baud Rate ของตัวรับจะต้องตรงกับ Baud Rate ทางด้านส่ง ถ้าไม่ตรงกันจะทำให้การรับข้อมูลผิดไป



รูปที่ 2.16 สัญญาณของ RS-232

SFR ที่เกี่ยวข้องกับการส่งข้อมูลออกทาง Serial Port

1. SCON (Serial Port Control Register) ตำแหน่งหน่วยความจำภายใน 98H รีจิสเตอร์ SCON มีขนาด 8 บิท ใช้สำหรับควบคุมการส่งและการรับข้อมูลผ่านทาง Serial Port แต่ละบิทของข้อมูลใน Register นี้มีความหมายเฉพาะดังแสดงในรูป 2.17

SCON : Serial Port Control Register Bit Addressable

SM0	SM1	SM2	REN	TB8	RB8	TI	RI

รูปที่ 2.17

RI Receive Interrupt Flag

Bit นี้จะถูกกำหนดโดย Hardware ให้มีค่าเป็น 0 หรือ 1 โดยที่ในการรับข้อมูลโหมด 0 นั้น บิท RB8 จะมีค่าเป็น 1 เมื่อมีข้อมูลเข้ามาครบทั้ง 8 บิท ส่วนใน Mode อื่น Bit RB8 จะเป็น 1 ก็ต่อเมื่อข้อมูลเข้ามาถึงเวลาครึ่งหนึ่งของ Stop Bit (ยกเว้นบางกรณีจะกล่าวไว้ใน SM2) บิทนี้จะสามารถเคลียร์ให้เป็น 0 ได้โดยใช้คำสั่ง Cir Bit โดยค่าตำแหน่งของบิทมีค่าเท่ากับ 98H บิทนี้มีประโยชน์ให้รู้ว่าข้อมูลได้เข้ามามาอยู่ใน SBUF ครบทั้งชุดแล้วพร้อมที่ CPU จะอ่านไปเก็บไว้ยังหน่วยความจำต่อไปหรืออาจกำหนดค่าใน Register IE และ IP เมื่อมีข้อมูลเข้ามาทางพอร์ทอนุกรมแล้วจะทำให้เกิดการขัดจังหวะ (Interrupt) การทำงานของโปรแกรมหลัก (Main Program) แล้วกระโดดไปทำงานในโปรแกรมตอบสนองการขัดจังหวะ (Interrupt Service Routine)

TI Transmit Interrupt Flag

ค่าในบิท TI จะถูกกำหนดให้เป็น 1 หรือ 0 ด้วย Hardware โดยในการส่งข้อมูลแบบอนุกรม Mode 0 Bit นี้จะเป็น 1 เมื่อจะบอกว่าการส่งข้อมูลใน Register SBUF ออกไปทาง Port อนุกรมครบทั้ง 8 บิท แต่ถ้าเป็นการส่งข้อมูลแบบอนุกรมในโหมดอื่น จะทำให้ Bit TI เป็น 1 เมื่อเริ่มการส่ง Stop Bit ข้อมูลนี้สามารถ Clear ให้เป็น 0 ได้ด้วยคำสั่ง CLR Bit โดยที่ค่าตำแหน่งของบิทนี้เท่ากับ 99H บิทนี้ยังมีประโยชน์เพื่อบอกว่าการส่งข้อมูลจาก SBUF ออกไปทางพอร์ทอนุกรมนั้นสิ้นสุดแล้วพร้อมที่จะให้โปรแกรมเขียนข้อมูลลงไปยัง SBUF นอกจากนี้การกำหนดค่าใน Register IE และ IP ยังสามารถที่จะกำหนดให้เกิดการขัดจังหวะการทำงานของโปรแกรมได้เมื่อบิทนี้ถูก Hardware ทำให้เป็น 1

PB8

เมื่อมีการกำหนดให้รับข้อมูลในโหมด 2 และ 3 จะใช้บิทนี้สำหรับเก็บข้อมูลบิทที่ 9 ที่เข้ามาทางพอร์ท RXD ส่วนในโหมด 1 นั้น บิทนี้จะเก็บ Stop Bit ซึ่งมีค่าเป็น 1 ค่าตำแหน่งประจำบิทคือ 9AH

TB8

ในการส่งข้อมูลแบบอนุกรมโหมด 2 และ 3 จะใช้บิทนี้เก็บข้อมูลบิทที่ 9 ส่วนโหมดอื่นจะไม่ใช้งานบิทนี้ ค่าในบิทนี้สามารถกำหนดได้โดยใช้คำสั่ง CLR Bit ค่าตำแหน่งบิทนี้คือ 9BH

REN Receive Enable

เป็นบิทที่จะใช้กำหนดให้ทำการรับข้อมูลเข้ามาจากทางพอร์ตอนุกรม (Serial Port) หรือไม่ถ้าบิทนี้เป็น 1 จะรับข้อมูลเข้ามา แต่ถ้าเป็น 0 ก็จะไม่รับข้อมูลเข้ามาแต่ถ้าเป็น 0 ก็จะไม่รับข้อมูลที่ขา RXD เข้ามาการทำให้บิทนี้เป็น 1 หรือ 0 ทำได้โดยใช้คำสั่ง SETB Bit หรือ CLR Bit ค่าตำแหน่งของบิทนี้คือ 9CH

SM2

เป็นบิทสำหรับควบคุมการทำงานของ Hardware ที่จะทำให้ RI เป็น 1 หรือไม่ในกรณีที่บิท SM2 เป็น 0 ค่าในบิท RI ก็จะเป็นไปตามที่ได้อธิบายมาแล้วในเรื่องบิท RI แต่ถ้า SM2 = 1

ในโหมด 1 บิท RI มีค่าเป็น 1 เมื่อข้อมูล Stop Bit เข้ามายังพอร์ทอนุกรมถูกต้องแต่ถ้า Stop Bit ไม่เข้ามายังพอร์ทอนุกรมอันอาจเกิดจากปัญหาในการส่งข้อมูลแล้วบิท RI จะมีค่าเป็น 0

SM0, SM1

เป็น 2 บิท ที่ใช้งานร่วมกัน เพื่อกำหนดโหมดของการรับ-ส่งข้อมูลของพอร์ตอนุกรมค่าใน 2 บิท นี้จะกำหนดโหมดได้ดังนี้

SM0	SM1	MODE	DESCRIPTION
0	0	0	Shift Register
0	1	1	8-Bit UART
1	0	2	9-Bit UART
1	1	3	9-Bit UART

Serial Port Set-Up

MODE	SCON	SM2 VARIATION
0	10H	
*1	50H	Single Processor
2	90H	Environment
3	D0H	SM2 = 0

Timer Register TH0, TL0, TH1, TL1

ตำแหน่งหน่วยความจำภายในเท่ากับ 8CH, 8AH, 8DH, 8DH

ใน 8051 จะมีวงจร Timer 0 และ Timer 1 (8052 จะมี Timer 2 อีก 1 ชุด) ใน Timer แต่ละชุดจะมี Register ขนาด 8 บิทอยู่ 2 ตัว เพื่อเก็บค่าของการนับของ Timer ได้สูงสุดถึง 16 บิทใน Timer 0 Register นั้คือ TH0, TL0 และใน Timer 1 คือ Register TH1, TL1, TLx (x หมายถึง 0 หรือ 1) จะเก็บค่าของการนับ 8 บิทล่าง และ THx จะเก็บค่าของการนับ 8 บิทบน ผู้ใช้จะสามารถการทำงานของวงจร Timer ในโหมด Timer หรือโหมด Counter ได้โดยการกำหนดใน Register ชื่อ TMOD (Timer Register) การทำงานเป็น Timer นั้นจะให้ Register ใน Timer 0 หรือ 1 ทำการนับจำนวนไซเคิล (Cycle) ของสัญญาณนาฬิกาที่ผ่านวงจรหาร 12 ดังรูปที่ 5.7 เมื่อมีการนับครบถึงค่าสูงสุดที่ Register TLx และ THx จะเก็บได้คือค่า FFFFH แล้วยังนับต่อไปค่าที่ได้จากการนับจะเป็น 000H ทำให้เกิดการ Set บิทบางบิทใน Register TCON เพื่อบอกสภาวะ Timer Overflow นี้ ในการให้วงจร Timer ทำงานเป็น Counter ก็คือการใช้ Register THx และ TLx ทำการนับจำนวนไซเคิลของสัญญาณที่เข้ามาทางขา T0 หรือ T1 ของ 8051 สัญญาณที่เข้ามาทางขา T0 หรือ T1 อาจมาจากอุปกรณ์ตรวจจับ (Sensor) ก็ได้แต่สภาวะของสัญญาณนี้จะต้องมีระดับโวลเตจของสภาวะลอจิก 0 หรือ 1 เป็นแบบ TTL คือลอจิก 0 จะต้องมีโวลเตจไม่เกิน 0.6 โวลท์ และลอจิก 1 จะต้องมีโวลเตจมากกว่า 2.4 โวลท์

TMOD Timer/Counter Mode Register

ตำแหน่งหน่วยความจำภายในเท่ากับ 89H

TMOD เป็น Register ขนาด 8 บิทมีหน้าที่ทำการควบคุมการทำงานของ Timer 0 และ Timer 1 แต่ละบิทใน Register นี้ความหมายเฉพาะดังรูปที่ 5.6

GATE	C/T	M1	M0	GATE	C/T	M1	M0

Timer 1 Timer 0

GATE when TRx [in TCON] is set and GATE = 1. Timer/Counter Rx will run only while INTx pin is high [Hardware Control]. When GATE = 0 Timer/Counter Rx will run only while TRx = [Software Control].

C/T Timer or Counter Serector.Cleared for Timer Operation [Input from Imtermal System Clock]. Set for Counter Operation [Input from Tx Input Pin].

M1 Mode Selector Bit [NOTE 1]

M0 Mode Selector Bit [NOTE 1]

NOTE 1 :

M1	M0	Operating Mode	
0	0	0	13-Bit Timer
0	1	1	16-Bit Timer/Counter
1	0	2	8-Bit Auto-Relode Timer/Counter Controlled by The Standard Timer 0 Control Bits. TH0 is an 8 Bit Timer and is Controlled by Timer 1 Control Bits.
1	1	3	[Timer 1] Timer/Counter 1 Stopped.

รูปที่ 2.18 TMOD Timer/Counter. Mode Register

ในรูปที่ 2.18 M0 เป็นชื่อของบิท 0 และ GATE ทางซ้ายสุดเป็นชื่อของบิท 7 รีจิสเตอร์นี้แบ่งข้อมูลออกเป็น 2 ชุด ชุดละ 4 บิทคือ บิท 0-3 ใช้สำหรับควบคุมการทำงานของ Timer 0 และบิท 4-7 ใช้ควบคุมการทำงานของ Timer ของแต่ละบิทที่มีชื่อเดียวกันจะเหมือนกัน

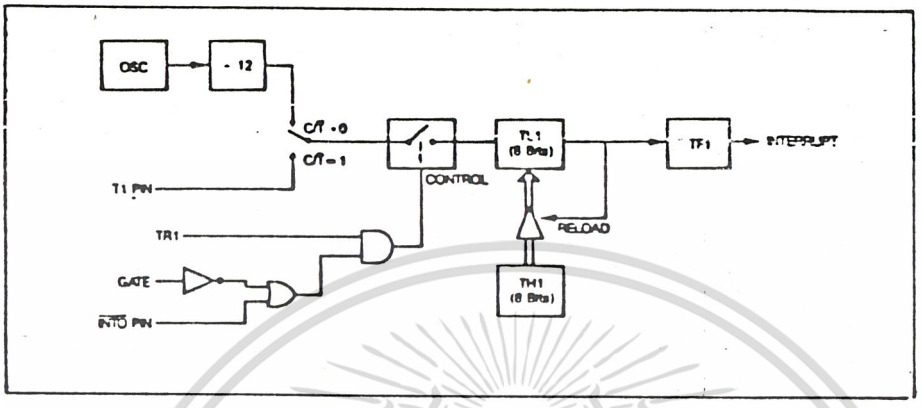
GATE เป็นบิทที่ใช้ควบคุมให้ Timer ทำงานหรือไม่ ถ้าบิทนี้ของ Timer x ถูกตั้งเป็น 1 จะทำให้ Timer ทำงานก็ต่อเมื่อที่ขา INTx มีสภาวะลอจิกเป็น 1 และบิท TRx ใน Register TCON เป็น 1 ด้วย

C/T บิทนี้ใช้สำหรับเลือกการทำงานของ Timer ว่าจะใช้เป็น Timer หรือ Counter ถ้าบิทนี้เป็น 1 ก็หมายความว่า เลือกการทำงานเป็น Counter ซึ่งจะนับจำนวนไซเคิลของสัญญาณที่เข้ามาทางขา Tx

M1, M0 บิทที่ใช้ร่วมกันเพื่อเลือกโหมดการทำงานของ Timer การทำงานโหมด 0, 1 และ 2 ของ Timer 0 จะเหมือนกับ Timer 1 แต่ในโหมด 3 การทำงานของทั้งสองจะต่างกัน ค่าใน M1 และ M0 จะเลือกโหมดการทำงานดังนี้

M1	M0	การทำงาน
0	0	โหมด 0 รีจิสเตอร์ THx และ TLx ทำตัวเป็นตัวนับ 13 บิท ค่าจากการนับ 8 บิทบนมาจาก 8 บิทของ THx และอีก 5 บิทล่างมาจากค่า 5 บิทล่างของรีจิสเตอร์ TLx โดยที่ 3 บิทบนของ TLx จะไม่ต้องสนใจเลย
0	1	โหมด 1 รีจิสเตอร์ THx และ TLx ทำตัวเป็นตัวนับ 16 บิทค่าจากการนับ 8 บิทบนอยู่ในรีจิสเตอร์ THx และค่าจากการ 8 บิทล่างอยู่ในรีจิสเตอร์ TLx
1	0	โหมด 2 ในการนับของ Register TLx ขนาด 8 บิท เมื่อนับถึงค่าสูงสุดคือ FFH เมื่อทำการนับต่อไปจะเกิดการ Overflow แล้วก็จะ Relode เอาข้อมูลจาก THx เข้าไปยัง TLx เพื่อเป็นค่าเริ่มต้นในการนับครั้งต่อไป
1	1	โหมด 3 การทำงานของ Timer 0 และ Timer 1 จะต่างกันดังที่จะกล่าวต่อไป

โหมด 2 Auto Relode



รูปที่ 2.19 Timer Mode 2

ในรูปที่ 2.19 เป็นไดอะแกรมของวงจร Timer 8051 ที่ทำงานโหมด 2 Timer 0 และ Timer 1 มีการทำงานในโหมด 2 เหมือนกันโดยสามารถกำหนดให้ทำหน้าที่เป็น Timer หรือ Counter ได้โดยบิท C/T และการควบคุมการนับได้โดยข้อมูลในบิท TR1 และ GATE ในรีจิสเตอร์ TMOD กับสัญญาณที่ขา INTx เมื่อเริ่มการทำงานข้อมูลในรีจิสเตอร์ TH1 จะถูกโหลดไปยังรีจิสเตอร์ TL1 ทำให้รีจิสเตอร์ TH1 และ TL1 มีการนับเหมือนกัน เมื่อเกิดการนับจำนวนไซเคิลของสัญญาณที่ออกจากสวิทช์ Control จะทำให้ค่าจากการนับในรีจิสเตอร์ TL1 เพิ่มขึ้นไปเรื่อย ๆ ทีละ 1 ไปจนถึง OFFH ในการนับครั้งต่อไปจะทำให้บิท TF1 ในรีจิสเตอร์ TCON ไม่เป็น 1 และข้อมูลในรีจิสเตอร์ TH1 จะถูกโหลดไปยังรีจิสเตอร์ TH1 จะถูกโหลดไปยังรีจิสเตอร์ TL1 เพื่อเป็นค่าเริ่มต้นการนับต่อไป

TCON Timer Control Register

ตำแหน่งหน่วยความจำภายในเท่ากับ 088H

รีจิสเตอร์ขนาด 8 บิทนี้ใช้ควบคุมการทำงานและบอกสภาวะของ Timer 0 และ Timer 1 แต่บิทของรีจิสเตอร์นี้จะทำงานต่างกันดังรูปที่ 2.20

TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0

รูปที่ 2.20

TF1 TCON.7 Timer 1 is overflow flag set by hardward when the timer/counter 1 overflows. Cleared by hardware as proceesor vectors to the interrupt service routine.

TR1 TCON.6 Timer 1 run control bit. Set/cleared by software to turn timer/counter 1 ONOFF.

TF0 TCON.5 Timer 0 overflow flag. Set by hardware when the timer/counter 0 overflow cleared by hardward as proceesor vectors to the service routine.

TR0 TCON.4 Timer 0 run control bit. Set/cleared by software to turn time/counter 0 ON/OFF.

IE1 TCON.3 External interrupt 1 edge flag. Set by hardware when external interrupt edge is detected cleared by hardware when interrupt is processed.

IT1 TCON.2 Interrupt 1 type control bit. Set/cleared by triggered external interrupt.

IE0 TCON.1 External interrupt 0 edge flag. Set by hardward when externternal interrupt edge detected cleared by hardware when interrupt is processed.

IT0 TCON.0 Interrupt 0 type control bit. Set/cleared by software to specify falling edge/low level triggered external interrupt.

ในรีจิสเตอร์ TCON แต่ละบิทมีหน้าที่การทำงานดังนี้

IT0 Interrupt 0 เป็นบิทที่จะใช้กำหนดวิธีการขัดจังหวะโปรแกรม อันเนื่องจากสภาวะของสัญญาณที่เข้ามาทางขา INT0

ถ้า IT0 เป็น 0 จะเกิดการขัดจังหวะโปรแกรม ถ้าสภาวะของสัญญาณที่ขา INT0 เปลี่ยนจาก 1 เป็น 0

ถ้า IT0 เป็น 0 จะเกิดการขัดจังหวะโปรแกรม ถ้าสภาวะของสัญญาณที่ขา INT0 เป็น 0

IE0 บิทนี้จะเป็น 1 ถ้าสัญญาณที่เข้ามาทางขา INT0 มีสภาวะลอจิกของสัญญาณ ตามที่กำหนดในบิท IT0 แล้วทำให้เกิดการขัดจังหวะโปรแกรม เมื่อเกิดการกระโดดไปทำงานยังโปรแกรมตอบสนองการขัดจังหวะแล้ว จะทำให้บิทนี้กลับเป็น 0

IT1 Interrupt 1 เป็นบิทที่จะใช้กำหนดวิธีการขัดจังหวะโปรแกรมอันเนื่องจากสภาวะของสัญญาณที่เข้ามาทางขา INT1

ถ้า IT1 เป็น 1 จะเกิดการขัดจังหวะโปรแกรม ถ้าสภาวะของสัญญาณที่ขา INT1 เปลี่ยนจาก 1 เป็น 0

ถ้า IT1 เป็น 0 จะเกิดการขัดจังหวะโปรแกรม ถ้าสภาวะของสัญญาณที่ขา INT1 เป็น 0

IE1 บิทนี้จะเป็น 1 ถ้ามีสัญญาณเข้ามาทาง INT1 มีสภาวะลอจิกของสัญญาณ ตามที่กำหนดในบิท IT1 แล้วทำให้เกิดการขัดจังหวะโปรแกรมเหมือนกับบิท IT0 ที่ทำงานกับสัญญาณ INT0

TR0 Timer 0 run control bit บิทนี้ถ้าเป็น 0 Timer 0 ไม่ทำการนับสัญญาณไม่ว่ากรณีใด ๆ ทั้งสิ้น แต่ถ้าบิทนี้เป็น 1 จะทำให้ Timer 0 ทำงาน โดยขึ้นสัญญาณ GATE, INT0 ดังใน บทที่ 5.1.10 ข้อมูลในบิทนี้จะสามารถ Set เป็น 1 หรือ Clear เป็น 0

TF0 Timer 0 Overflow Flag บิทนี้จะ 1 เมื่อการนับของ Register ของ Timer 0 [TL0 หรือ TH0 ขึ้นกับโหมดการทำงาน] เกิด Overflow ขึ้น คือเอาการนับเพิ่มไปจนถึงค่าสูงสุดแล้วนับต่อไป ทำให้ค่าการนับกลับมาเริ่มต้นใหม่ที่ 0 หรือค่า Relode เมื่อ 8051 กระโดดไปทำงานที่โปรแกรมตอบสนอง การขัดจังหวะจะทำให้บิทนี้กลับเป็น 0

TR1 Timer 1 Run Control Bit การทำงานจะเหมือนกับการทำงานของบิท TR0 แต่บิทนี้จะทำงานกับ Timer 1

TF1 Timer 1 Overflow Flag บิทนี้เหมือนกับบิท TF0 ต่างกันที่ขึ้นกับการทำงาน Timer 1

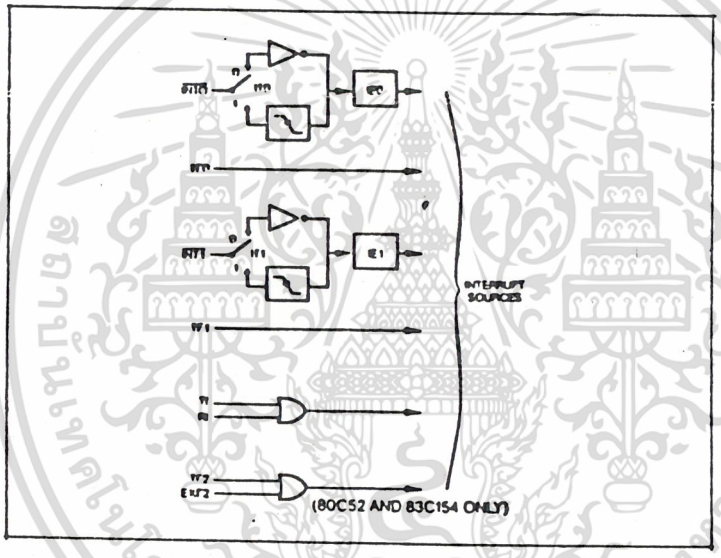
4 บิทแรกที่กล่าวมานั้นจะเกี่ยวข้องกับการขัดจังหวะ [Interrupt] ส่วนที่ 4 บิทหลังนั้นได้กล่าว มาแล้วอย่างละเอียดในเรื่องโหมดการทำงานของ Timer

ในขณะที่ Timer ทำงานในโหมดของ Timer นั้น รีจิสเตอร์ที่ทำหน้าที่เป็นตัวนับจะมีค่าเพิ่มขึ้น 1 ทุก ๆ 1 ไซเคิลของเครื่อง ซึ่งเท่ากับ 12 คาบของสัญญาณจากออสซิลเลเตอร์ ในกรณีที่ Timer ทำงานเป็น Counter เพื่อนับจำนวนไซเคิลของสัญญาณที่เข้ามาทางขา T0 หรือ T1 รีจิสเตอร์จะเพิ่มค่าไป 1 เมื่อมีการเปลี่ยนสถานะของสัญญาณดังกล่าวจาก 1 เป็น 0 โดยวงจรภายใน 8051 จะตรวจสอบสถานะของสัญญาณที่ขาดังกล่าวในช่วงเวลาเฟส 2 ของ State 5 (55P) ในทุก ๆ ไซเคิลของเครื่อง เช่น ในเวลา 55F2 ครั้งหนึ่งพบว่าสัญญาณที่ขา T0 มีสภาวะลอจิกเป็น 1 และในเวลา 55P2 ของไซเคิลของเครื่องถัดมาพบว่าสัญญาณที่ขา T0 มีสภาวะลอจิกเป็น 1 ก็จะทำให้ค่าในรีจิสเตอร์เป็นตัวนับเพิ่มค่าไป 1 แต่ละไซเคิลของเครื่องจะเท่ากับ 12 ไซเคิลของสัญญาณจากออสซิลเลเตอร์ ดังนั้นสัญญาณที่จะนับได้จะต้องเป็น 1 อย่างน้อยให้ถูกจับได้ใน 1 ไซเคิลของเครื่องและเป็น 0 อย่างน้อยไม่ต้องให้ถูกจับได้ใน 1 ไซเคิลของเครื่องเช่นกัน ทำให้ความถี่สูงสุดของสัญญาณที่ขา T0 หรือ T1 ที่จะนับได้ต้องไม่เกิน 1/24 เท่าของความถี่สัญญาณจากออสซิลเลเตอร์

IE Interrupt Enable Register

ตำแหน่งหน่วยความจำภายในเท่ากับ 0A8H

การขัดจังหวะการทำงาน (Interrupt) เป็นการที่มีสัญญาณหนึ่งหรือคำสั่งหนึ่งที่ (ไม่ใช่คำสั่ง CALL หรือ JMP) ที่จะทำให้การทำงานปกติของโปรแกรมถูกขัดจังหวะแล้วข้ามไปทำงานยังตำแหน่งหนึ่งตำแหน่งใดที่กำหนดไว้ เมื่อทำงานในโปรแกรมขัดจังหวะเสร็จสิ้นก็จะกลับมาทำงานในโปรแกรมที่ตำแหน่งก่อนที่จะไปทำงานยังโปรแกรมขัดจังหวะโปรแกรมที่ถูกกระโดดไปทำงาน เรียกว่า โปรแกรมตอบสนองการขัดจังหวะ [Interrupt Service Routine] ใน 8051 จะสามารถขัดจังหวะด้วยสัญญาณจาก 6 แหล่ง ดังรูปที่ 2.21 ถ้าเป็น 8052 หรือ 83154 จะสามารถขัดจังหวะได้ด้วยสัญญาณจาก 8 แหล่งคือสัญญาณในชุดล่างสุดของรูปที่ 2.21



รูปที่ 2.21 Interrupt Source

สัญญาณขัดจังหวะที่ 5 ในรูปที่ 2.21 จะสามารถทำให้เกิดการขัดจังหวะได้ 2 วิธีคือ มีข้อมูลเข้ามาทางพอร์ทอนุกรมเก็บอยู่ที่รีจิสเตอร์ SBUF และในกรณีที่ข้อมูลใน SBUF ส่งออกไปทางพอร์ทอนุกรมหมดแล้ว ไม่ว่าเกิดกรณีใด ๆ ก็ทำให้เกิดการขัดจังหวะขึ้น

สัญญาณภายนอกที่เข้ามายัง 8051 ทางขา INT0 และ INT1 จะสามารถทำให้เกิดการขัดจังหวะการทำงาน 8051 ได้ (สัญญาณที่ 1 และ 3 ในรูปที่ 2.21) โดยสภาวะของสัญญาณนั้นเปลี่ยนจาก 1 เป็น 0 หรือ เมื่อสัญญาณนั้นเป็น 0 แล้วแต่การกำหนดในบิท IT0 และ IT1 ของรีจิสเตอร์ TCON จะทำให้บิท IE0 กับ IE1 เป็นตัวสร้างสัญญาณขัดจังหวะต่อไป

จาก Timer 0 และ Timer 1 เมื่อค่าการนับในแต่ละโหมดถึงค่าสูงสุดในโหมดนั้นแล้วเมื่อทำการนับต่อไป ค่าการนับต่อไปจะเป็น 0 (หรืออาจเป็นค่าที่ Relode จาก THx ในโหมด 2) และทำให้บิท TF0, TF1 เป็น 1 ซึ่งสัญญาณจาก 2 บิทนี้จะสามารถทำให้เกิดการขัดจังหวะได้เช่นกันดังเช่นสัญญาณขัดจังหวะที่ 2 และ 4 ในรูปที่ 2.21

แหล่งกำเนิดสัญญาณทั้ง 6 ที่สามารถทำให้เกิดการขัดจังหวะได้ 5 แบบนี้ผู้ใช้สามารถกำหนดให้สัญญาณใดบ้างขัดจังหวะเรียกว่า Enable หรือไม่ให้เกิดการขัดจังหวะเรียกว่า Disable โดยการกำหนดในรีจิสเตอร์ IE [Interrupt Enable Register] ซึ่งมี 8 บิท แต่ละบิทสามารถ Enable ให้ขัดจังหวะได้จากแต่ละสัญญาณ ดังรูปที่ 2.22

MSB LSB

EA	X	ET2	ES	ET1	EX1	ET0	EX0

รูปที่ 2.22

SYMBOL	POSITION	FUNCTION
EA	IE.7	disables all interrupts. if IA = 0, no interrupt will be acknowledged if EA = 1 each interrupt source is individually enabled or disabled by setting or clearing its enable bit.
-	IE.6	reverseed
ET2	IE.5	enables or disable the timer 2 overflow or capture interrupt. if ET2 = 0. the serial port interrupt is dissabled.
ES	IE.4	enables or disables the serial port interrupt. if ES = 0. the timer 1 interrupt is disable.
ET1	IE.3	enables or disables the timer 1 overflow interrupt is disabled.
EX1	IE.2	enables or disables external interrupt 1. if EX1 = 0. external interrupt 1 is disables.
ET0	IE.1	enables or disables the timer 0 overflow interrupt. if ET0 = 0. the timer 0 interrupt is disabled.
EX0	IE.0	enables or disables external interrupt if EX0 = 0. external interrupt 0 is disabled.

ถ้าต้องการ Enable บิทใดก็ให้โปรแกรมกำหนดค่าในบิทนั้นเป็น 1 ถ้าค่าในบิทนั้นเป็น 0 หมายถึง Disable การ Disable จะทำให้ไม่มีการขัดจังหวะการทำงานของโปรแกรม เนื่องจากสัญญาณขอขัดจังหวะนั้น ๆ EX0 เป็นชื่อบิท 0 และ EA เป็นชื่อของบิท 7

EX0 บิทนี้ใช้สำหรับการ Enable สัญญาณที่เข้ามาทางขา INT0 ให้เกิดการขัดจังหวะหรือไม่

ET0 Timer 0 Interrupt Enable Bit ข้อมูลบิทนี้จะใช้ Enable หรือ Disable สัญญาณขัดจังหวะจาก Timer 0 [TF1]

EX1 บิทนี้จะใช้ Enable หรือ Disable สัญญาณที่เข้ามาทางขา INT1 ให้เกิดการขัดจังหวะหรือไม่

ET1 Timer 1 Interrupt Enable Bit บิทนี้จะใช้ Enable หรือ Disable สัญญาณขัดจังหวะจาก Timer 1 [TF1]

ES ข้อมูลในบิทนี้จะ Disable หรือ Enable การขัดจังหวะจาก Serial Port อันเนื่องจากมีข้อมูลเข้ามายัง SBUF หรือข้อมูลจาก SBUF ได้ส่งออกไปทาง Serial Port หมดแล้ว

ET2 Timer 2 Internal Enable Bit จะใช้งานเฉพาะใน 8052 และ 83152 เท่านั้นที่บิทจะใช้ Enable หรือ Disable สัญญาณขอขัดจังหวะที่มาจาก Timer 2 (สัญญาณที่ 6 ในรูปที่ 5.11)

EA บิทนี้จะควบคุมทั้ง 6 บิทที่กล่าวมาแล้วถ้าข้อมูลในบิทนี้เป็น 0 จะเป็นการ Disable ทุกบิทที่กล่าวมาแล้ว ทำให้ไม่เกิดการขัดจังหวะโปรแกรมได้เลย แต่ถ้าบิทนี้เป็น 1 การ Enable/Disable ใน 6 บิทที่กล่าวมาแล้วจะขึ้นกับข้อมูลในแต่ละบิทนั้น

หมายเหตุ : บิทอื่นนอกจากนี้จะไม่มีการใช้งาน

การกำหนดให้บิทใด Enable หรือ Disable นั้นจะเป็นไปโดยอิสระไม่ขึ้นแก่กัน จึงสามารถกำหนดให้บิทใดหรือมากกว่า 1 บิท Enable ก็ได้ ดังนั้น 8051 จึงมีรีจิสเตอร์อีกตัวที่ใช้เลือกว่าถ้ามีสัญญาณขอการขัดจังหวะโปรแกรมเข้ามาพร้อมกันมากกว่า 1 แล้วจะทำโปรแกรมตอบสนองการขัดจังหวะอันใดก่อน รีจิสเตอร์นั้นคือ IP Interrupt Pri-ority Registor

การรับ-ส่งข้อมูลทางพอร์ทอนุกรม

ในการรับ-ส่งข้อมูลแบบอนุกรมนั้น จะต้องมีการกำหนดโหมดการทำงานในรีจิสเตอร์ TCON และในบางโหมดของการทำงานจะสามารถกำหนดอัตราการส่งข้อมูลได้โดยการโปรแกรมใน Timer ข้อมูลที่จะส่งออกหรือรับเข้าทางพอร์ทอนุกรมจะอยู่ที่รีจิสเตอร์ SBUF การทำงานของวงจรภายในแต่ละโหมดมีดังนี้

การทำงานโหมด 1

การส่งข้อมูล

จากรูปที่ 5.18 บิท SMOD จะเป็นตัวเลือกว่าสัญญาณ Timer 1 Overflow ที่ส่งไปยังวงจรหาร 16 จะถูกหาร 2 ก่อนหรือไม่ ถ้า SMOD เป็น 1 สัญญาณ Timer 1 Overflow จะถูกหาร 2 ก่อนที่จะเข้าวงจรหาร 16 การส่งข้อมูลจะเริ่มจากการที่มีคำสั่งเขียนข้อมูลไปยังรีจิสเตอร์ SBUF จะมีสัญญาณ Write to SBUF เกิดขึ้นเพื่อรับข้อมูลจาก Internal Bus ด้านบนไปเก็บไปยังรีจิสเตอร์ SBUF และทำให้ Output ของ D Flip Flop ทางซ้ายของ SBUF มีค่าเป็น 1 และเป็นบิทที่ 9 ของการส่งข้อมูล สัญญาณ Write to SBUF ยังส่งไปยัง TX Control ด้วย ขณะนี้ข้อมูลในวงจรหาร 16 มีค่าเป็นอะไรไม่ทราบจึงจะรอจนกว่าข้อมูลในวงจรหาร 16 นับเพิ่มขึ้นจนถึงค่าสูงสุดแล้ววนกลับเป็น 0 คือเกิดการวนกลับทำให้เริ่มการส่งข้อมูลที่เวลา S1P1 ของไซเคิลเครื่องถัดไป (การส่งข้อมูลออกจะสัมพันธ์กับการเกิด Overflow ในวงจรหาร 16) สัญญาณ SEND จาก TX Control เปลี่ยนสภาวะลอจิกเป็น 0 แล้วเริ่มส่งข้อมูลที่เป็น Start Bit (0) ออกไป เมื่อส่งออกไปแล้ววงจร TX Control ก็ทำให้สัญญาณ Data เป็น 1 เพื่อเลื่อนข้อมูลใน SBUF ออกไป เริ่มจากบิท 0 จนถึงบิทที่ 7 การส่งข้อมูลถูกเลื่อนออกไปนั้นจะมี 0 เลื่อนเข้ามาทางซ้ายของรีจิสเตอร์ SBUF เมื่อข้อมูลเลื่อนออกไปทั้ง 8 บิท แล้วบิทที่ 9 ซึ่งเป็น 1 และตอบต้นอยู่ทางซ้ายสุดจะถูกเลื่อนมาอยู่ในตำแหน่งสุดท้ายทางขวาของรีจิสเตอร์ SBUF และทางซ้ายของหลักนี้จะมี 0 อยู่ทั้ง 8 บิทใน SBUF ทำให้ Zero Detector รู้ว่าเป็นข้อมูลบิทสุดท้ายแล้วที่ส่งออกโดยจะมีสัญญาณมาบอกกับวงจร TX Control ด้วย เมื่อ TX Control ส่งสัญญาณ Shift ออกไปเป็นการส่งข้อมูลบิทสุดท้าย (บิท 7) ออกไป ก็จะรออีก 1 TX Clock [Bit Clock] ก็จะทำให้ขา TXD ส่งข้อมูล Stop Bit (1) ออกมา สัญญาณ Data ซึ่งมีสภาวะลอจิกเป็น 1 มาตั้งแต่เริ่มส่งข้อมูลบิท 0 ก็จะกลับเป็น 0 และบิท T1 ก็จะเป็น 1 เพื่อบอกการสิ้นสุดการส่งข้อมูลทั้งหมดเมื่อสัญญาณ TX Clock ไซเคิลที่ 10 นับตั้งแต่สัญญาณ SEND เปลี่ยนสภาวะลอจิกเป็น 0

$$\text{Baud Rate} = \frac{2^{SMOD}}{32} * (\text{Timer 1 Overflow Rate})$$

ในขณะที่ Timer 1 เป็นตัวกำหนด Buad Rate นี้จะต้อง Disable ไม่ให้เกิดการขัดจังหวะเนื่องมาจากการ Overflow Timer 1 อาจใช้ในโหมดของ Timer หรือ Counter ก็ได้ ซึ่งเมื่อการนับโดยรีจิสเตอร์ตัวนับมีค่าสูงสุดแล้วกลับมาเป็น 0 ก็จะเกิด Overflow เช่นเดียวกัน แต่โดยปกติแล้วจะใช้ Timer 1 นี้ในโหมดของ Timer ที่มีการทำงานแบบ Auto Relode โหมด 2 เพื่อว่าเมื่อค่าในการนับโดยรีจิสเตอร์ TL1 ถึงค่าสูงสุดก็จะโหลดค่าในรีจิสเตอร์ TH1 มาไว้ใน TL1 สำหรับเป็นค่าเริ่มต้นการนับต่อไป ซึ่ง Baud rate จะมีค่า

$$\text{Baud Rate} = \frac{2^{SMOD}}{32} \times \frac{\text{Oscillator frequency}}{(12 \times 1256 - [TH\ 1])}$$

โดยที่ SMOD เป็นบิทหนึ่งในรีจิสเตอร์ PCON

เช่นความถี่ของออสซิลเลเตอร์เท่ากับ 11.059 MHz บิท SMOD = 0 รีจิสเตอร์ TH1 มีค่า E8H, Timer 1 ทำงานในโหมด 2 จะได้ว่าอัตราการส่ง-รับข้อมูลแบบอนุกรม

$$= \frac{2^{0}}{32} \times \frac{11.059 \times 10^{5}}{12 \times (256 - 232)}$$

$$= \frac{1}{32} \times \frac{11.059 \times 10}{12 \times 24}$$

$$= 1200 \text{ บิท/วินาที}$$

มาตรฐาน RS-232

พอร์ทอนุกรมส่วนใหญ่ จะมีรูปร่างขึ้นอยู่กับมาตรฐาน RS-232 คือมี 25 ขา (แต่ IBM AT จะมีเพียง 9 ขา) แต่พอร์ทส่วนใหญ่จะมีสัญญาณไม่เหมือนสัญญาณของ RS-232 ทั้งหมดเพราะว่าบางสัญญาณไม่จำเป็นต้องใช้สัญญาณพื้นฐานของ RS-232 แสดงดังตารางที่ 6.1

สัญญาณ	ชื่อย่อ	หมายเลขขา
Request to Send	RTS	4
Clear to Send	CTS	5
Data Send Ready	DSR	6
Data Terminal Ready	DTR	20
Transmit Data	TXD	2
Receive Data	RXD	3
Ground	GND	7

สัญญาณมีทั้งหมดมีมากกว่านี้ เพราะเริ่มแรกนั้น พอร์ทอนุกรมถูกออกแบบมาเพื่อใช้งานร่วมกับโมเด็ม ดังนั้นเมื่อนำไปใช้กับอุปกรณ์อื่น บางสัญญาณจึงไม่จำเป็น เพราะสัญญาณเหล่านี้มีไว้เพื่อเป็นข้อตกลงระหว่างโมเด็ม และคอมพิวเตอร์ว่า

1) คอมพิวเตอร์จะไม่ส่งข้อมูลให้แก่โมเด็ม ก่อนที่โมเด็มจะพร้อมส่งข้อมูล

2) คอมพิวเตอร์จะไม่อ่านข้อมูลจากโมเด็ม ก่อนที่โมเด็มจะพร้อม

Framing Error

คือความผิดพลาด ของการส่งข้อมูลที่เกิดจากสัญญาณนาฬิกา ที่ควบคุมการทำงานของอุปกรณ์ทั้ง 2 ด้านที่ไม่เท่ากัน เพราะจากการทำงานของพอร์ทอนุกรม เมื่อพอร์ทได้รับบิทเริ่มต้นก็จะสุ่มอ่านค่าจากส่วนรับข้อมูล 1 ครั้งต่อ 1 รอบ เพื่ออ่านบิทต่อไป ซึ่งระยะเวลาในการสุ่มอ่านแต่ละรอบกำหนดได้จาก Baud Rate ถ้าหากว่า Computer ทั้ง 2 เครื่องมีสัญญาณนาฬิกาไม่ตรงกัน CPU ทางด้านรับจะอ่านข้อมูลจากส่วนรับข้อมูลของตน ช้าเกินไปหรือเร็วเกินไป ก่อนที่ข้อมูลจะถูกส่งจากคอมพิวเตอร์ด้านส่ง ทำให้เกิด Framing Error ขึ้น

ปัญหาของการสื่อสาร

ในการส่งและรับข้อมูลความจริงแล้วจะต้องใช้ขาสัญญาณหลาย ๆ ขาเพื่อใช้ในการบอกสถานะของฝ่ายส่งและฝ่ายรับว่าพร้อมที่จะรับหรือส่งหรือยัง แต่เพื่อที่จะลดค่าใช้จ่ายเรื่องสายส่ง จะลดขาสัญญาณเหลือเพียง TXD, RXD และ GND (แต่ใน Project นี้จะใช้เฉพาะ TXD กับ GND) ซึ่งจะทำให้เกิดปัญหาการเขียนทับของข้อมูล (Overrun Error)

เมื่อการติดต่อผ่านพอร์ทอนุกรมใช้สายส่งเพียง 2 สายนั้นจะต้องมีกรรมวิธีพิเศษเล็กน้อยเพื่อให้พอร์ทของตัวส่งเข้าใจว่าพอร์ทด้านรับพร้อมที่จะรับข้อมูลตลอดเวลาโดยการต่อขา 6, 8 และขา 20 ของ Connector เข้าด้วยกัน วิธีการเช่นนี้เป็นการตัดฮาร์ดแวร์แฮนด์เชคกิ้งไปแต่การทำเช่นนี้จะทำให้เกิดปัญหาข้อมูลที่ส่งไปชุดก่อนหน้าไม่เสร็จ ทำให้เกิดปัญหาข้อมูลถูกเขียนทับได้ง่าย ซึ่งเกิดจากการส่งข้อมูลชุดใหม่ถูกเขียนทับข้อมูลชุดก่อนหน้า

การใช้งาน Port อนุกรมสามารถทำได้ 3 วิธีด้วยกันคือ

1) ผ่านทางดอส

2) ผ่านทาง Bios

3) การเขียนโปรแกรมควบคุมพอร์ทอนุกรมโดยตรง

หมายเหตุ : 1) การเรียกใช้พอร์ทอนุกรมผ่านทางดอสนั้นไม่เหมาะสมเท่าใดนัก เพราะว่าดอสไม่มีวิธีที่จะตรวจดูสถานะของการรับส่งและตัวพอร์ทอนุกรมว่าการรับส่งข้อมูลถูกต้องหรือไม่เพียงใด

บทที่3

รายละเอียดการสร้างระบบสั่งอาหาร

โมดูลภาคส่ง

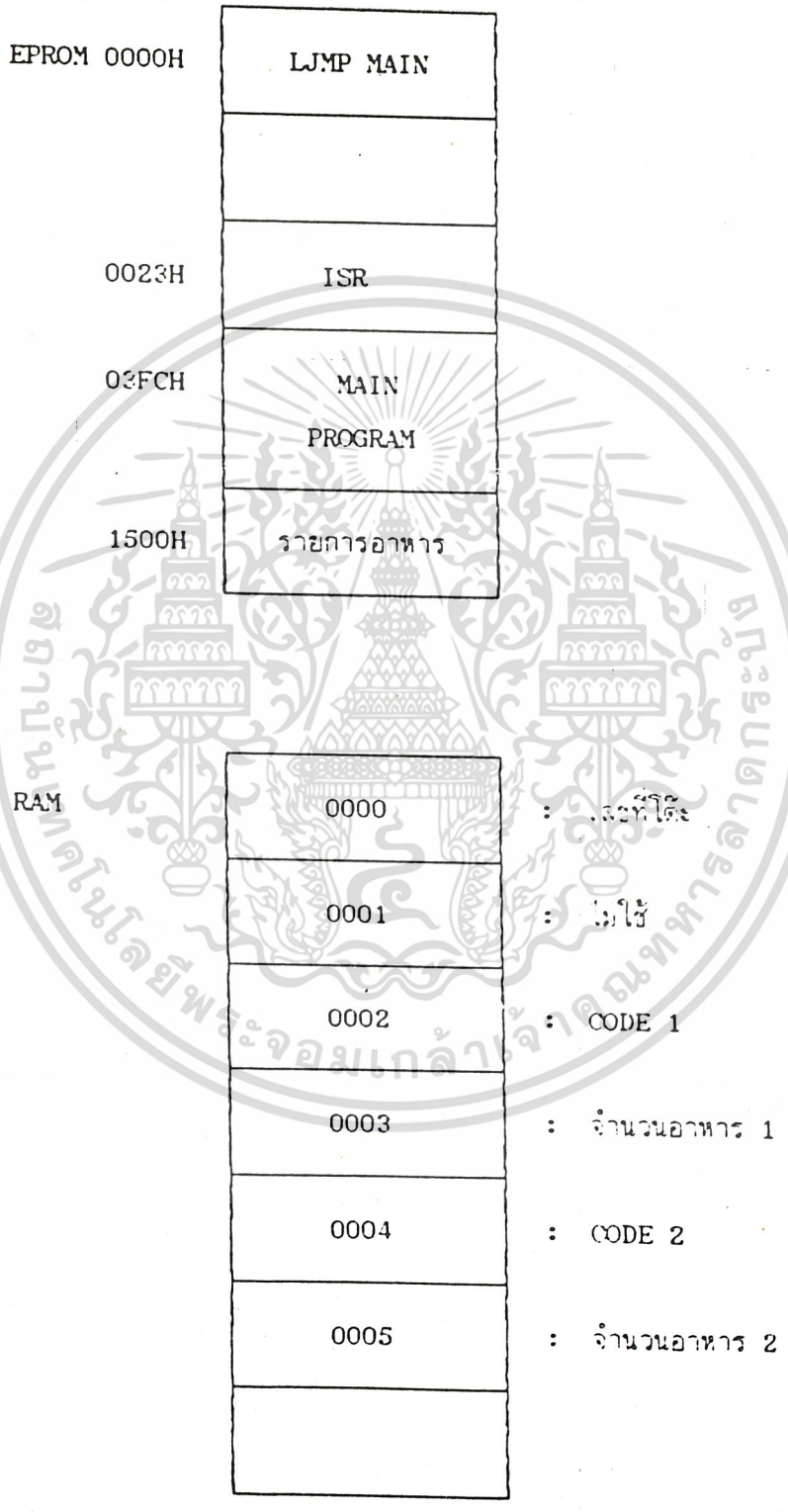
โมดูลภาคส่งประกอบด้วยระบบไมโครคอนโทรลเลอร์ 8031,KEYBORD, LCD DISPLAY และวงจรภาคส่ง ระบบ 8031 ประกอบด้วย ไมโครโปรเซสเซอร์ 8031 หน่วยความจำโปรแกรม EPROM และหน่วยเก็บความจำข้อมูลการสั่งอาหารเก็บไว้ใน RAM 8031 จะรับข้อมูล input จากคีย์บอร์ดนำมาประมวลผลเก็บข้อมูลไว้ใน RAM ตลอดจนส่งข้อมูลผ่านไปยังวงจรภาคส่งต่อไป การสั่งอาหารเริ่มตั้งแต่มีการป้อนข้อมูลเข้าทางคีย์บอร์ด โดยการทำงานของวงจรคีย์บอร์ดจะสัมพันธ์กับโปรแกรมในส่วนของการ SCANKEY คือเริ่มแรก 8031จะส่งข้อมูลออกทางพอร์ต 1 แล้วไปพักข้อมูลไว้ที่ IC 373 รูปแบบข้อมูลจะมี LOGIC เป็น LOW 1 bit นอกนั้นเป็น HIGH โดย LSB จะเป็น bit LOW ทั้งนี้จะเป็นการกำหนดแถวที่จะSCAN นั่นเอง LSB คือแถวบนสุดของคีย์บอร์ด ขณะที่พอร์ต 1 จะกำหนดทุก bit เป็นHIGH หากมีการกดคีย์ที่ colume ใด ค่า LOGIC LOW จะปรากฏที่ bit ใด bit หนึ่งของพอร์ต 1 และเมื่อ 8031 อ่านค่ากลับเข้าไปจะทำให้สามารถทราบว่ามีการกดคีย์ใด ข้อมูลที่ CPU ได้รับเข้ามาจะถูกนำไปประมวลผลตามโปรแกรม และนำข้อมูลไปเก็บไว้ในหน่วยความจำชั่วคราวและยังมีการนำข้อมูลไปแสดงผลบนจอ LCD การติดต่อกับ LCD จำเป็นที่จะต้องมีตัว drive LCD ซึ่งจะเป็นตัวพักข้อมูลไว้ โดยใช้ IC เบอร์ 373 นอกจากนี้จะมีการส่งสัญญาณควบคุมการเขียนอ่านคำสั่ง และข้อมูลที่จะแสดงผลผ่านพอร์ตของ 8031 โดยตรง

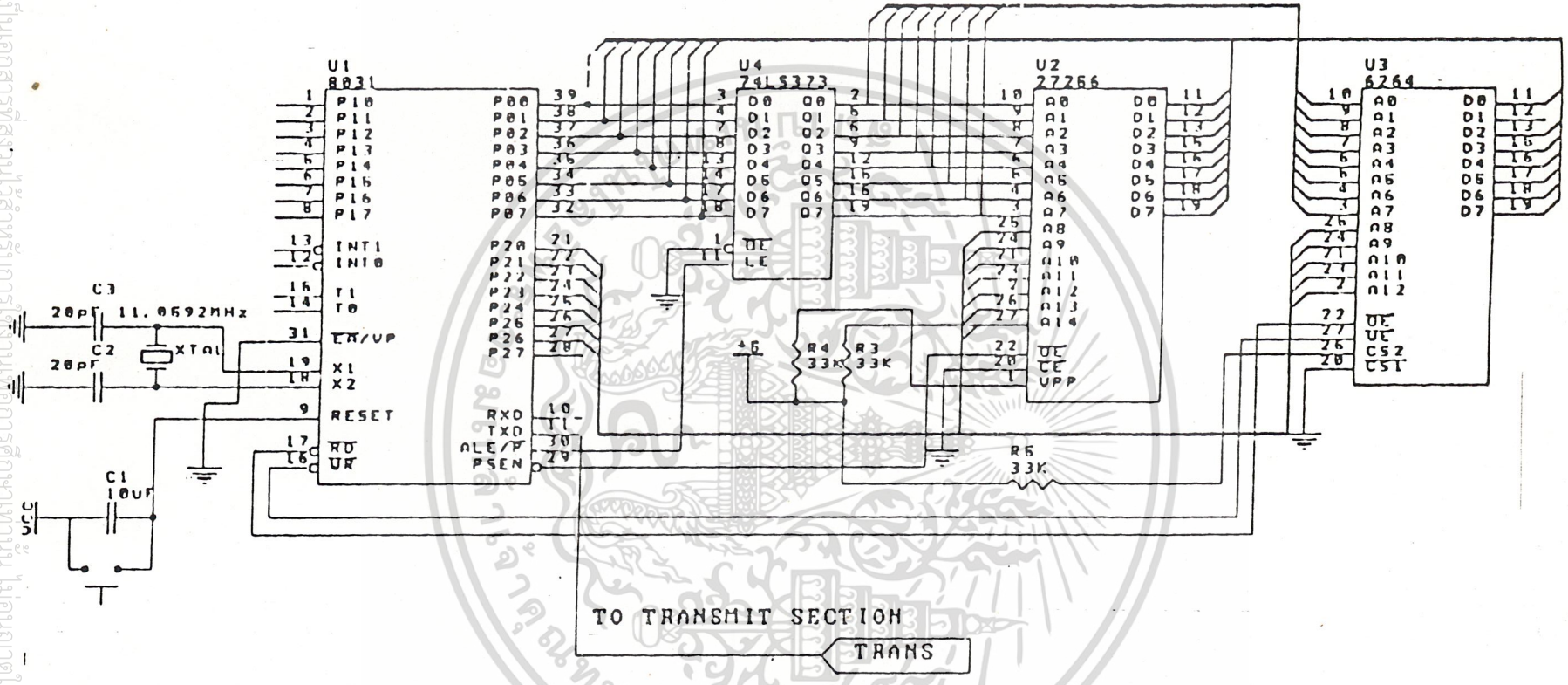
Memory Map ของ Module ตัวส่งมีดังนี้ EPROM จะบรรจุโปรแกรมการทำงานทั้งหมดไว้ โดย Main Program เริ่มจาก address 03FCH รายการอาหารที่มีอยู่เก็บไว้ใน address ตั้งแต่ 1500H เป็นต้นไป address 0023H เป็น interupt vecter สำหรับการส่งข้อมูลผ่านทางพอร์ตอนุกรม RAM เป็น

ตัวเก็บข้อมูลรายการอาหารที่สั่งเริ่มตั้งแต่ address 0000H โดยตำแหน่ง 0001H ไม่ใช้จะข้ามไปที่ตำแหน่ง 0002H ซึ่งใช้เก็บรหัสอาหาร ตำแหน่งถัดไปเก็บจำนวนอาหาร ถัดมา address 0004H จะเก็บรหัสอาหารรายการที่ 2 เป็นอย่างนี้เรื่อยไปจนสิ้นสุดรายการอาหารที่สั่ง

ในส่วนวงจรภาคส่งข้อมูลผ่านทาง infrared นั้น ข้อมูล data ที่ส่งจะถูกนำไป modulate แบบ amplitude modulation กับสัญญาณพัลซ์ที่ oscillate จากวงจร oscillate ซึ่งประกอบด้วย IC 7404, R และ C ความถี่ในการ oscillate มีค่าประมาณ 20 kHz สัญญาณ modulate ที่ได้จะนำไปขับตัว IR เพื่อส่งต่อไป

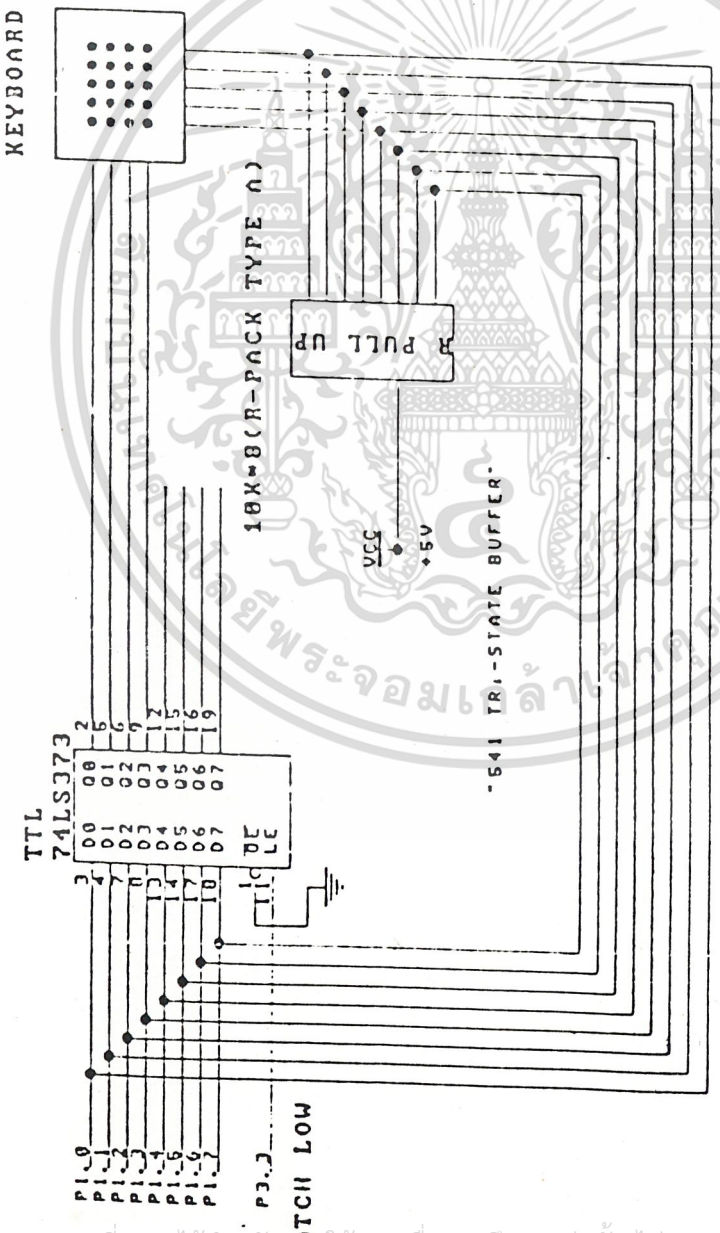
Memory Map





8031 SYSTEM

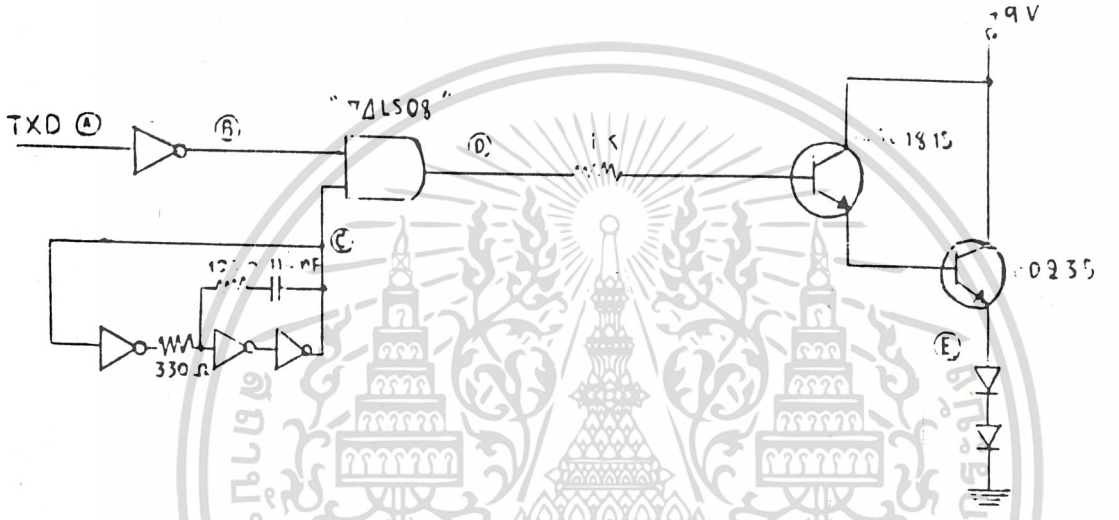
Title		
Size A4	Number	
Date: 11-MAR 1994		Sheet
File: B:PROJECT3/1		Drawn B



The connection of keyboard

การส่งข้อมูลผ่านทาง Infrared

เนื่องจากการใช้อินฟาเรดเป็น Carrier เราจะต้องใช้ความถี่ประมาณ 20 KHz ในการ Switching ทรานซิสเตอร์เพื่อขับตัวส่งอินฟาเรด แต่ข้อมูลที่ออกจาก Serial Port (TXD) มี Baud Rate 9600 Baud จะเห็นว่าความถี่ที่เป็นไปได้สูงสุดแค่ 4800 Hz เพราะฉะนั้นเราต้อง MOD Carrier เข้าไปกับข้อมูลในการ MOD เราจะนำข้อมูลที่ออกมาจาก TXD มา and กับ carrier ดัง ..สดงไว้ดังรูป

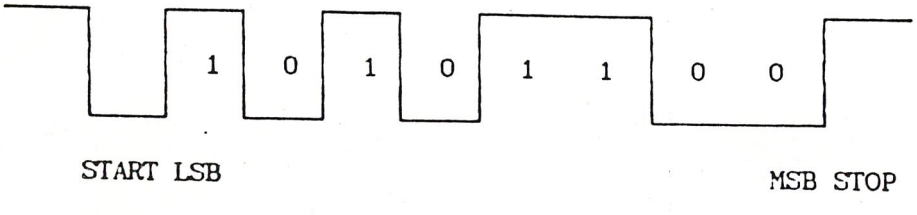


ในการ MOD จะใช้ความถี่ประมาณ 20 KHz ซึ่งมาจาก RC Oscillator ความถี่ที่ได้ เท่ากับ

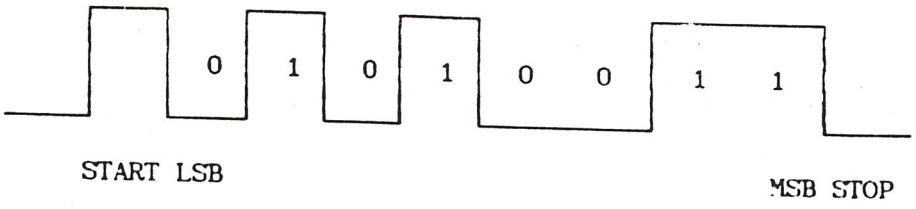
$$F_o = \frac{1}{2C[0.41R_p + 0.7R_1]} \quad ; \quad R_p = \frac{R1 * R2}{R1 + R2}$$

สัญญาณที่จุดต่าง ๆ แสดงได้ดังนี้

A สัญญาณที่ออกจาก TXD สมมุติว่าเป็น 35H

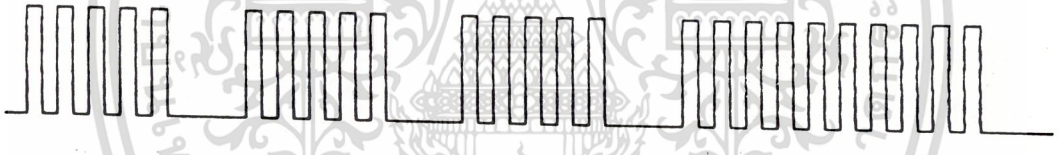


B ข้อมูลที่ผ่าน Not Gate



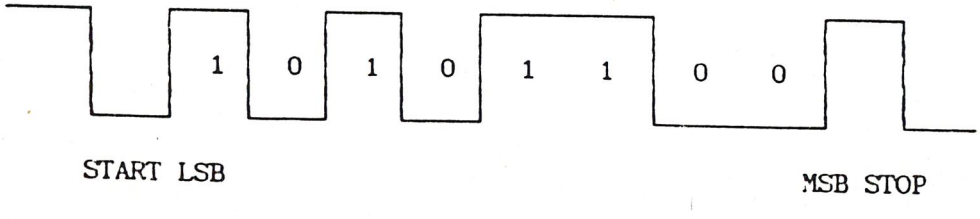
C Carrier

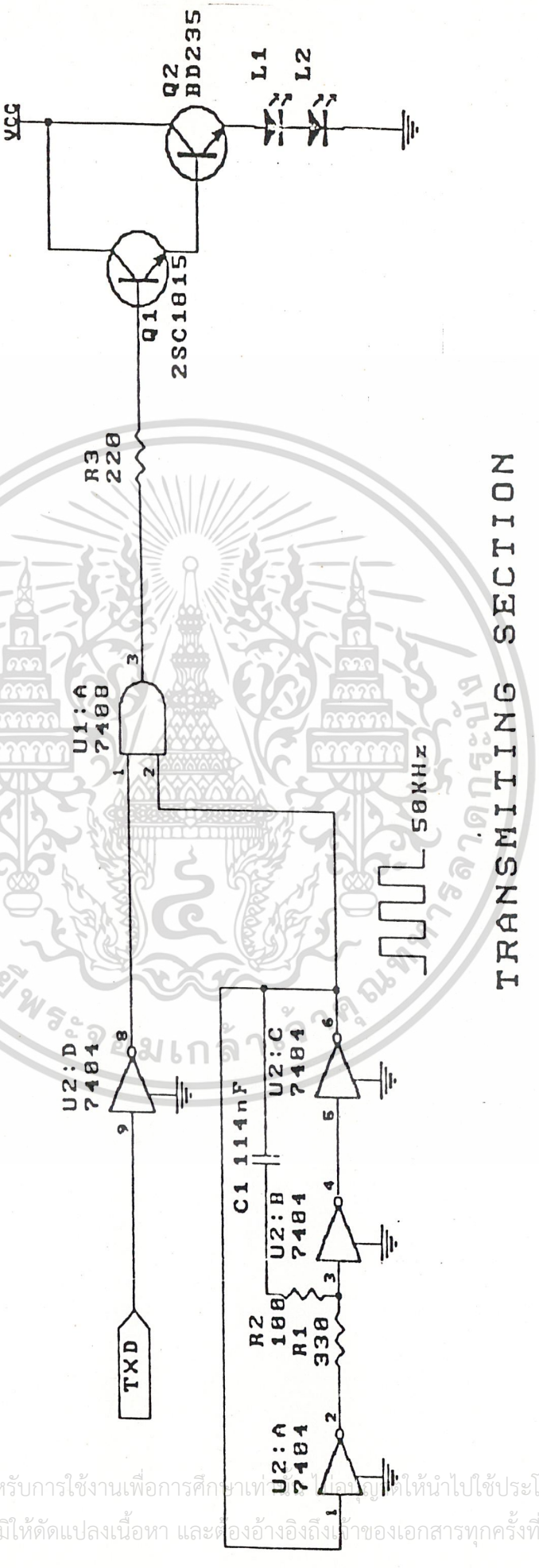
D สัญญาณที่ผ่านการ MOD แล้ว



E สัญญาณที่ส่งออกจากตัวส่ง

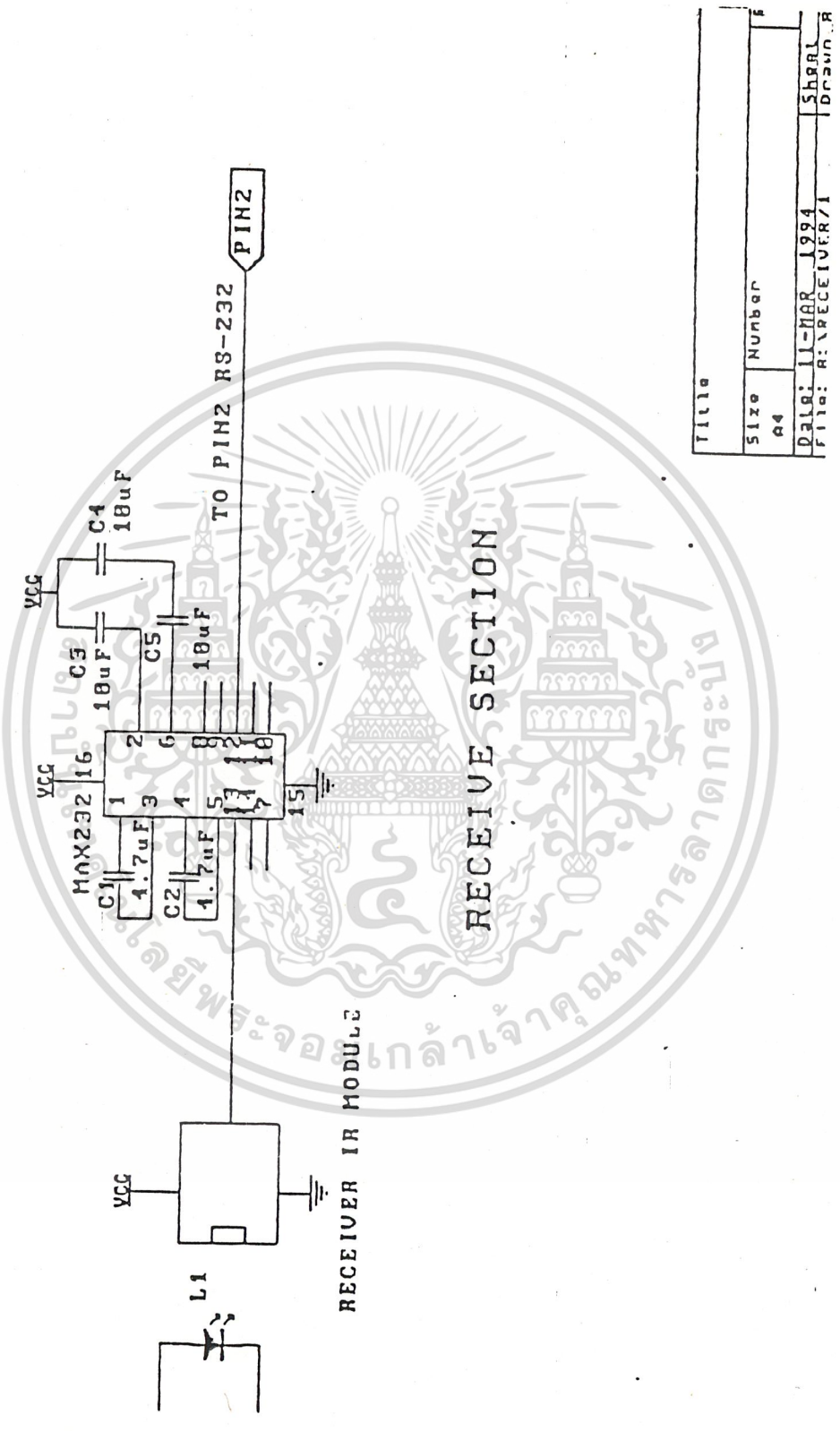
F สัญญาณที่ได้จากตัวรับ



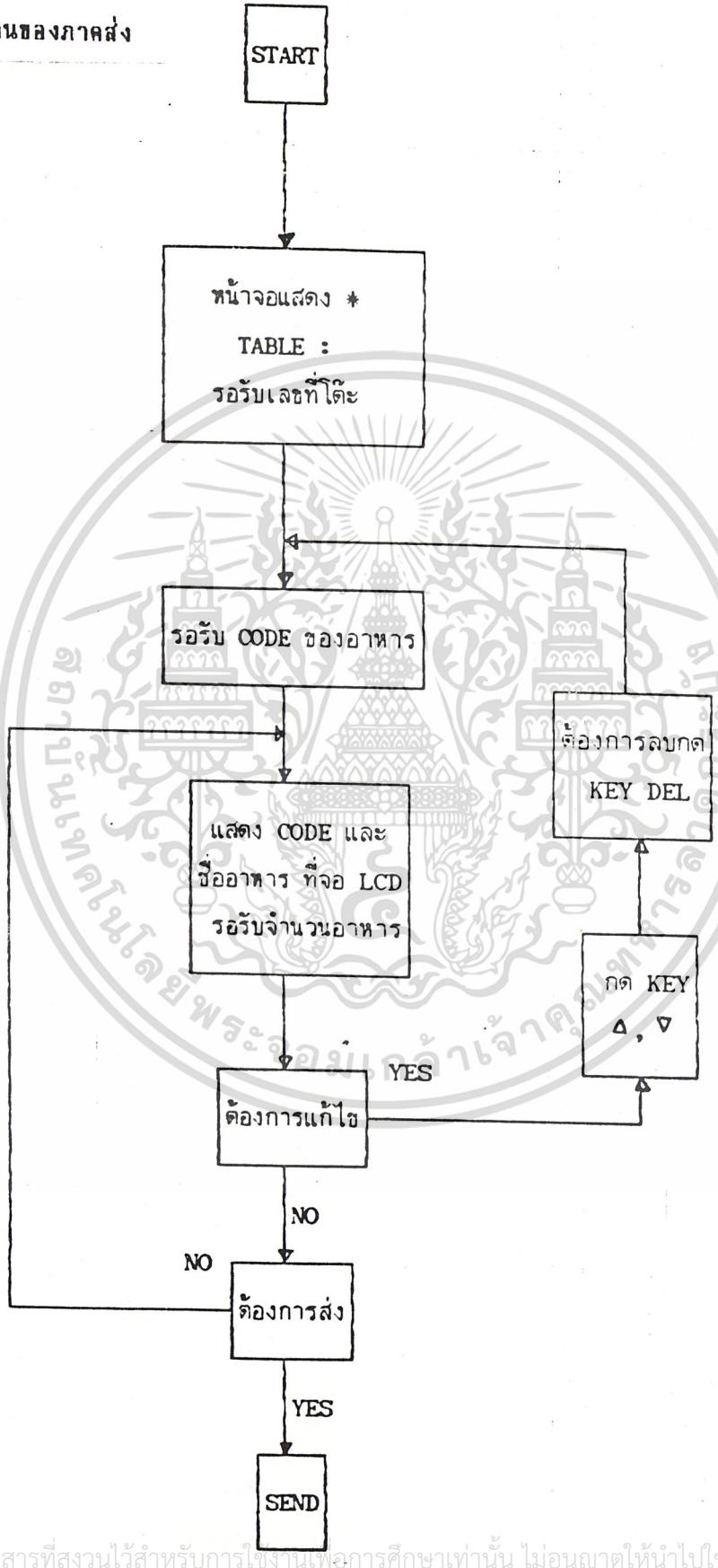


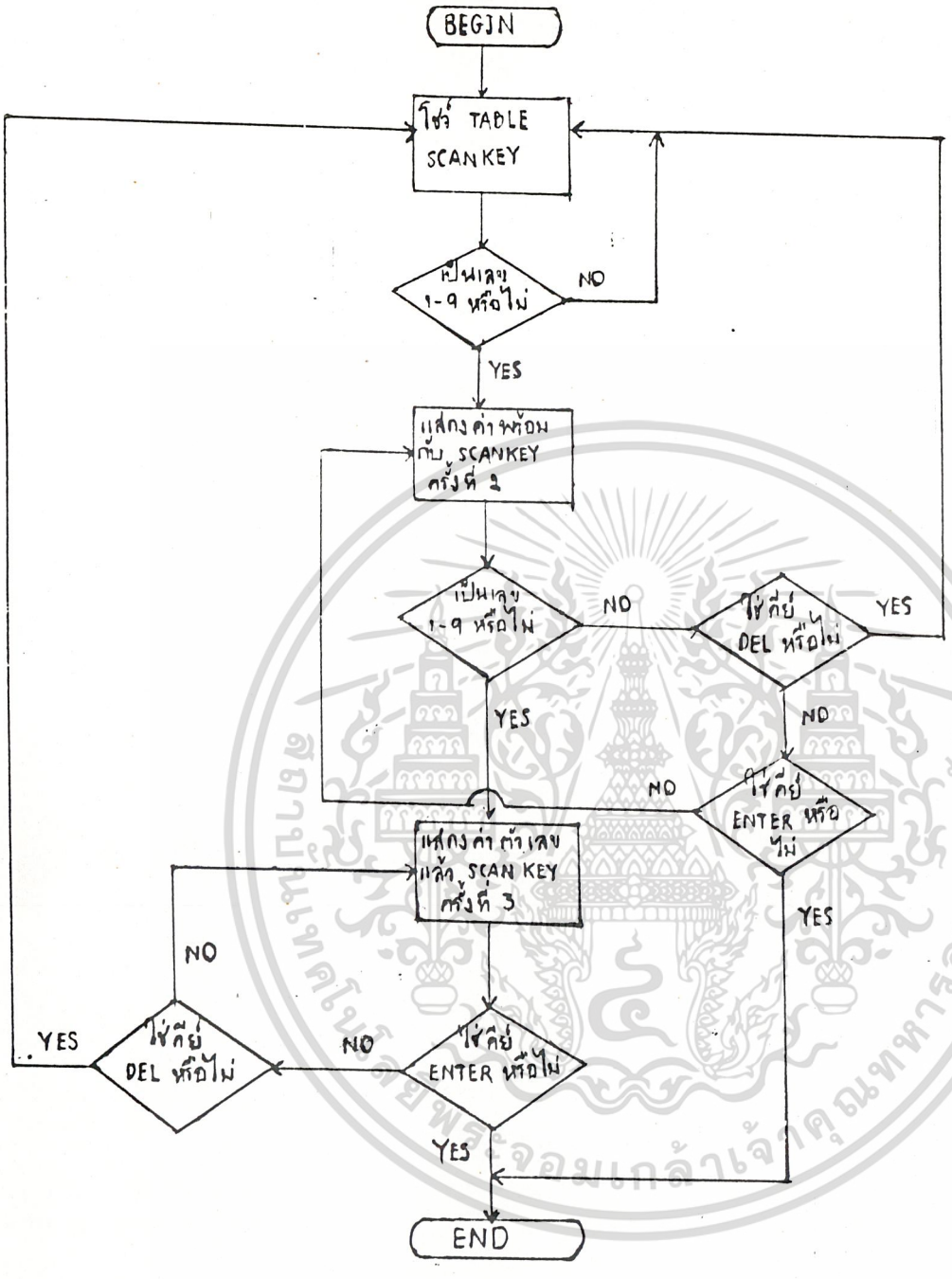
โมดูลภาครับ

โมดูลภาครับประกอบด้วย ตัวรับสัญญาณอินฟราเรด เป็นโมดูลสำเร็จรูป โดยใช้ โฟโตทรานซิสเตอร์พร้อมกับมีวงจรกรองสัญญาณในตัวทำให้ได้ข้อมูลเหมือนกับข้อมูลก่อนผ่านการ modulate ในการส่งข้อมูลนี้จะถูกนำไปผ่าน IC MAX 232 เพื่อยกระดับสัญญาณข้อมูล ให้อยู่ในระดับที่เหมาะสม ตามที่กำหนดไว้ในมาตราฐาน RS-232 สัญญาณ input เมื่อเข้าตัว MAX 232 จะผ่าน NOT GATE output ที่ได้จะส่งผ่านสาย RS-232 เข้าสู่ PC ต่อไป เพราะฉะนั้นที่โมดูลภาคส่งจึงต้องมีการ inverse สัญญาณข้อมูลก่อนส่งออกทางอินฟราเรด โดยผ่าน NOT GATE เช่นกัน



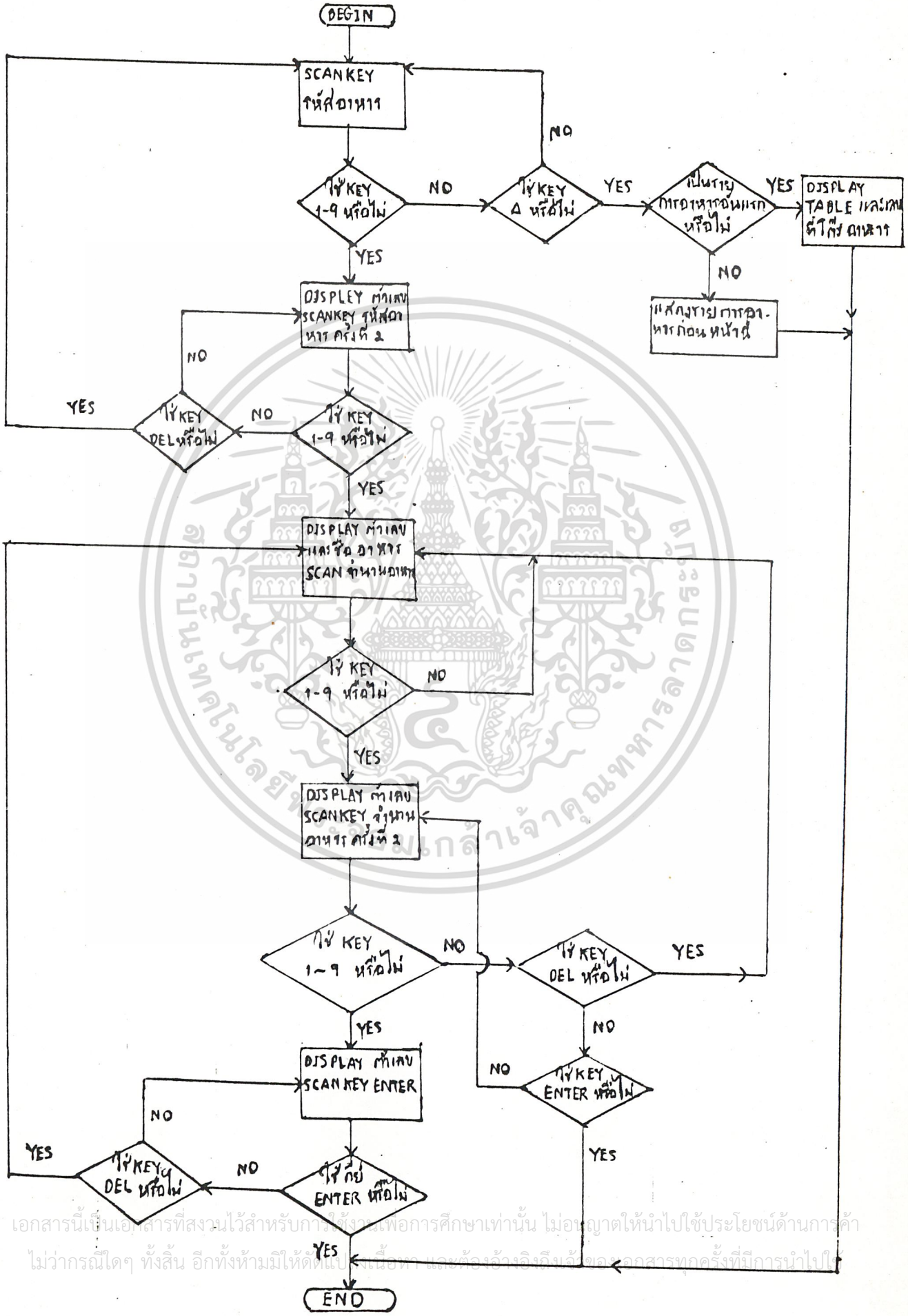
ขั้นตอนการทำงานของภาคส่ง

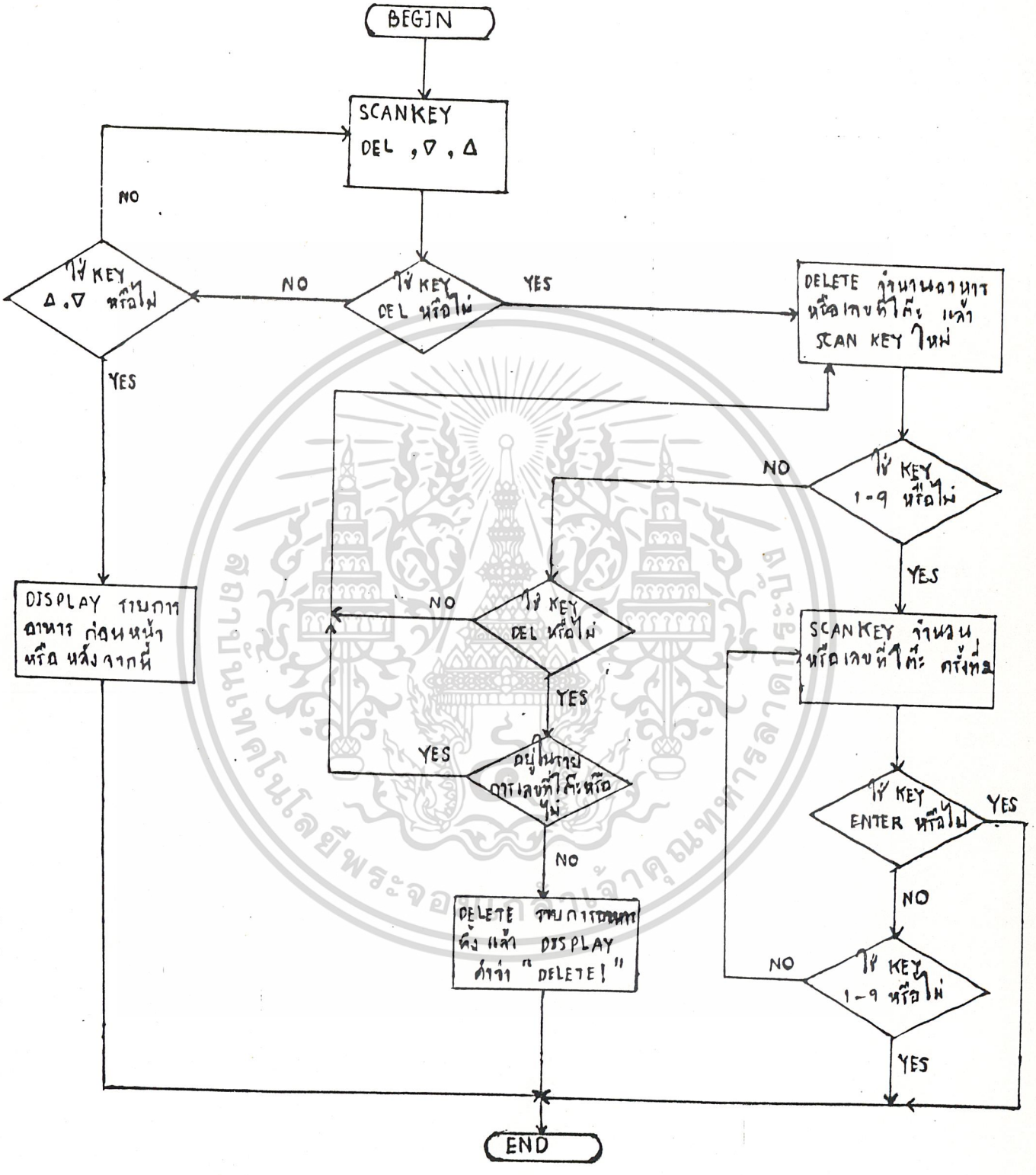




โฟลชาร์ทการทำงานของเครื่องส่งโดยละเอียด

รับ CODE อาหารและจำนวน

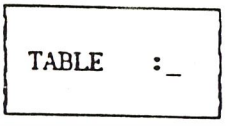




การแก้ไขรายการอาหาร หลังจาก กด KEY △ ดูรายการอาหารที่สั่งไปแล้ว

การทำงานภาคส่ง

เริ่มต้น เปิดเครื่อง หรือ Reset ที่จอ LCD จะปรากฏ

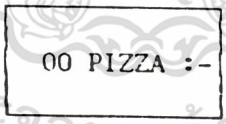


รูปที่ 1

Cursor จะกระพริบหลัง : เป็นการรอการป้อนเลขที่โต๊ะ หลังจากป้อนเลขที่โต๊ะแล้วหน้าจอ LCD จะ Clear . Cursor จะกระพริบอยู่ซ้ายสุด ซึ่งรอรับ Code ของอาหาร

รูปที่ 2

หลังจากป้อนรหัสของอาหารแล้วหน้าจอจะ Show รหัสของอาหารและตามด้วยชื่ออาหาร แล้ว Cursor จะกระพริบอยู่หน้า : ซึ่งรอรับจำนวนที่จะสั่ง



รูปที่ 3

พอป้อนจำนวนอาหารแล้วหน้าจอจะ Clear ดังรูปที่ 2 เพื่อรอรับรหัสอาหารชุดต่อไป นอกจากนี้ยังมี Key UP y, Down v เพื่อเลื่อนไปดูรายการที่สั่งแล้ว ถ้าต้องการลบก็ใช้ Key y, v เลื่อนไปหารายการอาหารนั้นแล้วกด Key DEL จะเห็นว่าจำนวนของอาหารจะหายไปซึ่งเราสามารถใส่จำนวนอาหารใหม่เข้าไปได้ แต่ถ้าไม่ใส่จำนวนอาหารใหม่เข้าไป แต่กด DEL ซ้ำอีกครั้ง รายการอาหารนั้นจะถูกลบออกไป โดยที่หน้าจอ LED จะกระพริบ 2 ครั้ง แล้วแสดงคำว่า Delete เป็นเวลาประมาณ 30 วินาที แล้วจะโชว์รายการอาหารถัดไป

หลังจากแก้ไขเรียบร้อยแล้วถ้าต้องการส่งก็กด Key Send ซึ่งหน้าจอจะโชว์

TOTAL ORDER : en

: e คือ จำนวนรายการทั้งหมด

ข้อมูลก็จะถูกส่งออกไป

ตำแหน่ง Key

△	▽	SEND	RESET
0	1	2	3
4	5	6	7
8	9	ENTER	DEL

รายการอาหาร

CODE	
00	Pizza
01	Slad
02	Steak
03	Hotdog
04	Hamberger
05	Chicken Pie
06	Apple Pie
07	Onion Soup
08	Donut
09	Hot Coffee
10	Ice Coffee
11	Ice Tea
12	Coca Cola
13	Punch
14	Juice
15	Milk
16	Beer

บทที่4

การทดสอบ

โปรแจคนั้นต้องต่อวงจรตามรูป 8031 SYSTEM ซึ่งใช้ 8031 เป็น CPU มี EPROM, RAM, LCD และ KEYBOARD

วิธีการทดสอบ HARDWARE

ทำได้โดย RAMPACK หรือ EPROM EMULATOR เก็บโปรแกรมแทน EPROM โดยเขียนโปรแกรมทดสอบ HARDWARE ง่ายๆ คือ เขียนโปรแกรมให้นำข้อมูลไปเก็บไว้ที่ RAM ADDRESS ไหนก็ได้ แล้ว MOVX ค่านั้นกลับมาโดยส่งออก PORT 1 ของ 8031 แสดงได้ดังนี้

```
        ORG     0000H
        CPU     "8031.TBL"
        MOV     DPTR,#0000H
        MOV     RO.#3000H
        MOV     A,#55H
RELOAD :MOVX    @DPTR,A  ;นำค่า 55H ไปเก็บไว้ใน RAM
                         ;ตั้งแต่ ADDRESS 0000H-3000H
        INC     DPTR
        DJNZ    RO,RELOAD
        MOV     DPTR,#1500H
        MOVX    A,@DPTR ;ส่งค่าที่ ADDRESS 1500H
                        ;ไปไว้ที่ REGISTER A
        MOV     P1,A    ;OUT ค่าใน ADDRESS 1500H
                        ;ไปที่ PART 1 ของ 8031
        END
```

หลังจาก RUN PROGRAM แล้วใช้ LOGIC PROBE วัดที่ PORT 1 ของ 8031 จะต้องได้ค่าเป็น 01010101

การทดสอบโปรแกรมรับ

เมื่อเราเขียน Program รับเรียบร้อยแล้ว จะมีวิธีทดสอบโดยใช้ EIT Board เป็นตัวส่งข้อมูลเข้ามา ในขณะที่เรา Run Program รับรออยู่ ต้องกำหนด Baud Rate ให้ตรงกันด้วย หลังจากส่งข้อมูลมาแล้ว ถ้าโปรแกรมรับสามารถรับได้จะนำข้อมูลที่ได้ไปเก็บไว้ที่ File in_fi.dat ซึ่งเราสามารถใช้คำสั่ง Type ดูได้ว่าตรงกับข้อมูลในหน่วยความจำของ EIT Board ที่ส่งมาหรือไม่ ถ้าตรงแสดงว่าโปรแกรมรับถูกต้อง

การทดสอบโปรแกรมส่ง

หลังจากทราบว่าโปรแกรมรับเรารับได้แล้วก็ทดสอบโปรแกรมส่งของเรา เริ่มแรกใช้สายต่อจาก TXD ผ่าน MAX232 เข้าไปยัง RS-232 แล้วใช้คำสั่ง TYPE ดู File in_fi.dat ว่าตรงกับข้อมูลที่เราส่งไปหรือไม่ ถ้าข้อมูลที่ได้ถูกต้อง เราก็ทำการเขียนโปรแกรมต่อโดยนำ File in_fi.dat ไป Decode เพื่อจะแสดงออกทางหน้าจอว่ามีข้อมูลอะไรมาบ้าง ในการ Decode เราจะตกลงกันระหว่างตัวส่งกับตัวรับไว้ว่า Byte แรกที่มาคือ เลขที่โต๊ะ Byte ต่อมาจะไม่ใช้ Byte ต่อมาจะเป็น Code ของอาหาร เช่น "00" คือ Pizza , "01" คือ Salad และ Byte ต่อมาคือ จำนวนของอาหารอย่างแรก และ Byte ต่อไปคือ Code ของอาหารอย่างที่สอง เป็นอย่างนี้เรื่อย ๆ จนหมดข้อมูล และ Byte สุดท้ายที่ส่งมาจะเป็น FFH เพื่อบอกให้ตัวรับทราบว่าสิ้นสุดการส่ง

ในการทดสอบการส่ง Infared ก็ทดสอบโดยป้อนไฟให้ตัวรับก่อนแล้วส่งข้อมูลไปโดยใช้ Osciloscope จับดูที่ขา Signal ของตัวรับว่ามี PULSE มาหรือไม่ ถ้ามี PULSE มา ทดลองต่อขา Signal เข้ากับ MAX232 แล้วต่อไปยัง RS232 หลังจากนั้นลอง Run โปรแกรมรับ แล้วทำการส่งดูตรวจสอบ Output ที่ File in_fi.dat ว่ามีความผิดพลาดหรือไม่

ปัญหาที่พบและการแก้ไข

1. ในการต่อวงจร SCANKEY โดยใช้ 74HCT373 เป็นตัวLATCH ข้อมูลนั้น หากต่อ PORT1 ของ 8031 เข้ากับขา 3,4,7,9 ของ 373 และต่อ KEYBOARD เข้ากับขา 2,5,6,7 การทำงาน SCAN ข้อมูลจะพิมพ์จากแก้ไขโดยการต่อ KEYBOARD เข้ากับขาก 12,15,16,19 การทำงานจึงจะถูกต้อง

2. ในการส่งและรับข้อมูลยาวๆซึ่งมีสัญญาณรบกวน (NAOSE) จากภายนอกจะทำให้ตัวรับๆข้อมูลในขณะที่ไม่ได้ส่งข้อมูลไป วิธีแก้คือ กำหนดให้ข้อมูลตัวแรก เป็นตัวบอกว่าจะมีข้อมูลส่งตามมา และสุดท้ายจะมี CODE บอกการสิ้นสุดของข้อมูล จะทำให้การรับส่งข้อมูลแม่นยำยิ่งขึ้น

ในที่นี้ จะใช้ 99H เป็นตัวแรก และ FFH เป็นตัวสุดท้าย

3. ในการส่งด้วย INFARED สำคัญที่สุด คือส่วน OSCILLATOR ที่จะนำไปตอบกับข้อมูลที่ออกจาก TXD ต้องใช้ FUNCTION GEN ปรับความถี่ไปเรื่อยๆ เริ่มที่ 15 KHz แล้งทดลองส่ง จะได้ความถี่ที่ 20 KHz ตัวรับจะรับได้ชัดที่สุด

4. ในการส่งถ้าใช้ BAUD RATE สูงจะทำให้การรับข้อมูลผิดพลาดได้สูง จึงต้องลด BAUD RATE ในที่นี้ใช้ 1,200 BAUD

ข้อที่ควรนำไปปรับปรุงพัฒนา

ปรับปรุงให้เพิ่ม BAUD RATE ได้สูงกว่านี้และให้ได้ไกลกว่าเดิม การที่จะทำให้ BAUD RATE สูงและแม่นยำ จะต้องพัฒนาทั้งภาค HARDWARE คือปรับปรุงภาค MODULATION และวงจรส่งและหาตัวส่ง IR ที่มีประสิทธิภาพสูง ส่วนในภาค SOFTWARE อาจจะมีการส่งข้อมูลซ้ำๆ กันหลายๆรอบ แล้วโปรแกรมรับก็ทำการรับแล้วนำข้อมูลเปรียบเทียบกันหาข้อมูลที่ถูกต้องที่สุดนำมา

ใช้ในการประมวลผล

เนื่องจากรีโมตนี้เป็นการส่งรับด้วย INFRARED ซึ่งเมื่อนำไปใช้งานจริง จำเป็นต้องมีเครื่องส่งหลายเครื่อง ซึ่งหากมีการส่งพร้อมกันไปยังเครื่องรับเครื่องเดียวกัน จะมีการรบกวนกันของสัญญาณ ทำให้ข้อมูลที่รับได้ผิดพลาด ซึ่งในจุดนี้ผู้ที่สนใจโครงงานนี้ควรนำไปพิจารณาด้วย

ทางด้าน HARDWARE เราสามารถปรับปรุงให้สามารถลดการกินกระแสลงไปได้อีก โดยอาจใช้แนวคิดแบบวงจรรีโมตทีวี โดยมีการกินพลังงานเฉพาะเมื่อเวลากด KEY แต่ควรคำนึงไว้ว่าการปรับปรุงวงจรให้ซับซ้อนยิ่งขึ้นจะทำให้ขนาดของเครื่องส่งมีขนาดใหญ่ ไม่เหมาะในการนำใช้งาน

บทสรุป ทำอะไรบ้าง และได้ผลอย่างไร

ในการทำ PROJECT ชิ้นนี้ส่วนสำคัญคือ SOFTWARE ของทั้งภาครับและภาคส่ง เราได้พัฒนาทั้งด้าน HARDWARE และ SOFTWARE ทำให้กินกระแสในปริมาณที่น้อย ในเครื่องส่งมีฟังก์ชันการทำงานต่างๆที่จำเป็นต่อการสื่อสารอยู่ครบครัน ตลอดจนมีการแสดงผลโดยใช้ LCD ซึ่งเป็นอุปกรณ์ที่มีประสิทธิภาพสูงและทันสมัย ทางด้านรับมีระบบการรับข้อมูลที่น่าเชื่อถือได้ และสามารถจัดเก็บข้อมูลเพื่อทำการวิเคราะห์ทางสถิติตลอดจนมีรูปแบบ menu ที่ง่ายต่อการใช้งาน

ในการส่งสัญญาณคลื่น INFARED นี้เราจำเป็นที่จะต้องมีการ ENCODE สัญญาณไว้ด้วย เพื่อไม่ให้เกิดความสับสน เนื่องจากมี REMOTE ของอุปกรณ์หลายชนิดซึ่งอาจจะเข้ามารบกวนภาครับของเราได้

บรรณานุกรม

1. Blue Ridge Summit,PA[1990], "Electronic Circuit Volume 2" TAB, BOOKS Inc.

2. Kenneth J. Ayala [1991], " 8051 Microcontroller ", West Publishing Company.

3.Herbert Schildt [1988], "C Power User Guide", McGraw - Hill Inc.

ภาคผนวก

-รายการอุปกรณ์ที่ใช้

-โปรแกรมภาคส่ง

-โปรแกรมภาครับ

รายการอุปกรณ์ที่ใช้

ภาคส่ง

1. 80C31 CPU	1	ตัว
2. LCD MODULE 1 แถว 16 ตัวอักษร	1	ตัว
3. 74HCT373	3	ตัว
4. 74HCT04	1	ตัว
5. 74HCT08	1	ตัว
6. KEYBOARD 16 KEYS	1	ตัว
7. 6264 RAM	1	ตัว
8. 27265 EPROM	1	ตัว
9. 7805	1	ตัว
10. XYAL 110.52 MHz	1	ตัว
11. CAPACITOR 30 pF ceramic	2	ตัว
12. CAPACITOR 10 uF electrolyte	1	ตัว
13. CAPACITOR 47 uF electrolyte	1	ตัว
14. DIODE 1N4001	1	ตัว
15. R 10K 1/4	7	ตัว
16. VR 10K	1	ตัว
17. TRANSISTOR 2SC1815	1	ตัว
18. TRANSISTOR BD235	1	ตัว
19. ตัวส่ง IR	1	ตัว

ภาครับ

1. ตัวรับ IR MODULE SONY	1	ตัว
2. MAX232	1	ตัว
3. CAPACITOR 4.7 uF electrolyte	2	ตัว
4. CAPACITOR 10 uF electrolyte	3	ตัว
5. สาย RS232 พร้อม CONNECTOR	1	ตัว

โปรแกรมภาคส่ง

```
000000                          ORG     0000H
000000                          CPU     "8051.TBL"
                                ;
00008000    =                   PDAT:   EQU     8000H
00008001    =                   PSIG    EQU     8001H
00008003    =                   PCON:   EQU     8003H
0000002F    =                   COUNT1: EQU     2FH
0000002E    =                   COUNT2: EQU     2EH
00000083    =                   DPH:    EQU     83H
00000082    =                   DPL:    EQU     82H
000000E7    =                   ACC.7:  EQU     0E7H
00000098    =                   SCON:   EQU     98H
00000099    =                   SCON.1: EQU     99H
00000099    =                   SBUF:   EQU     99H
0000008D    =                   TH1:    EQU     8DH
0000008B    =                   TL1:    EQU     8BH
00000089    =                   TMOD:   EQU     89H
00000088    =                   TCON:   EQU     88H
000000A8    =                   IE:     EQU     0A8H
00000087    =                   PCON.7: EQU     87H
000000E0    =                   A:      EQU     0E0H
000000E2    =                   ACC.2:  EQU     0E2H
00000081    =                   SP:     EQU     81H
00000090    =                   P1:     EQU     90H
000000B4    =                   P3.4:   EQU     0B4H
00000000    =                   R0:     EQU     00H
00000007    =                   R7:     EQU     07H
00000006    =                   R6:     EQU     06H
000000F0    =                   B:      EQU     0F0H
                                ;
```

```
000000 02002E                    LJMP    MAIN
                                 ;
                                 ;This is a serial interrupt
000023                           ORG     0023H
                                 ;
000023 109903                    JBC     SCON.1,TRANS
000026 02002D                    LJMP    EXIT
000029 E0              TRANS:    MOVX    A,@DPTR
00002A F599                      MOV     SBUF,A
00002C A3                        INC     DPTR
00002D 32              EXIT:     RETI
                                 ;
                                 ;Main program begin here
                                 ;
00002E 758130          MAIN      MOV     SP,#30H
000031 751700                    MOV     17H,#00H
000034 751800                    MOV     18H,#00H
000037 120130                    LCALL   INITLCD
00003A 9004F4                    MOV     DPTR,#TAB
00003D 120190                    LCALL   DISP
000040 1201DA                    LCALL   ENTDAT
000043 900000                    MOV     DPTR,#0000H
000046 F0                        MOVX    @DPTR,A
000047 752F01                    MOV     COUNT1,#01H
00004A 752E01                    MOV     COUNT2,#01H
00004D 7F00                      MOV     R7,#00H
00004F 7E02                      MOV     R6,#02H
000051 8F18                      MOV     18H,R7
000053 8E17                      MOV     17H,R6
000055 120130                    LCALL   INITLCD
000058 1200F1          ORI:      LCALL   SCAN
00005B 120256                    LCALL   CHECK
00005E BF0003                    CJNE    R7,#00H,CHUP
```

```
000061 02005A            LJMP    NEXTL
000064 B80CF1  CHUP:     CJNE    R0,#0CH,0RI
000067 020098            LJMP    COM1
00006A 120268  NEXTL:    LCALL   ONELINE
00006D E57F              MOV     A,7FH
00006F B4000F            CJNE    A,#00H,COMBACK
000072 E57E              MOV     A,7EH
000074 B4000A            CJNE    A,#00H,COMBACK
000077 120130            LCALL   INITLCD
00007A 052F              INC     COUNT1
00007C 052E              INC     COUNT2
00007E 1202BF            LCALL   INCADD
000081 1200F1  COMBACK:  LCALL   SCAN
000084 120256            LCALL   CHECK
000087 BF0008            CJNE    R7,#00H,COMM
00008A E52F              MOV     A,COUNT1
00008C B52EF2            CJNE    A,COUNT2,COMBACK
00008F 02006A            LJMP    NEXTL
000092 B80A03  COMM:     CJNE    R0,#0AH,COM1
000095 020081            LJMP    COMBACK
000098 B80C13  COMM1:    CJNE    R0,#0CH,COM2
00009B E52E              MOV     A,COUNT2
00009D B40003            CJNE    A,#00H,CNTI1
0000A0 020081            LJMP    COMBACK
0000A3 120130  CNTI1:    LCALL   INITLCD
0000A6 1202EF            LCALL   UP
0000A9 152E              DEC     COUNT2
0000AB 020081            LJMP    COMBACK
0000AE B80D23  COM2:     CJNE    R0,#0DH,COM3
0000B1 C3                CLR     C
0000B2 E52F              MOV     A,COUNT1
0000B4 952E              SUBB    A,COUNT2
0000B6 40C9              JC      COMBACK
```

```
0000B8 C3
0000B9 B4010B
0000BC 1202BF
0000BF 120130
0000C2 052E
0000C4 020081
0000C7 40B8
0000C9 120130
0000CC 120305
0000CF 052E
0000D1 020081

0000D4 B80B0E
0000D7 E52F
0000D9 B52E03
0000DC 020081
0000DF 12030C
0000E2 020081
0000E5 B80E99
0000E8 1204C3
0000EB 12049C
0000EE 020081


0000F1 7FEE
0000F3 8F90
0000F5 D2B4
0000F7 C2B4
0000F9 7590FF
0000FC E590
0000FE C4
0000FF FA
```

```
                CLR     C
                CJNE    A,#01H,CNTI2
                LCALL   INCADD
                LCALL   INITLCD
                INC     COUNT2
                LJMP    COMBACK
CNTI2:          JC      COMBACK
                LCALL   INITLCD
                LCALL   DOWN
                INC     COUNT2
                LJMP    COMBACK
                ;
COM3:           CJNE    R0,#0BH,COM4
                MOV     A,COUNT1
                CJNE    A,COUNT2,CR
                LJMP    COMBACK
CR:             LCALL   CLEAR
                LJMP    COMBACK
COM4:           CJNE    R0,#0EH,COMBACK
                LCALL   TOTA
                LCALL   SEND
                LJMP    COMBACK
                ;
                ;
SCAN:           MOV     R7,#0EEH
NEWR:           MOV     P1,R7
                SETB    P3.4
                CLR     P3.4
                MOV     P1,#0FFH
                MOV     A,P1
                SWAP    A
                MOV     R,A
```

```
000100 B4FF06                CJNE    A,#0FFH,PRESS
000103 EF                    MOV     A,R7
000104 23                    RL      A
000105 FF                    MOV     R7,A
000106 0200F3                LJMP    NEWR
000109 120180        PRESS:  LCALL   DELAY
00010C 7590FF                MOV     P1,#0FFH
00010F E590                  MOV     A,P1
000111 B4FFF5                CJNE    A,#0FFH,PRESS
000114 EF            DECODE: MOV     A,R7
000115 7B04                  MOV     R3,#04H
000117 7C00                  MOV     R4,#00H
000119 D3            DEROW:  SETB    C
00011A 13                    RRC     A
00011B 5006                  JNC     DECOL
00011D 0C                    INC     R4
00011E 0C                    INC     R4
00011F 0C                    INC     R4
000120 0C                    INC     R4
000121 DBF5                  DJNZ    R3,DEROW
000123 EA            DECOL:  MOV     A,R2
000124 7B04                  MOV     R3,#04H
000126 D3            DECOL1: SETB    C
000127 13                    RRC     A
000128 5003                  JNC     FINI
00012A 0C                    INC     R4
00012B DBF9                  DJNZ    R3,DECOL1
00012D A804          FINI:   MOV     R0,04H
00012F 22                    RET
                             ;
000130 7480          INITLCD: MOV    A,#80H
000132 908003                MOV     DPTR,#PCON
000135 F0                    MOVX    @DPTR,A
```

```
000136 7400          MOV     A,#00H
000138 908001        MOV     DPTR,#PSIG
00013B F0            MOVX    @DPTR,A
00013C 7438          MOV     A,#38H
00013E 908000        MOV     DPTR,#PDAT
000141 F0            MOVX    @DPTR,A
000142 7400          MOV     A,#00H
000144 12016F        LCALL   EPLUSE
000147 120180        LCALL   DELAY
                     ;
00014A 740F          MOV     A,#00001111B
00014C 908000        MOV     DPTR,#PDAT
00014F F0            MOVX    @DPTR,A
000150 7400          MOV     A,#00H
000152 12016F        LCALL   EPLUSE
                     ;
000155 7406          MOV     A,#00000110B
000157 908000        MOV     DPTR,#PDAT
00015A F0            MOVX    @DPTR,A
00015B 7400          MOV     A,#00H
00015D 12016F        LCALL   EPLUSE
                     ;
000160 7401          MOV     A,#00000001B
000162 908000        MOV     DPTR,#PDAT
000165 F0            MOVX    @DPTR,A
000166 7400          MOV     A,#00H
000168 12016F        LCALL   EPLUSE
00016B 120180        LCALL   DELAY
00016E 22            RET
                     ;
```

```
00016F D2E2     EPLUSE:  SETB    ACC.2
000171 908001            MOV     DPTR,#PSIG
000174 F0                MOVX    @DPTR,A
000175 7E00              MOV     R6,#00H
000177 DEFE     EP1:     DJNZ    R6, EP1
000179 C2E2              CLR     ACC.2
00017B 908001            MOV     DPTR,#PSIG
00017E F0                MOVX    @DPTR,A
00017F 22                RET
                         ;
000180 750964   DELAY:   MOV     09H,#100D
000183 75080A   DEL1:    MOV     08H,#10D
000186 00       DEL2:    NOP
000187 00                NOP
000188 00                NOP
000189 D508FA            DJNZ    08H,DEL2
00018C D509F4            DJNZ    09H,DEL1
00018F 22                RET
                         ;
000190 7402     DISP:    MOV     A,#02H
000192 7C02              MOV     R4,#02D
000194 7B06              MOV     R3,#06D
000196 C083     NEXT:    PUSH    DPH
000198 C082              PUSH    DPL
00019A 1201BC            LCALL   GOTO
00019D D082              POP     DPL
00019F D083              POP     DPH
0001A1 7400     REOUT:   MOV     A,#00H
0001A3 93                MOVC    A,@A+DPTR
0001A4 C083              PUSH    DPH
0001A6 C082              PUSH    DPL
0001A8 1201CC            LCALL   WRBYTE
0001AB D082              POP     DPL
```

```
0001AD D083                    POP     DPH
0001AF A3                      INC     DPTR
0001B0 DBEF                    DJNZ    R3,REOUT
0001B2 7440                    MOV     A,#40H
0001B4 DC01                    DJNZ    R4,NEXTL1
0001B6 22                      RET
0001B7 7B06            NEXTL1: MOV     R3,#06H
0001B9 020196                  LJMP    NEXT
                               ;
0001BC D2E7            GOTO:   SETB    ACC.7
0001BE 908000                  MOV     DPTR,#PDAT
0001C1 F0                      MOVX    @DPTR,A
0001C2 7400                    MOV     A,#00H
0001C4 908001                  MOV     DPTR,#PSIG
0001C7 F0                      MOVX    @DPTR,A
0001C8 12016F                  LCALL   EPLUSE
0001CB 22                      RET
0001CC 908000          WRBYTE: MOV     DPTR,#PDAT
0001CF F0                      MOVX    @DPTR,A
0001D0 7401                    MOV     A,#01H
0001D2 908001                  MOV     DPTR,#PSIG
0001D5 F0                      MOVX    @DPTR,A
0001D6 12016F                  LCALL   EPLUSE
0001D9 22                      RET
                               ;
0001DA 1200F1          ENTDAT: LCALL   SCAN
0001DD 120256          ENTDAT1:LCALL   CHECK
0001E0 757E00                  MOV     7EH,#00H
0001E3 BF00F4                  CJNE    R7,#00H,ENTDAT
0001E6 8811                    MOV     11H,R0
0001E8 120261                  LCALL   DISPC
```

```
0001EB 1200F1        WORNG:  LCALL   SCAN
0001EE 8810                  MOV     10H,R0
0001F0 B81A06                CJNE    R0,#0AH,DEL
0001F3 85112D                MOV     2DH,11H
0001F6 02024F                LJMP    ENTER
0001F9 B80B3B        DEL:    CJNE    R0,#0BH,CHAR
0001FC 120130        ADEL:   LCALL   INITLCD
0001FF E517                  MOV     A,17H
000201 2518                  ADD     A,18H
000203 B40009                CJNE    A,#00H,NOTA
000206 9004F4                MOV     DPTR,#TAB
000209 120190                LCALL   DISP
00020C 0201DA                LJMP    ENTDAT
00020F 851883        NOTA:   MOV     DPH,18H
000212 851782                MOV     DPL,17H
000215 E0                    MOVX    A,@DPTR
000216 F515                  MOV     15H,A
000218 54F0                  ANL     A,#0F0H
00021A C4                    SWAP    A
00021B F8                    MOV     R0,A
00021C 120261                LCALL   DISPC
00021F E515                  MOV     A,15H
000221 540F                  ANL     A,#0FH
000223 F8                    MOV     R0,A
000224 120261                LCALL   DISPS
000227 1202CB                LCALL   DISPS
00022A 1200F1                LCALL   SCAN
00022D B80BAD                CJNE    R0,#0BH,ENTDATI
000230 757EFF                MOV     7EH,#0FFH
000233 120130                LCALL   INITLCD
000236 22                    RET
```

```
000237 120256      CHAR:   LCALL   CHECK
00023A BF00AE              CJNE    R7,#00H,WRONG
00023D 120261              LCALL   DISPC
000240 E511                MOV     A,11H
000242 C4                  SWAP    A
000243 2510                ADD     A,10H
000245 F52D                MOV     2DH,A
000247 1200F1      LOAD:   LCALL   SCAN
00024A B80A03              CJNE    R0,#0AH,CD
00024D E52D                MOV     A,2DH
00024F 22          ENTER:  RET
000250 B80BF4      CD:     CJNE    R0,#0BH,LOAD
000253 0201FC              LJMP    ADEL
                           ;
000256 7FFF        CHECK:  MOV     R7,#0FFH
000258 C3                  CLR     C
000259 E8                  MOV     A,R0
00025A 940A                SUBB    A,#0AH
00025C 5002                JNC     CO
00025E 7F00                MOV     R7,#00H
000260 22          CO:     RET
                           ;
000261 E8          DISPC:  MOV     A,R0
000262 2430                ADD     A,#30H
000264 1201CC              LCALL   WRBYTE
000267 22                  RET
                           ;
000268 E8          ONELINE: MOV    A,R0
000269 C4                  SWAP    A
00026A F514                MOV     14H,A
00026C 120261              LCALL   DISPC
00026F 757F00              MOV     7FH,#00H
                           ;
```

```
000272 1200F1      BEG1:   LCALL   SCAN
000275 120256              LCALL   CHECK
000278 BF001A              CJNE    R7,#00H,CHDEL
00027B E514                MOV     A,14H
00027D 28                  ADD     A,R0
00027E F514                MOV     14H,A
000280 851415              MOV     15H,14H
000283 120261              LCALL   DISPC
                           ;
000286 851883              MOV     DPH,18H
000289 851782              MOV     DPL,17H
00028C E514                MOV     A,14H
00028E F0                  MOVX    @DPTR,A
00028F 1202CB              LCALL   DISPC
000292 02029F              LJMP    NEX
                           ;
000295 B80BDA      CHDEL:  CJNE    R0,#0BH,BEG1
000298 120130              LCALL   INITLCD
00029B 757FFF              MOV     7FH,#0FFH
00029E 22                  RET
                           ;
00029F 1200F1      NEX:    LCALL   SCAN
0002A2 B80B07              CJNE    R0,#0BH,NODEL
0002A5 120130              LCALL   INITLCD
0002A8 757FFF              MOV     7FH,#0FFH
0002AB 22                  RET
0002AC 1201DD      NODEL:  LCALL   ENTDAT1
0002AF E57E                MOV     A,7EH
0002B1 B4000A              CJNE    A,#00H,BACK
0002B4 E52D                MOV     A,2DH
0002B6 851883              MOV     DPH,18H
0002B9 851782              MOV     DPL,17H
0002BC A3                  INC     DPTR
```

```
0002BD F0                       MOVX    @DPTR,A
0002BE 22           BACK:       RET
                                ;
0002BF C3           INCADD:     CLR     C
0002C0 0517                     INC     17H
0002C2 0517                     INC     17H
0002C4 E518                     MOV     A,18H
0002C6 3400                     ADDC    A,#00H
0002C8 F518                     MOV     18H,A
0002CA 22                       RET
                                ;
0002CB E515         DISPS:      MOV     A,15H
0002CD 75F00C                   MOV     B,#0CH
0002D0 C3                       CLR     C
0002D1 940A                     SUBB    A,#0AH
0002D3 5005                     JNC     CONV
0002D5 E515                     MOV     A,15H
0002D7 0202DF                   LJMP    MULTI
0002DA E515         CONV:       MOV     A,15H
0002DC C3                       CLR     C
0002DD 9406                     SUBB    A,#06H
0002DF A4           MULTI:      MUL     AB
0002E0 C3                       CLR     C
0002E1 2400                     ADD     A,#00H
0002E3 F582                     MOV     DPL,A
0002E5 E5F0                     MOV     A,B
0002E7 2415                     ADD     A,#15H
0002E9 F583                     MOV     DPH,A
0002EB 120190                   LCALL   DISP
0002EE 22                       RET
                                ;
```

```
0002EF 12041C    UP:     LCALL   DECADD
0002F2 E52E              MOV     A,COUNT2
0002F4 B4010A            CJNE    A,#01H,UPE
0002F7 9004F4            MOV     DPTR,#TAB
0002FA 120190            LCALL   DIPS
0002FD 120428            LCALL   SHOW1
000300 22                RET
000301 12043E    UPE:    LCALL   UPDWE
000304 22                RET
                         ;
000305 1202BF    DOWN:   LCALL   INCADD
000308 12043E            LCALL   UPDWE
00030B 22                RET
                         ;
00030C 120130    CLEAR:  LCALL   INITLCD
00030F E518              MOV     A,18H
000311 2517              ADD     A,17H
000313 B40009            CJNE    A,#00H,NONTA
000316 9004F4            MOV     DPTR,#TAB
000319 120190            LCALL   DISP
00031C 020328            LJMP    CHDAT
00031F 120428    NONTA:  LCALL   SHOW1
000322 851915            MOV     15H,19H
000325 1202CB            LCALL   DISPS
000328 1200F1    CHDAT:  LCALL   SCAN
00032B 120256            LCALL   CHECK
00032E BF003C            CJNE    R7,#00H,CON1
000331 8811              MOV     11H,R0
000333 120261            LCALL   DISPC
000336 1200F1    WRONG1: LCALL   SCAN
000339 8810              MOV     10H,R0
00033B B80A05            CJNE    R0,#0AH,CHAR1
00033E E511              MOV     A,11H
```

```
000340 020351            LJMP    ENTER1
000343 120256  CHAR1:     LCALL   CHECK
000346 BF00ED            CJNE    R7,#00H,WRONG1
000349 120261            LCALL   DISPC
00034C E511              MOV     A,11H
00034E C4                SWAP    A
00034F 2510              ADD     A,10H
000351 A92E    ENTER:    MOV     R1,COUNT2
000353 B90009            CJNE    R1,#00H,INCA
000356 851883            MOV     DPH,18H
000359 851782            MOV     DPL,17H
00035C 020366            LJMP    OVE
00035F 851883  INCA:     MOV     DPH,18H
000362 851782            MOV     DPL,17H
000365 A3                INC     DPTR
000366 F0      OVE:      MOVX    @DPTR,A
000367 7420              MOV     A,#20H
000369 1201CC            LCALL   WRBYTE
00036C 22                RET
00036D B80B3D  CON1:     CJNE    R0,#0BH,OUT1
000370 E517              MOV     A,17H
000372 2518              ADD     A,18H
000374 B40003            CJNE    A,#00H,NET
000377 020328            LJMP    CHDAT
00037A 85171C  NET:      MOV     1CH,17H
00037D 85181D            MOV     1DH,18H
000380 752302            MOV     23H,#02H
000383 120130  REFLA:    LCALL   INITLCD
000386 120477            LCALL   OFF
000389 120467            LCALL   LDELAY
00038C 120130            LCALL   INITLCD
00038F 12043E            LCALL   UPDATE
000392 120467            LCALL   LDELAY
```

```
000395 D523EB            DJNZ    23H,RFFLA
000398 120488            LCALL   SHODEL
00039B 120467            LCALL   LDELAY
00039E 120467            LCALL   LDELAY
0003A1 120467            LCALL   LDELAY
0003A4 851C17            MOV     17H,1CH
0003A7 851D18            MOV     18H,1DH
0003AA 1203AE            LCALL   CLRL
0003AD 22        OUT1:   RET
                         ;
0003AE E52F      CLRL:   MOV     A,COUNT1
0003B0 C3                CLR     C
0003B1 952E              SUBB    A,COUNT2
0003B3 14                DEC     A
0003B4 F51B              MOV     1BH,A
0003B6 F51E              MOV     1EH,A
0003B8 85181D            MOV     1DH,18H
0003BB 85171C            MOV     1CH,17H
0003BE B40003            CJNE    A,#00H,NLDAT
0003C1 0203EE            LJMP    LDAT
0003C4 1203F4    NLDAT:  LCALL   DUMP
                         ;
0003C7 C3                CLR     C
0003C8 851C17            MOV     17H,1CH
0003CB 0517              INC     17H
0003CD E51D              MOV     A,1DH
0003CF 3400              ADDC    A,#00H
0003D1 F518              MOV     18H,A
0003D3 851E1B            MOV     1BH,1EH
0003D6 1203F4            LCALL   DUMP
0003D9 152F              DEC     COUNT1
0003DB 851C17            MOV     17H,1CH
0003DE 851D18            MOV     18H,1DH
```

```
0003E1 120130                LCALL   INITLCD
0003E4 12043E                LCALL   UPDWE
0003E7 851C17                MOV     17H,1CH
0003EA 851D18                MOV     18H,1DH
0003ED 22                    RET
0003EE 120130        LDAT:   LCALL   INITLCD
0003F1 152F                  DEC     COUNT1
0003F3 22                    RET
                             ;
0003F4 C3            DUMP:   CLR     C
0003F5 AE17                  MOV     R6,17H
0003F7 0E                    INC     R6
0003F8 0E                    INC     R6
0003F9 E518                  MOV     A,18H
0003FB 3400                  ADDC    A,#00H
0003FD FF                    MOV     R7,A
0003FE 8F83                  MOV     DPH,R7
000400 8E82                  MOV     DPL,R6
000402 E0                    MOVX    A,@DPTR
000403 C083                  PUSH    DPH
000405 C082                  PUSH    DPL
000407 851883                MOV     DPH,18H
00040A 851782                MOV     DPL,17H
00040D F0                    MOVX    @DPTR,A
00040E D082                  POP     DPL
000410 D083                  POP     DPH
000412 858217                MOV     17H,DPL
000415 858318                MOV     18H,DPH
000418 D51BD9                DJNZ    1BH,DUMP
00041B 22                    RET
                             ;
00041C C3            DECADD: CLR     C
00041D 1517                  DEC     17H
```

```
00045D 540F      SWA:    ANL     A,#0FH
00045F F51A              MOV     1AH,A
000461 E519              MOV     A,19H
000463 54F0              ANL     A,#0F0H
000465 C4                SWAP    A
000466 22                RET
                         ;
000467 7521FF    LDELAY: MOV     21H,#0FFH
00046A 7522EA    LDE1:   MOV     22H,#0EAH
00046D 00        LDE2:   NOP
00046E 00                NOP
00046F 00                NOP
000470 D522FA            DJNZ    22H,LDE2
000473 D521F4            DJNZ    21H,LDE1
000476 22                RET
000477 900500    OFF:    MOV     DPTR,#TAB1
00047A 120190            LCALL   DISP
00047D 7420              MOV     A,#20H
00047F 1201CC            LCALL   WRBYTE
000482 7420              MOV     A,#20H
000484 1201CC            LCALL   WRBYTE
000487 22                RET
                         ;
000488 120130    SHODEL: LCALL   INITLCD
00048B 90050C            MOV     DPTR,#TAB2
00048E 120190            LCALL   DISP
000491 7420              MOV     A,#20H
000493 1201CC            LCALL   WRBYTE
000496 7420              MOV     A,#20H
000498 1201CC            LCALL   WRBYTE
00049B 22                RET
                         ;
```

```
00049C E52F      SEND:   MOV     A,COUNT1
00049E 252F              ADD     A,COUNT1
0004A0 FB                MOV     R3,A
0004A1 758300            MOV     DPH,#00H
0004A4 F582              MOV     DPL,A
0004A6 74FF              MOV     A,#0FFH
0004A8 F0                MOVX    @DPTR,A
0004A9 900000            MOV     DPTR,#0000H
0004AC 53877F            ANL     PCON.7,#7FH
0004AF 758DFA            MOV     TH1,#0FAH
0004B2 759850            MOV     SCON,#50H
0004B5 758920            MOV     TMOD,#20H
0004B8 758840            MOV     TCON,#40H
0004BB 75A890            MOV     IE,#90H
0004BE D299              SETB    SCON.1
0004C0 0204C0    LLOOP1: LJMP    LLOOP1
                         ;
0004C3 120130    TOTA:   LCALL   INITLCD
0004C6 900518            MOV     DPTR,#TAB3
0004C9 120190            LCALL   DISP
0004CC AA2F              MOV     R2,COUNT1
0004CE 1A                DEC     R2
0004CF EA                MOV     A,R2
0004D0 C3                CLR     C
0004D1 940A              SUBB    A,#0AH
0004D3 500B              JNC     A6
0004D5 8A00              MOV     R0,R2
0004D7 120261            LCALL   DISPC
0004DA 7420              MOV     A,#20H
0004DC 1201CC            LCALL   WRBYTE
0004DF 22                RET
```

```
0004E0 C3          A6:     CLR     C
0004E1 EA                  MOV     A,R2
0004E2 2406                ADD     A,#06H
0004E4 FA                  MOV     R2,A
0004E5 C4                  SWAP    A
0004E6 540F                ANL     A,#0FH
0004E8 F8                  MOV     R0,A
0004E9 120261              LCALL   DISPC
0004EC EA                  MOV     A,R2
0004ED 540F                ANL     A,#0FH
0004EF F8                  MOV     R0,A
0004F0 120261              LCALL   DISPC
0004F3 22                  RET
                           ;
                           ;
0004F4 20205441424C45  TAB:    DFB "    TABLE    :"
                           ;
000500 20202020202020  TAB1:   DFB "             "
                           ;
00050C 202044454C4554  TAB2:   DFB "    DELETE   :"
                           ;
000518 544F54414C204F  TAB3:   DFB "TOTAL ORDER   :"
                           ;
001500                     ORG     1500H
001500 2050495A5A4120          DFB " PIZZA      :"
00150C 2053414C414420          DFB " SALAD      :"
001518 20535445414B20          DFB " STEAK      :"
001524 20484F54444F47          DFB " HOTDOG     :"
001530 2048414D424552          DFB " HAMBERGER  :"
00153C 434849434B454E          DFB " CHICKEN PIE:"
001548 204150504C4520          DFB " APPLE PIE  :"
001554 204F4E494F4E20          DFB " ONION SOUP :"
001560 20444F4E555420          DFB " DONUT      :"
```

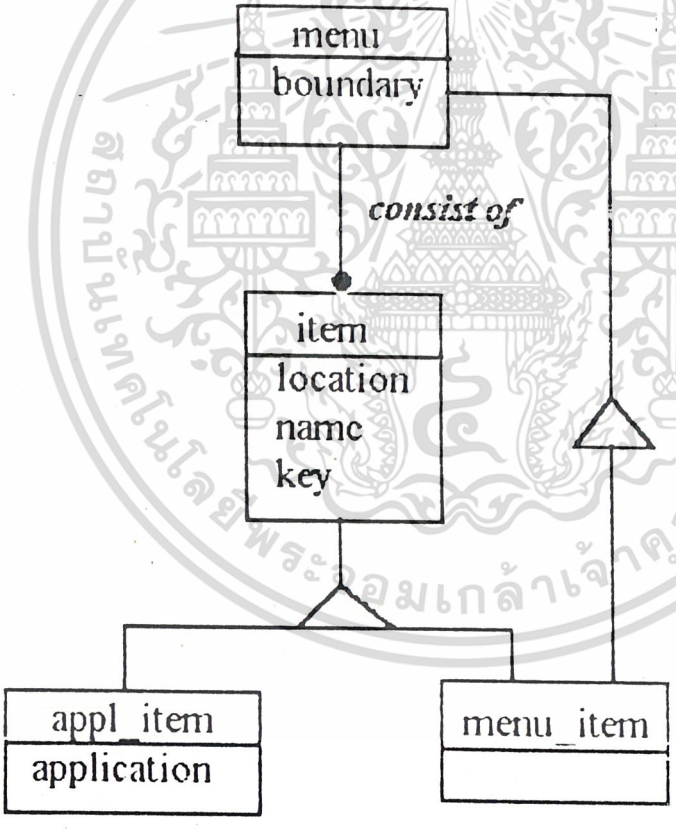
```
00156C 20484F5420434F
001578 2049434520434F
001584 20494345205445
001590 20434F43412043
00159C 2050554E434820
0015A8 204A5549434520
0015B4 204D494C4B2020
0015C0 20424545522020
000000
```

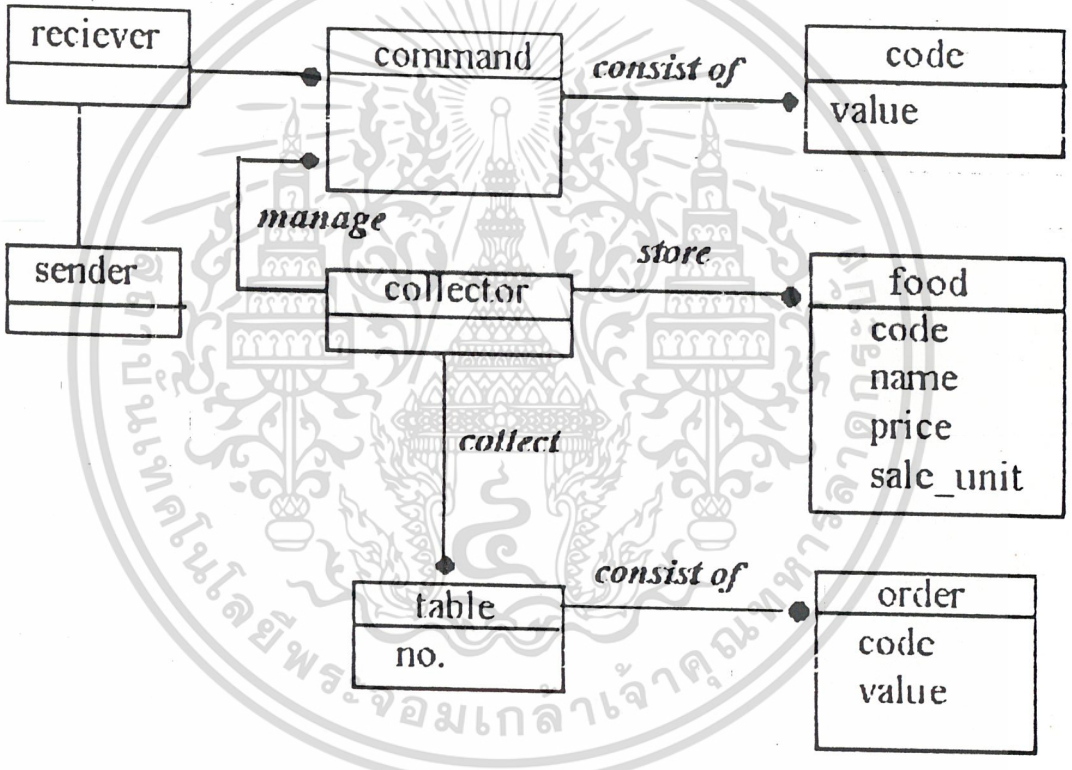
```
DFB " HOT COFFEE    :"
DFB " ICE COFFEE    :"
DFB " ICE TEA       :"
DFB " COCA COLA     :"
DFB " PUNCH         :"
DFB " JUICE         :"
DFB " MILK          :"
DFB " BEER          :"
END
```

000000E0	A	000004E0	AG	000000E2	ACC.
000000E7	ACC.7	000001FC	ADEL	000000F0	B
000002BE	BACK	00000272	BEG1	00000250	CD
00000237	CHAR	00000343	CHAR1	00000328	CHDA
00000295	CHDEL	00000256	CHECK	00000064	CHUP
0000030C	CLEAR	000003AE	CLRL	000000A3	CNT1
000000C7	CNT12	00000260	CO	00000098	COM1
000000AE	COM2	000000D4	COM3	000000E5	COM4
00000081	COMBACK	00000092	COMM	0000036D	CON1
000002DA	CONV	0000002F	COUNT1	0000002E	COUN
000000DF	CR	0000041C	DECADD	00000114	DECO
00000123	DECOL	00000126	DECOL1	000001F9	DEL
00000183	DEL1	00000186	DEL2	00000180	DELA
00000119	DEROW	00000190	DISP	00000261	DISP
000002CB	DISPS	00000305	DOWN	00000083	DPH
00000082	DPL	000003F4	DUMP	000001DA	ENTD
000001DD	ENTDAT1	0000024F	ENTER	00000351	ENTE
00000177	EP1	0000016F	EPLUSE	0000002D	EXIT
0000012D	FINI	000001BC	GOTO	000000A8	IE
0000035F	INCA	000002BF	INCADD	00000130	INIT
000003EE	LDAT	0000046A	LDE1	0000046D	LDE2
00000467	LDELAY	000004C0	LLOOP1	00000247	LOAD
0000002E	MAIN	000002DF	MULTI	0000037A	NET
000000F3	NEWR	0000029F	NEX	00000196	NEXT

0000006A	NEXTL	000001B7	NEXTL1	000003C4	NLDA
000002AC	NODEL	0000031F	NONTA	0000020F	NOTA
00000477	OFF	00000268	ONELINE	00000058	OR1
000003AD	OUT1	00000366	OVE	00000090	P1
000000B4	P3.4	00008003	PCON	00000087	PCON
00008000	PDAT	00000109	PRESS	00008001	PSIG
00000000	R0	00000006	R6	00000007	R7
00000383	REFLA	000001A1	REOUT	00000099	SBUF
000000F1	SCAN	00000098	SCON	00000099	SCON
0000049C	SEND	00000488	SHODEL	00000428	SHOW
00000081	SP	0000045D	SWA	000004F4	TAB
00000500	TAB1	0000050C	TAB2	00000518	TAB3
00000088	TCON	0000008D	TH1	0000008B	TL1
00000089	TMOD	000004C3	TOTA	00000029	TRAN
000002EF	UP	0000043E	UPDWE	00000301	UPE
000001CC	WRBYTE	000001EB	WRONG	00000336	WRON

โปรแกรมภาครับ





```
#ifndef _ITEM_H
#define _ITEM_H

/*------------------------------Class Item------------------------------*/

class item{
  int x, y;     //location
  char* name;
  int key;

  void _draw(int attr);

public:
  item(int xx, int yy, char* nn, int ky);
  ~item() { delete name; }

  void write() { _draw(0x70); }
  void light() { _draw(0x20); }
  int is_this_key(int ky) { return key == ky; }

  virtual void active() = 0;
};

/*-------------------------Class Application Item-----------------------*/

typedef void (*func)();

class appl_item : public item{
  func application;

public:
  appl_item(int x, int y, char* name, int key, func appl)
    : item(x, y, name, key), application(appl) { }

  void active() { (*application)(); }
};

/*-------------------------------Class Menu-----------------------------*/

#include "plate.h"
#include<conio.h>

class menu{
  clist<item*> group;
  char* buffer;

protected:
  int x1, y1;           //boundary
  int x2, y2;

public:
  menu(int xx1, int yy1, int xx2, int yy2, item** grp, int sz )
    : x1(xx1), y1(yy1), x2(xx2), y2(yy2), group(grp, sz)
  {
    buffer = new char[2*(x2-x1+1)*(y2-y1+1)];
  }

  ~menu() { delete buffer; }

  void display_all();
  void hide_all() { puttext(x1, y1, x2, y2, buffer); }
  void active_cur() { group.look()->active(); }
  void deactive_cur();

  void light_next();
```

```
  void light_prev();
  void light_loc(int i);

  int ismatch(int ch);
  int is_cur_active();
};

inline void menu::light_next()
{
  group.look()->write();
  group.next();
  group.look()->light();
}

inline void menu::light_prev()
{
  group.look()->write();
  group.prev();
  group.look()->light();
}

inline void menu::light_loc(int i)
{
  group.look()->write();
  group.set(i);
  group.look()->light();
}

/*-------------------------------Class Menu Item---------------------------*/

#include "tool.h"

class menu_item : public menu, public item{

  public:
    int isactive;
    menu_item(int x1, int y1, int x2, int y2, item** group, int size,
              int xx1, int yy1, char* name, int key)
     : menu(x1, y1, x2, y2, group, size), item(xx1, yy1, name, key),
       isactive(0) { }

   void display_all()
     { menu::display_all(); boundary(x1+1, y1, x2-1, y2, 0x70); }


   void active() { display_all(); isactive = 1; }
   void deactive() { menu::hide_all(); isactive = 0; }
};

inline int menu::is_cur_active()
  { return ((menu_item*)(group.look()))->isactive != 0; }

inline void menu::deactive_cur()
  { ((menu_item*)(group.look()))->deactive(); }

/*-------------------------------------------------------------------------*/

extern void main_menu(menu& head, int ch);      //state diagram of main menu
extern int sub_menu(menu_item& sub, int ch);    //state diagram of sub menu

#endif
```

```
#include<string.h>
#include<conio.h>
#include "item.h"
#include "tool.h"

item::item(int xx, int yy, char* nn, int ky)
  : x(xx), y(yy), key(ky)
{
  name = new char[strlen(nn)+1];
  strcpy(name, nn);
}

void item::_draw(int attr)
{
  textattr(attr);
  gotoxy(x, y);
  char* p = name;
  while( *p ){
    if( *p != '~' ) putch(*p++);
    else {
      p++;
      textattr( (attr&0xf0) | 0x04 );
      putch(*p++);
      textattr(attr);
    }
  }
}

void menu::display_all()
{
  gettext(x1, y1, x2, y2, buffer);
  textattr(0x75);
  window(x1, y1, x2, y2);
  clrscr();
  window(1, 1, 80, 25);
  group.reset();
  do{
    group.look()->write();
    group.next();
  } while( !group.isend() );
  group.look()->light();
}

int menu::ismatch(int ch)      //return location if match , -1 not found
{
  clist<item*> list = group;
  list.reset();
  do{
    if( list.look()->is_this_key(ch) ) return list.getcur();
    list.next();
  } while( !list.isend() );
  return -1;
}

#include<dos.h>

#define ENTER 13
#define BS 8
#define LEFTARROW 256+75
#define RIGHTARROW 256+77
#define UPARROW 256+72
#define DOWNARROW 256+80
#define ESC 27

void main_menu(menu& head, int ch)
{
```

```
  int j;
  if( ch == ENTER || ch == DOWNARROW ) head.active_cur();
  else if( ch == LEFTARROW ){
    if( head.is_cur_active() ){
      head.deactive_cur();
      head.light_prev();
      head.active_cur();
    }
    else head.light_prev();
  }
  else if( ch == RIGHTARROW ){
    if( head.is_cur_active() ){
      head.deactive_cur();
      head.light_next();
      head.active_cur();
    } else head.light_next();
  }
  else if( (j=head.ismatch(ch)) != -1 ){
    if( head.is_cur_active() ) head.deactive_cur();
    head.light_loc(j);
    head.active_cur();
  }
}

int sub_menu(menu_item& m, int ch)
{
  int isuse = 0;
  int j;
  if( m.isactive ){
    isuse = 1;
    if( ch == DOWNARROW ) m.light_next();
    else if( ch == UPARROW ) m.light_prev();
    else if( ch == ENTER ){
      m.deactive();
      m.active_cur();
    }
    else if( ch == ESC ) m.deactive();
    else if( (j = m.ismatch(ch)) != -1 ){
      m.light_loc(j);
      delay(100);
      m.deactive();
      m.active_cur();
    }
    else isuse=0;
  }
  return isuse;
}
```

```
#ifndef _INOUT_H
#define _INOUT_H

/*-----------------------------Class Output-------------------------------*/
#include<conio.h>

class output{
  int x1, y1;           //boundary
  int x2, y2;
  int attribute;        //screen attribute
  int x, y;             //current cursor

public:
  output(int x1, int y1, int x2, int y2, int attr);
  void set();
  void store();
};

inline void output::set()
{
  textattr(attribute);
  window(x1+1, y1+1, x2-1, y2-1);
  gotoxy(x, y);
}

inline void output::store()
{
  x = wherex(); y = wherey();
  window(1, 1, 80, 25);
}


/*-----------------------------Class Input--------------------------------*/

class input{
  int x, y;     //location
  char keybuf[20];
  char buffer[2*30*5];

public:
  int current;

  input(int xx, int yy)
    : x(xx), y(yy) { }

  void display(char* command, char* comment);
  void collect(char ch);
  void erase();
  char* terminate();
  char* quit();

};

inline void input::collect(char ch)
{
  keybuf[current++] = ch;
  putch(ch);
}

inline void input::erase()
{
  current--;
  int xx = wherex();
  gotoxy(xx-1, 1);              //window mode x+5, y+3, x+24, y+3
  putch(' ');
  gotoxy(xx-1, 1);
```

```
}

inline char* input::terminate()
{
  puttext(x, y, x+29, y+4, buffer);
  keybuf[current] = '\0';
  window(1, 1, 80, 25);
  if( current != 0 ) return keybuf;
  return 0;
}

inline char* input::quit()
{
  puttext(x, y, x+29, y+4, buffer);
  window(1, 1, 80, 25);
  return 0;
}


extern char* prompt(input& in, char* command, char* comment);

#endif
```

```
#include "tool.h"
#include<conio.h>
#include<string.h>
#include "inout.h"
#include "ctype.h"

output::output(int xx1, int yy1, int xx2, int yy2, int attr)
  : x1(xx1), y1(yy1), x2(xx2), y2(yy2), attribute(attr)
{
  boundary(x1, y1, x2, y2, (attribute&0xf0) | 0x0f);
  textattr(attribute);
  window(x1+1, y1+1, x2-1, y2-1);
  clrscr();
  window(1, 1, 80, 25);
  x = wherex(); y = wherey();
}

void input::display(char* command, char* comment)
{
  gettext(x, y, x+29, y+4, buffer);
  boundary(x, y, x+29, y+4, 0x7f);
  textattr(0x70);
  int space = strlen(command)/2;
  gotoxy(x+14-space, y);
  cprintf("%s", command);
  window(x+1, y+1, x+28, y+3);
  clrscr();
  gotoxy(4, 1);
  cprintf("%s", comment);
  textattr(0x17);
  window(x+4, y+2, x+24, y+2);
  clrscr();
  current = 0;
}

#define ENTER 13
#define BS 8
#define LEFTARROW 256+75
#define ESC 27

char* prompt(input& in, char* command, char* comment)
{
  _setcursortype(_NORMALCURSOR);
  in.display(command, comment);
  int ch;
  for(;;){
    ch = keyin();
    if( ch ==  BS || ch == LEFTARROW ){
      if( in.current > 0 ) in.erase();
    }
    else if( ch == ENTER ){
      _setcursortype(_NOCURSOR);
      return in.terminate();
    }
    else if( ch == ESC ){
      _setcursortype(_NOCURSOR);
      return in.quit();
    }
    else if( isascii(ch) && in.current < 20 ) in.collect(ch);
  }
}
```

```
#ifndef _PLATE_H
#define _PLATE_H

/*------------------------Class Circular list-----------------------------*/

template< class T > class clist{
  T* list;
  int size;
  int current;

public:
  clist(T* lst,int sz)
    : list(lst), size(sz), current(0) { }

  void next(){ if( ++current >= size ) current=0; }
  void prev(){ if( --current < 0 ) current = size-1; }
  T look(){ return list[current]; }
  void set(int loc) { current = loc; }
  void reset() { current = 0; }
  int isend() { return current == 0; }          //return to zero again
  int getcur() { return current; }

};

/*------------------------------Class Queue-------------------------------*/

template<class T> class list{
public:
  T inform;
  list<T>* next;

  list(T in, list<T>* nn)
    : inform(in), next(nn) { }
};

template<class T> class queue{
  list<T>* head;
  list<T>* tail;

public:
  queue()
    : head(0), tail(0) { }
  ~queue();

  int isempty() { return head==0; }
  void put(T);
  void put_head(T);
  T get();
};

template<class T>
  inline void queue<T>::put(T in)
{
  if( isempty() ) head = tail = new list<T>(in, 0);
  else tail = tail->next = new list<T>(in, 0);
}

template<class T>
  inline void queue<T>::put_head(T in)
{
  if( isempty() ) head = tail = new list<T>(in, 0);
  else head = new list<T>(in, head);
}

template<class T>
  inline T queue<T>::get()
```

```
{
  list<T>* cur = head;
  if( head == tail ) head = tail = 0;
  else head = head->next;
  T val = cur->inform;
  delete cur;
  return val;
}

template<class T>
  queue<T>::~queue()
{
  while( !isempty() ) get();
}

/*--------------------------------Class Hash--------------------------------*/

template<class T> class hash{
  T* table;
  int size;

public:
  hash(int sz);
  ~hash() { delete table; }

  int ismember(int loc) { return loc>=0 && loc<size; }
  int getsz() { return size; }
  void insert(T in, int loc) { table[loc] = in; }
  T look(int loc) { return table[loc]; }
};

template<class T>
  hash<T>::hash(int sz)
{
  table = new T[size=sz];
  for(int i=0; i<size; i++) table[i] = 0;
}

/*--------------------------------------------------------------------------*/

#endif
```

```
#include<stdio.h>
extern void boundary(int x1, int y1, int x2, int y2, int attr);
extern int keyin();
extern int isprnrdy();
extern void title();
extern void title(FILE*);
extern void display(nt, char*, float, int , float);
extern void print(FILE*, int, char*, float, int, float);
extern void display_time();
extern void print_time(FILE*);
```

```
#include<conio.h>
#include "tool.h"

#define VERTLINE 205
#define UPPERRIGHT 200
#define LOWWERRIGHT 188
#define LOWWERLEFT 187
#define UPPERLEFT 201
#define HORIZLINE 186

void boundary(int x1, int y1, int x2, int y2, int attr)
{
  textattr(attr);
  for(int i=x1; i<=x2; i++){
    gotoxy(i, y1); putch(VERTLINE);
    gotoxy(i, y2); putch(VERTLINE);
    if(i==x1 || i==x2){
      for(int j=y1; j<=y2; j++){
        gotoxy(i, j);
        putch(HORIZLINE);
      }
      if(i==x1){
       gotoxy(x1, y1); putch(UPPERLEFT);
       gotoxy(x1, y2); putch(UPPERRIGHT);
      }
      if(i==x2){
        gotoxy(x2, y1); putch(LOWWERLEFT);
        gotoxy(x2, y2); putch(LOWWERRIGHT);
      }
    }
  }
}

int keyin()
{
  int offset = 0;
  int ch = getch();
  if( ch == 0 ){
    offset = 256;
    ch = getch();
  }
  return offset+ch;
}

#include<dos.h>

int isprnrdy()
{
  REGS regs;
  regs.h.ah = 2;
  regs.x.dx = 0;
  int86(0x17, &regs, &regs);
  return ((regs.h.ah&0x20)==0 && (regs.h.ah&0x08)==0);
}

#include<conio.h>
#include<stdio.h>
#include<time.h>

void title()
{
  cprintf(""
"  CODE        NAME                PRICE/UNIT              UNIT          AMOUNT\r\n");
  cprintf(""
"  ------------------------------------------------------------------------\r\n");
}
```

```
void title(FILE* dest)
{
  fprintf(dest, ""
"  CODE         NAME              PRICE/UNIT          UNIT        AMOUNT\n");
  fprintf(dest, ""
"  ------------------------------------------------------------------\n");
}

void display(int code, char* name, float price, int order, float amount)
{
  int y = wherey();
  gotoxy(5, y); cprintf("%.2d", code);
  gotoxy(15, y); cprintf("%s", name);
  gotoxy(37, y); cprintf("%4.2f", price);
  gotoxy(54, y); cprintf("%.2d", order);
  gotoxy(64, y); cprintf("%4.2f", amount);
  cprintf("\r\n");
}

#include<string.h>

void print(FILE* out, int code, char* name,
                      float price, int order, float amount)
{
  fprintf(out,"    %.2d        %s", code, name);
  for(int i=0; i<12-strlen(name); i++) fputc(' ', out);
  fprintf(out,"              %4.2f                %.2d        %4.2f\n",
          price , order, amount );
}

void display_time()
{
  time_t longtime;
  time(&longtime);
  cprintf("      Current time : %s\r\n", ctime(&longtime) );
}

void print_time(FILE* out)
{
  time_t longtime;
  time(&longtime);
  fprintf(out, "      Current time : %s\n", ctime(&longtime) );
}
```

```
#ifndef _COLLECT_H
#define _COLLECT_H

#include "plate.h"
#define ERROR 0xcc

/*-----------------------------Class Table-------------------------------*/

typedef hash<int> table;

/*-----------------------------Class Command-----------------------------*/

typedef queue<int> command;

/*-----------------------------Class Reciever----------------------------*/

class receiver{
  queue<command*>* dest;
  command* cur_list;

public:
  int is_wait_code;

  receiver(queue<command*>* q)
    :dest(q), is_wait_code(0) { }

  void start(int code)
    { cur_list = new command; cur_list->put(code); is_wait_code = 1;}
  void store(int code) { cur_list->put(code); }
  void end() { dest->put(cur_list); is_wait_code = 0;}
  void error()
    { cur_list->put_head(ERROR); dest->put(cur_list); is_wait_code = 0; }
};

extern void interrupt port_controller(...);

/*-----------------------------Class food--------------------------------*/

#include<string.h>

struct food{
  char* name;
  float price;
  int sale_unit;

  food(char* nn, float prc, int unit=0)
    : price(prc), sale_unit(unit)
  { name = new char[strlen(nn)+1]; strcpy(name, nn); }

  ~food() { delete name; }
};

/*-----------------------------Class Collector---------------------------*/

#include<stdio.h>

class collector{
  hash<food*>* menu;
  hash<table*> list;

public:
  enum{ TONONE=0, TOSCR=1, TOPRN=2, TOFILE=4 };
  collector(hash<food*>* m, int sz)
    : menu(m), list(sz) { }
  ~collector();
```

```
  void manage(command* cmd);

  void show_menu();
  void load_stat();
  void clear_stat();
  void menu_print();

  void print_table(int no, int command, FILE*);
  void table_print();
  void show_table();
  void account(int);
  void account();

  void order_food(int no, int code, int val);
  void order();
  void cancel();
  void record(FILE*);
};

/*-------------------------------------------------------------------------*/

#endif
```

```
#include<conio.h>
#include "collect.h"
#include "inout.h"
#include "tool.h"
#include<stdio.h>
#include<string.h>
#include<stdlib.h>
#include<math.h>

#define ERROR 0xcc
#define ACCOUNT 0xbb
#define CANCEL 0xaa
#define ORDER 0x99
#define END 0xff

extern output* out;
extern output* message;
extern input* in;

collector::~collector()
{
  for(int i=0; i<list.getsz(); i++){
    if( list.look(i) != 0 ) delete list.look(i);
  }
}

void collector::manage(command* lst)
{
  int cmd = lst->get();
  if( cmd == ERROR ){
    while( !lst->isempty() ) lst->get();
    message->set();
    cprintf(" <Port Error while recieving.>");
    getch();
    clrscr();
    message->store();
    delete lst;
    return;
  }
  int tblno = lst->get();
  if( !list.ismember(tblno) ){
    while( !lst->isempty() ) lst->get();
    message->set();
    cprintf(" Table %d is not supported.", tblno );
    getch();
    clrscr();
    message->store();
    delete lst;
    return;
  }
  if( cmd == ACCOUNT ){
    account(tblno);
    delete lst;
    return;
  }
  int code, amount;
  while( !lst->isempty() ){
    code = lst->get();
    amount = lst->get();
    if( cmd == ORDER ) order_food(tblno, code, amount);
    else if( cmd == CANCEL ) order_food(tblno, code, 0-amount);
  }
  delete lst;
}

void collector::show_menu()
```

```
{
  out->set();
  clrscr();
  title();
  for(int i=0; i<menu->getsz(); i++){
    food* fd = menu->look(i);
    int amount = (fd->sale_unit)*(fd->price);
    display(i, fd->name, fd->price, fd->sale_unit, amount);
  }
  out->store();
}

void collector::load_stat()
{
  char* nn = prompt(*in, "Statistic File", "File name");
  if( strcmp(nn, "") == 0)  return;
  FILE* fin = fopen( nn, "r");
  if( fin == 0 ){
    message->set();
    cprintf(" <Can not open: %s>", nn);
    getch();
    clrscr();
    message->store();
    return;
  }
  int value;
  for(int i=0; i<menu->getsz(); i++){
    fscanf(fin, "%d", &value);
    menu->look(i)->sale_unit = value;
  }
  fclose(fin);
}

void collector::clear_stat()
{
  for(int i=0; i<menu->getsz(); i++)
    menu->look(i)->sale_unit = 0;
}

void collector::menu_print()
{
  for(;;){
    if(!isprnrdy()){
      char* ans = prompt(*in, "Printer not ready", "Retry(y/n)");
      if( strcmp(ans, "y")==0 ) continue;
      else if( strcmp(ans, "n")==0 ) break;
    }
    else{
      title(stdprn);
      int amount;
      for(int i=0; i<menu->getsz(); i++){
        food* fd = menu->look(i);
        amount = (fd->sale_unit)*(fd->price);
        print(stdprn, i, fd->name, fd->price, fd->sale_unit, amount);
      }
      break;
    }
  }
}

void collector::print_table(int no, int cmd, FILE* out)
{
  int amount;
  float total = 0;
  table* tbl = list.look(no);
  for(int i=0; i<menu->getsz(); i++){
```

```
    if( tbl->look(i) != 0 ){
      food* fd = menu->look(i);
      amount = (fd->price)*(tbl->look(i));
      if( (cmd&0x01) == TOSCR )
        display(i, fd->name, fd->price, tbl->look(i), amount);
      if( (cmd&0x02) == TOPRN )
        print(stdprn, i, fd->name, fd->price, tbl->look(i), amount);
      if( (cmd&0x04) == TOFILE )
        print(out, i, fd->name, fd->price, tbl->look(i), amount);
      total += amount;
    }
  }
  if( (cmd&0x01) == TOSCR ) cprintf(""
"
                                                    TOTAL    %4.2f \r\n"
                                    , total );

   if( (cmd&0x02) == TOPRN ) fprintf(stdprn, ""
"
                                                    TOTAL    %4.2f \n"
                                    , total );

   if( (cmd&0x04) == TOFILE) fprintf(out, ""
"
                                                    TOTAL    %4.2f \n"
                                    , total );

}

void collector::table_print()
{
  char* answer = prompt(*in, "Table to Print", "Table number");
  if( strcmp(answer, "") == 0 ) return;
  int no = atoi(answer);
  if( list.ismember(no ) ){
    if( list.look(no) != 0 ){
      for(;;){
        if( !isprnrdy() ){
          char* ans = prompt(*in, "Printer not Ready", "Retry(y/n)");
          if( strcmp(ans, "y") == 0  ) continue;
          else if( strcmp(ans, "n") == 0 ) break;
        }
        else {
          fprintf(stdprn, "                                    Table Number %d\n", no);
          title(stdprn);
          print_table(no, TOPRN, 0);
          break;
        }
      }
    }
    else{
      message->set();
      cprintf(" <No order in table number %d.>", no);
      getch();
      clrscr();
      message->store();
    }
  }
  else{
    message->set();
    cprintf(" <Range of table number is 0 to %d>", (list.getsz()-1) );
    getch();
    clrscr();
    message->store();
  }
}

void collector::show_table()
{
```

```
  char* answer = prompt(*in, "Display Table", "Table number" );
  if( strcmp(answer, "") == 0 ) return;
  int no = atoi(answer);
  if( list.ismember(no) ){
    if( list.look(no) != 0 ) {
      out->set();
      cprintf("                                    Table Number #%d \r\n", no);
      title();
      print_table(no, TOSCR, 0);
      out->store();
    }
    else{
      message->set();
      cprintf(" <Table number %d is currently not opened>", no);
      getch();
      clrscr();
      message->store();
    }
  }
  else{
    message->set();
    cprintf(" <Table number is range of 0 to %d>", (list.getsz() -1));
    getch();
    clrscr();
    message->store();
  }
}

void collector::account()
{
  char* answer = prompt(*in, "Account Table", "Table number");
  if( strcmp(answer, "") == 0 ) return;
  int no = atoi(answer);
  account(no);
}

void collector::account(int no)
{
  if( list.ismember(no) ){
    if( list.look(no) != 0 ){
      out->set();
      FILE* fout = fopen("output.dat", "a");
      cprintf("                                    Table Number #%d \r\n", no);
      fprintf(fout,"                               Table Number #%d\n", no);
      display_time(); title();
      print_time(fout); title(fout);
      for(;;){
        if( !isprnrdy() ){
          out->store();
          char* ans = prompt(*in, "Printer not Ready", "Retry(y/n)");
          out->set();
          if( strcmp(ans, "y") == 0 ) continue;
          if( strcmp(ans, "n") == 0 ){
            print_table(no, (TOSCR|TOFILE), fout);
            break;
          }
        }
        else {
          fprintf(stdprn, "                                    Table Number #%d\n", no);
          print_time(stdprn);
          title(stdprn);
          print_table(no, (TOSCR|TOPRN|TOFILE), fout);
          fputc('\x0c', stdprn);
          break;
        }
      }
```

```
      delete list.book(no);
      list.insert(0, no);
      fclose(fout);
      out->store();
      textattr(0x70);
      gotoxy(2+3*no, 25);
      cprintf("%.2d", no);
    }
    else{
      message->set();
      cprintf(" <There is no order in table>");
      getch();
      clrscr();
      message->store();
    }
  }
  else {
    message->set();
    cprintf(" <Table number range is 0 to %d>", (list.getsz()-1)) ;
    getch();
    clrscr();
    message->store();
  }
}

void collector::order_food(int no, int code, int amount)
{
  table* tbl = list.look(no);
  if( amount >= 0 ){                    //order
    if( tbl == 0 ){                     //table is not opened
      tbl = new table(menu->getsz());
      list.insert(tbl, no);
      textattr(0x7e);
      gotoxy(2+3*no, 25);
      cprintf("%.2d", no);
    }
  }
  else{                                 //cancel
    if( tbl== 0 ){                      //table is not opened
      message->set();
      cprintf(" <There is no order to cancel>");
      getch();
      clrscr();
      message->store();
      return;
    }
  }
  if( !menu->ismember(code) ){
    message->set();
    cprintf(" <No food code: %d>", code);
    getch();
    clrscr();
    message->store();
    return;
  }
  int old = tbl->look(code);
  if( old + amount >= 0 ){
    out->set();
    old += amount;
    tbl->insert( old, code);
    food* fd = menu->look(code);
    (fd->sale_unit) += amount;
    cprintf(" <Table %d: %s>\n", no, ((amount>0)? "Order": "Cancel") );
    display(code, fd->name, fd->price, abs(amount), (fd->price)*abs(amount));
    for(;;){
      if( !isprnrdy() ){
```

```
        out->store();
        char* ans = prompt(*in, "Printer not Ready", "Retry(y/n)" );
        out->set();
        if( strcmp(ans, "y") == 0 ) continue;
        if( strcmp(ans, "n") == 0 ) break;
      }
      else{
        fprintf(stdprn,"<Table %d %s>\n", no, ((amount>0)? "Order": "Cancel"));
        print(stdprn, code, fd->name, fd->price
                        , abs(amount), (fd->price)*abs(amount));
        break;
      }
    }
    out->store();
  }
  else {
    message->set();
    cprintf(" <Cancel excess: Current ordered unit is %d>", old );
    getch();
    clrscr();
    message->store();
  }
}

void collector::order()
{
  char* cno = prompt(*in, "Order Food" , "Table/Food/Amount" );
  if( strcmp(cno, "") == 0 ) return;
  char* code = strchr(cno, '/');
  if( code == 0 ) return;
  *code++ ='\0';
  char* amount = strchr(code, '/');
  if( amount == 0 ) return;
  *amount++ = '\0';
  int no = atoi(cno);
  if( !list.ismember(no) ){
    message->set();
    cprintf(" <Table range is 0 to %d>", (list.getsz()-1));
    getch();
    clrscr();
    message->store();
  }
  else{
    order_food(no, atoi(code), atoi(amount) );
  }
}

void collector::cancel()
{
  char* cno = prompt(*in, "Cancel Food" , "Table/Food/Amount" );
  if( strcmp(cno, "") == 0 ) return;
  char* code = strchr(cno, '/');
  if( code == 0 ) return;
  *code++ ='\0';
  char* amount = strchr(code, '/');
  if( amount == 0 ) return;
  *amount++ = '\0';
  int no = atoi(cno);
  if( !list.ismember(no) ){
    message->set();
    cprintf(" <Table range is 0 to %d>", (list.getsz()-1));
    getch();
    clrscr();
    message->store();
  }
  else{
```

```
    order_food(no, atoi(code), 0-atoi(amount) );
  }
}

void collector::record(FILE* fout)
{
  if( fout == 0 ){
    message->set();
    cprintf(" <Can not open such file>");
    getch();
    clrscr();
    message->store();
  }
  else{
    for(int i=0; i<menu->getsz(); i++){
      fprintf(fout, "%d\n", (menu->book(i)->sale_unit) );
    }
    fputc('\n', fout);
    print_time(fout);
  }
}


#include<dos.h>
#include<bios.h>
extern void interrupt (*origin)(...);
extern receiver port;

void interrupt port controller(...)
{
  disable();
  (*origin)();
  int code = bioscom(_COM_RECEIVE, 0, 1);
  if( (code & 0xff00) != 0 ) code = ERROR;
  else code &= 0x00ff;

  if( port.is_wait_code == 0 ){
    if( code == ORDER || code == CANCEL || code == ACCOUNT )
      port.start(code);
  }
  else{
    if( code == ERROR ) port.error();
    else if( code == END ) port.end();
    else port.store(code);
  }
  enable();
}
```

```
#include "item.h"
#include "inout.h"
#include "collect.h"
#include "plate.h"
#include "tool.h"
#include <stdlib.h>
#include <conio.h>
#include <bios.h>
#include <dos.h>
#include <string.h>

input* in;
output* out, *message;
collector* cashier;

void interrupt (*origin)(...);
queue<command*> bin;
receiver port(&bin);

inline void show_menu() { cashier->show_menu(); }
inline void load_stat() { cashier->load_stat(); }
inline void clear_stat() { cashier->clear_stat(); }
inline void menu_print() { cashier->menu_print(); }
inline void show_table() { cashier->show_table(); }
inline void account() { cashier->account(); }
inline void table_print() { cashier->table_print(); }
inline void order() { cashier->order(); }
inline void cancel() { cashier->cancel(); }

void quit()
{
  char* ans = prompt(*in, "Quit Program", "Quit(y/n)" );
  if( strcmp(ans, "n") == 0 ) return;
  else {
    FILE* fout = fopen("stat.dat", "w");
    cashier->record(fout);
    fclose(fout);
  }
  delete in;
  delete out;
  delete message;
  delete cashier;
  setvect(0x0b, origin);
  textattr(0x07);
  clrscr();
  _setcursortype( NORMALCURSOR);
  exit(1);
}

  appl_item i1(6, 3, " ~Show                ", 's', &show_menu);
  appl_item i2(6, 4, " ~Load Statistic... ", 'l', &load_stat);
  appl_item i3(6, 5, " ~Clear Statistic   ", 'c', &clear_stat);
  appl_item i4(6, 6, " ~Print               ", 'p', &menu_print);
  appl_item i5(6, 7, " ~Quit                ", 'q', &quit);

  appl_item i6(12, 3, " ~Show ...       ", 's', &show_table);
  appl_item i7(12, 4, " ~Account...     ", 'a', &account);
  appl_item i8(12, 5, " ~Print...       ", 'p', &table_print);

  appl_item i9(19, 3, " ~Order...       ", 'o', &order);
  appl_item i10(19, 4," ~Cancel..       ", 'c', &cancel);

  item* g1[] = { &i1, &i2, &i3, &i4, &i5 };
  item* g2[] = { &i6, &i7, &i8 };
  item* g3[] = { &i9, &i10 };
```

```
  menu_item i11(4, 2, 26, 8, g1, 5, 6, 1, " ~Menu ", 'm');
  menu_item i12(10,2, 28, 6, g2, 3, 12, 1," ~Table ", 't');
  menu_item i13(17,2, 35, 5, g3, 2, 19, 1, " ~Food ", 'f');

  item* g4[] = { &i11, &i12, &i13 };

  menu m(1, 1, 80, 1, g4, 3);

food f1("PIZZA", 75), f2("SALAD", 30), f3("STEAK", 70), f4("HOT DOG", 15),
     f5("HAMBERGER", 22), f6("CHICKEN PIE", 18), f7("APPLE PIE", 15),
     f8("ONION SOUP", 25), f9("DONUT", 8), f10("HOT COFFEE", 12),
     f11("ICE COFFEE", 16), f12("ICE TEA", 16), f13("COCA COLA", 13),
     f14("PUNCH", 13), f15("JUICE", 12), f16("MILK", 12), f17("BEER", 30);

hash<food*> menus(17);

int main()
{
  menus.insert( &f1, 0); menus.insert( &f2, 1); menus.insert( &f3, 2);
  menus.insert( &f4, 3); menus.insert( &f5, 4); menus.insert( &f6, 5);
  menus.insert( &f7, 6); menus.insert( &f8, 7); menus.insert( &f9, 8);
  menus.insert( &f10, 9); menus.insert( &f11, 10); menus.insert( &f12, 11);
  menus.insert( &f13, 12); menus.insert(&f14, 13); menus.insert( &f15, 14);
  menus.insert( &f16, 15); menus.insert( &f17, 16);

  origin = getvect(0x0b);
  setvect(0x0b, port_controller);
  bioscom(_COM_INIT, (_COM_CHR8 | _COM_STOP1
                      | _COM_NOPARITY | _COM_1200 ),1);
  clrscr();
  _setcursortype(_NOCURSOR);
  in = new input(30, 13);
  out = new output(1, 2, 80, 21, 0x07);
  message = new output(1, 22, 80, 24, 0x0c);

  int no;
  for(;;){
    no = atoi( prompt(*in, "Number of table", "Amount"));
    if( no > 0 && no <= 25 ) {
      cashier = new collector(&menus, no);
      break;
    }
  }
  gotoxy(1, 25);
  textattr(0x70);
  clreol();
  for(int i=0; i<no; i++){
    gotoxy(2+3*i, 25);
    cprintf("%.2d", i);
  }

  m.display_all();
  for(;;){
    if( kbhit() ){
      int ch = keyin();
      if( sub_menu(i11, ch) || sub_menu(i12, ch) || sub_menu(i13, ch) )
        continue;
      main_menu(m, ch);
    }
    else if( !bin.isempty() ){
      if( m.is_cur_active() ) m.deactive_cur();
      cashier->manage(bin.get());
    }
  }
}
```

กิตติกรรมประกาศ

ขอขอบคุณ อาจารย์กิติพล ชิดสกุล อาจารย์ที่ปรึกษาของโครงงานนี้ ที่ช่วยให้คำแนะนำและสนับสนุน ตลอดจนช่วยจัดหาเครื่องมือและอุปกรณ์ ไว้ให้พร้อมสำหรับการทำโครงงานนี้

ขอขอบคุณ บิดามารดา ที่ให้กำลังใจแก่ผู้จัดทำตลอดมาจนโครงงานนี้สำเร็จลุล่วงไปด้วยดี

ขอขอบคุณ เพื่อนๆทุกคน ที่ให้ความร่วมมือและเป็นกำลังใจให้ผู้จัดทำมาโดยตลอด