โทรศัพท์สาธารณะชนิดใช้บัตรแม่เหล็ก

MAGNETIC CARD - PUBLIC TELEPHONE

โดย

นาย สมชาย รุ่งเรือง

นาย สุรเชษฐ์ พยุงธรรม

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตร์บัณฑิต

สาขาวิศวกรรมโทรคมนาคม

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2536

โทรศัพท์สาธารณะชนิดใช้บัตรแม่เหล็ก

Magnetic Card - Public Telephone

โดย นาย สมชาย รุ่งเรือง 33100395

นาย สุรเชษฎ์ พยุงธรรม 33100449

อาจารย์ที่ปรึกษา รศ. ถวิล พึ่งมา

บทคัดย่อ

จุดประสงค์ของโครงงานนี้ เพื่อที่จะจัดทำเครื่องโทรศัพท์สาธารณะประเภทใช้การ์ด (card) โดยจะใช้ชิงเกิ้ลชิพเบอร์ 8031 เป็นตัวไมโครคอนโทรลเลอร์ และจะใช้วงจรพื้นฐานทางด้านโทรศัพท์ เช่น วงจร detect ringback, RTC ในการทำให้เราทราบถึงสัญญาณต่างๆ และสามารถตรวจสอบได้ นอกจากนี้ยังใช้สเต็ปปิ้งมอเตอร์ (stepping motor) เป็นตัวในการดึงการ์ดเพื่อที่จะนำไปสู่การตรวจสอบรหัสข้อมูล และจำนวนราคา การทำงานทั้งหมดนี้จะทำงานได้อย่างมีประสิทธิภาพเพียงใด ขึ้นอยู่กับส่วนของโปรแกรมที่ใช้ ซึ่งในที่นี้ผู้จัดทำจะใช้ภาษาแอสเซมบลี้ (assembly ASM51)

ABSTRACT

The main goal of this project is to invent a public-card telephone using single chip 8031 as a microcontroller. The fundamental telephony circuits, e.g. detect ringback and RTC , are designed to acknowledge as well as check any signal. In addition we apply stepper motor to feed card for data inspection and disply value information. Of all these operations the efficient depends on utilized language program. In this project, we use assembly language to do so.

สารบัญ

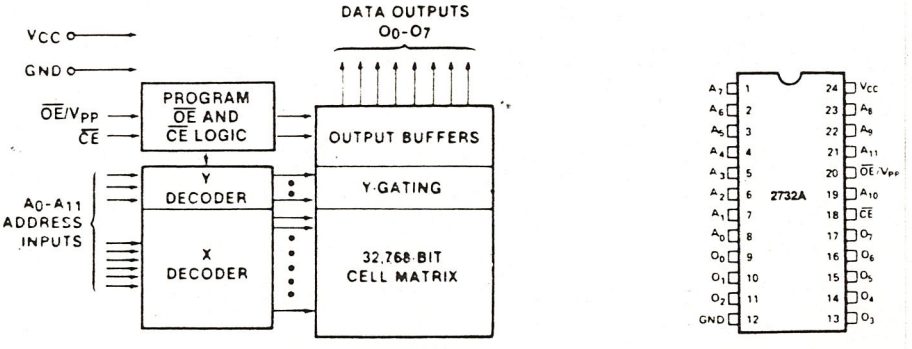
บทที่ 1 ทฤษฎีและหลักการ

ส่วนประกอบของวงจรควบคุมการทำงาน

1. 8031 วงจรสร้างสัญญาณนาฬิกาภายใน 8031 จะถูกควบคุมความถี่ของการออสซิลเลตด้วยคริสตอล X1 ความถี่ 6 MHz ดังนั้นสัญญาณนาฬิกาภายใน 8031 จะมีความถี่ 6 MHz ขา Reset ถูกต่อด้วยตัวสวิทชิ่งทรานซิสเตอร์ 2SC458 ซึ่งต่ออยู่กับตัวต้านทาน และตัวเก็บประจุ พร้อมทั้งต่อกับสวิทช์ เมื่อสวิทช์ถูกกดทำให้โวลท์เตจที่ขา Reset มีค่าลดลง เมื่อโวลเตจลดลงจนกระทั่งถึงค่าเป็นระดับลอจิก 0 8031 ก็จะออกจากการรีเซตและเริ่มทำงานที่โปรแกรมตำแหน่ง 0000H ในหน่วยความจำสำหรับโปรแกรม วงจรข้างต้นได้ต่อขา $\overline{EA}/V_{PP}$ ไว้ที่กราวด์หรือลอจิก 0 เป็นการบอกกับ 8031 ว่าคำสั่งที่เก็บไว้ในหน่วยความจำตำแหน่ง 0000H ถึง 0FFFH อยู่ภายนอกตัว 8031 ดังนั้นจะต้องเขียนโปรแกรมเก็บไว้ในหน่วยความจำสำหรับโปรแกรมภายนอก 8031 เริ่มจากตำแหน่ง 0000H เพื่อว่าเมื่อ 8031ออกจากสภาวะการรีเซตจะสามารถอ่านคำสั่งเข้าไปทำงานได้

2. 2732 เป็น EPROM ขนาด 4Kx8 บิท หรือ 4 กิโลไบท์ ซึ่งการบอกขนาดความจุของหน่วยความจำมักจะบอกเป็นเลขสองชุดคูณกันอยู่ โดยชุดแรกเป็นจำนวนตำแหน่งของหน่วยความจำที่สามารถใช้กับข้อมูล และตัวเลขชุดหลังเป็นจำนวนบิทของข้อมูลต่อหนึ่งตำแหน่งของหน่วยความจำ 2732 บรรจุอยู่ในตัวถังแบบ Dual Inline Package (DIP) ที่มีขาต่อออกมาภายนอกสองแถวแถวละ 12 ขา แต่ละขาถูกใช้งานต่างกันดังรูป 1.1 A_0 ถึง A_{11} เป็นขาสัญญาณที่ใช้ชี้ตำแหน่ง(Address) ของหน่วยความจำ ในกรณีนี้มีทั้งหมด 12 ขาจึงชี้ตำแหน่งหน่วยความจำได้ 2^{12}=4096 หรือ 4K ตำแหน่ง O_0 ถึง O_7 เป็นขาที่ใช้ส่งข้อมูลจากตำแหน่งที่ชี้โดยสัญญาณที่ขา A0 ถึง A11 ออกมาภายนอก 2732 ดังนั้นข้อมูลนี้จะส่งออกได้ทีละ 8 บิท ขา V_{CC} เป็นจุดที่ต้องป้อนไฟเลี้ยง +5โวลท์และGND เป็นจุดที่ต้องต่อกับกราวด์ของวงจร ส่วนขา $\overline{CE}$ Chip Enable ถ้าเป็น0เป็นการเลือกการติดต่อกับ 2732 ในระหว่างนี้ข้อมูลจากตำแหน่งที่ต้องการจะถูกเตรียมไว้ที่ Output Buffer เพื่อส่งออก Output Buffer เป็นวงจรที่มีเอาท์พุทแบบ 3-state ควบคุมโดยสัญญาณจากขา $\overline{OE}$ ถ้าสัญญาณที่ขานี้เป็นลอจิก 1 จะทำให้ Output Buffer อยู่ในสภาวะHigh Impedance แต่ถ้าสัญญาณที่ขา $\overline{OE}$ เป็น 0 จะทำให้ข้อมูลที่

Output Buffer ถูกส่งออกมาภายนอก เมื่อเสร็จสิ้นการอ่านข้อมูลจะให้สัญญาณที่ขา $\overline{CE}$ และ $\overline{OE}$ เป็น 1



รูปที่ 1.1

3. 6264 เป็น Static RAM ขนาด 8Kx8 บิท หรือ 8 กิโลไบท์ หน่วยความจำนี้สามารถเขียนหรืออ่านข้อมูลภายในได้โดยการป้อนสัญญาณเข้าไปที่ 6264 6264 นี้บรรจุอยู่ในตัวถังแบบ DIP ขนาด 28 ขา แบ่งออกเป็นสองแถวๆละ 14 ขา ขา V_{cc} มีไว้สำหรับป้อนไฟเลี้ยง 5 โวลท์ และขา GND ไว้สำหรับต่อกับกราวด์ของวงจร

ขา A_0 ถึง A_{12} รวมเป็น 13 ขา จะใช้สำหรับป้อนค่าตำแหน่งหน่วยความจำที่ต้องการติดต่อด้วยแอดเดรส 13 เส้นสามารถชี้หน่วยความจำได้ 2^{13}=8K ตำแหน่ง ขา I/O_0 ถึง I/O_7 ใช้สำหรับส่งข้อมูลเข้าไปยัง 6264 หรือรับข้อมูลออกจาก 6264 ส่วนสัญญาณ $\overline{CS}_1$, CS_2, $\overline{OE}$ และ $\overline{WE}$ ใช้ควบคุมการอ่าน, เขียนข้อมูลกับ 6264

การอ่านข้อมูล จะเริ่มจากการป้อนค่าตำแหน่งหน่วยความจำเข้าไปยังขา A_0 ถึง A_{12} จากนั้นสัญญาณที่ขา $\overline{CS}_1$ และ CS_2 จะต้องเปลี่ยนจาก 1 เป็น 0 และ 0 เป็น 1 ตามลำดับขณะนี้ข้อมูลถูกอ่านจาก Memory Matrix มาเก็บไว้ใน Output Data Buffer ขณะนี้สัญญาณที่ขา $\overline{OE}$ ยังเป็น 1 ทำให้เอาท์พุทของ RAM อยู่ในสภาวะ High Impedance จากนั้นเมื่อสัญญาณ $\overline{OE}$ เป็นสภาวะลอจิก 0 จะทำให้ข้อมูลใน Output Buffer ถูกส่งออกมา เมื่ออ่านข้อมูลจากหน่วยความจำเสร็จแล้วก็จะเลิกการทำงานโดยให้สัญญาณ $\overline{CS}_1$ และ $\overline{OE}$ กลับเป็น 1 , CS_2 กลับเป็น 0 ขา I/O_0 ถึง I/O_7

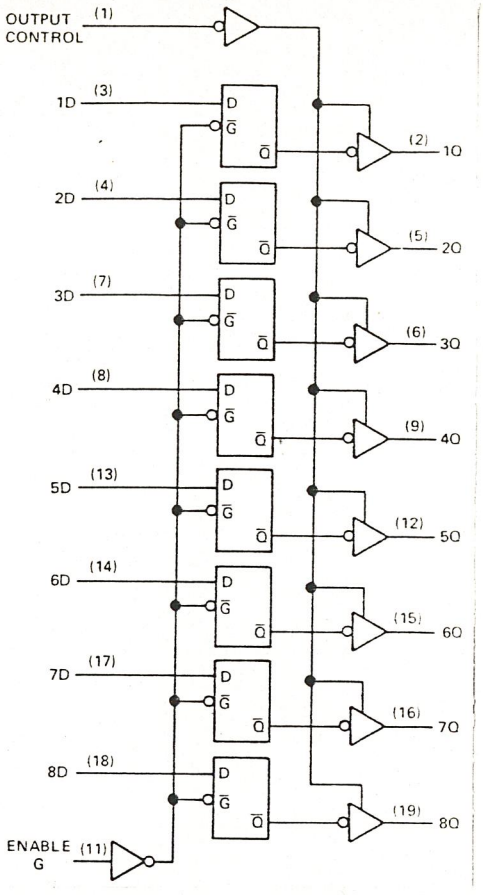
ของ 6264 ก็กลับเข้าสู่สภาวะ High Impedance

การเขียนข้อมูลจะเริ่มจากการป้อนค่าตำแหน่งหน่วยความจำเข้าไปยังขา A_0 ถึง A_{12} ของ 6264 จากนั้นจะให้สัญญาณที่ขา $\overline{OE}$ มีสภาวะลอจิก 1 ตลอดเวลาของการเรียกข้อมูลแล้วป้อนสัญญาณที่มีลอจิก 0 เข้าที่ขา $\overline{CS_1}$ และ $\overline{WE}$, ลอจิก 1 ที่ขา CS_2 ข้อมูลที่ต้องการนำไปเก็บ (Data In) จะถูกป้อนเข้าไปทางขา I/O_0 ถึง I/O_7 ขณะนี้ข้อมูลที่ป้อนเข้าไปจะถูกนำเข้าไปเก็บในหน่วยความจำ จากนั้นให้ป้อนสภาวะลอจิก 1 เข้าไปที่ขา $\overline{CS_1}$ และ $\overline{WE}$, ลอจิก 0 ที่ขา CS_2 ก็จะสิ้นสุดการเขียนข้อมูลเข้าไปยัง 6264

ในการติดต่อกับหน่วยความจำนั้น ค่าตำแหน่งหน่วยความจำต้องคงที่ตลอดเวลาการติดต่อ แต่ไดอะแกรมเวลาของการอ่านหรือเขียนข้อมูลกับหน่วยความจำภายนอกของ 8031 นั้น ค่าตำแหน่งหน่วยความจำจะถูกส่งออกมาทางพอร์ท 0 และพอร์ท 2 โดยค่าตำแหน่ง 8 บิทบนจะส่งออกมาทางพอร์ท 2 ตลอดเวลาการอ่านหรือเขียนข้อมูล แต่ค่าตำแหน่ง 8 บิทล่างจะส่งออกมาทางพอร์ท 0 เพียงเวลาหนึ่งเท่านั้น จึงจำเป็นที่จะต้องใช้อุปกรณ์ภายนอกทำการ Latch เพื่อรักษาตำแหน่งหน่วยความจำที่ออกมาทางพอร์ท 0 ไว้ตลอดเวลาการติดต่อ

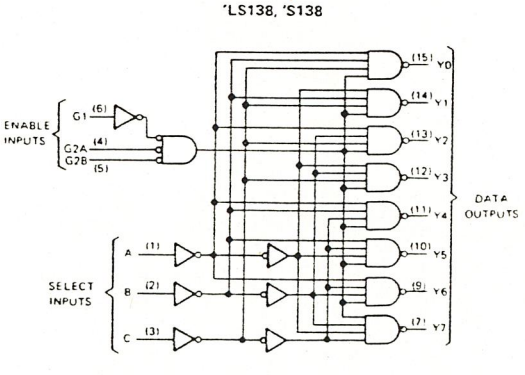
4. 74LS373 เป็น D-FF 8 ตัวต่อขนานกันอยู่ทำให้สามารถ latch ข้อมูลได้ 8 สัญญาณดังรูป 1.2 ข้อมูลที่ปรากฏตรงขา D_0 ถึง D_7 ของ 74LS373 จะถูก Latch ข้อมูลเข้าที่ขา G เพื่อให้เกิดการ Latch ค่าตำแหน่งหน่วยความจำที่ขอบขาลงของสัญญาณ ALE ภาคเอาท์พุทของ 74LS373 เป็น Tristate Buffer ที่ควบคุมด้วยสัญญาณจากขา $\overline{OC}$ (Output Control)

ถ้าสัญญาณที่ขา $\overline{OC}$ เป็นลอจิก 1 จะทำให้ขา Q_0 ถึง Q_7 อยู่ในสภาวะ High Impedanceแต่ถ้าขาสัญญาณที่ขา $\overline{OC}$ มีลอจิก 0 จะทำให้ข้อมูลจากขา Q ของ D-FF ทุกตัวถูกส่งออกมาทางขา Q_0 ถึง Q_7 ดังนั้นในรูปวงจร จึงต่อขา $\overline{OC}$ ลงกราวด์ หรือลอจิก 0 พอร์ท 2 ซึ่งส่งค่าตำแหน่งหน่วยความจำ 8 บิทบนออกมา (A_8 ถึง A_{15}) จะต่อเข้ากับ A_8 ถึง A_{11} ของ 2732 และต่อเข้ากับ A_8 ถึง A_{12} ของ 6264



รูปที่ 1.2

5. 74LS138 เป็นวงจรถอดรหัสซึ่งทำหน้าที่เปลี่ยนรหัสจากเลขฐานสองเป็นรหัสอื่น 74LS138 เป็นตัวถอดรหัสแบบ 3 to 8 Line Decoders ทำการถอดรหัสตำแหน่งหน่วยความจำ รูปที่ 1.3 เป็นไดอะแกรมภายในและตารางฟังก์ชันของ 74LS138



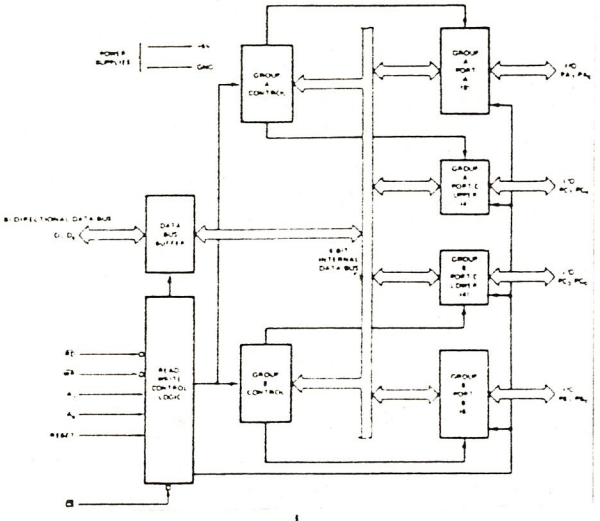
'LS138, 'S138
FUNCTION TABLE

INPUTS					OUTPUTS							
ENABLE		SELECT										
G1	G2*	C	B	A	Y0	Y1	Y2	Y3	Y4	Y5	Y6	Y7
X	H	X	X	X	H	H	H	H	H	H	H	H
L	X	X	X	X	H	H	H	H	H	H	H	H
H	L	L	L	L	L	H	H	H	H	H	H	H
H	L	L	L	H	H	L	H	H	H	H	H	H
H	L	L	H	L	H	H	L	H	H	H	H	H
H	L	L	H	H	H	H	H	L	H	H	H	H
H	L	H	L	L	H	H	H	H	L	H	H	H
H	L	H	L	H	H	H	H	H	H	L	H	H
H	L	H	H	L	H	H	H	H	H	H	L	H
H	L	H	H	H	H	H	H	H	H	H	H	L

*G2 = G2A + G2B
H = high level, L = low level, X = irrelevant

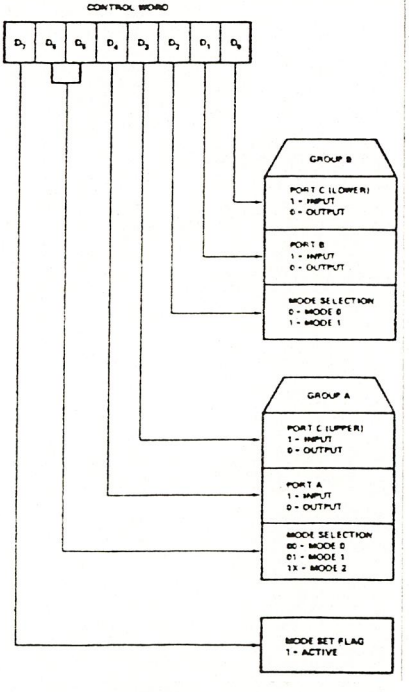
รูปที่ 1.3

6. 8255 เป็นอุปกรณ์ I/O พอร์ทแบบหนึ่งที่เรียกว่า Programable Peripheral Interface (PPI) มีโครงสร้างภายในดังรูปที่ 1.4



รูปที่ 1.4

จากรูปแสดงให้เห็นว่า 8255 จะมีพอร์ททั้งหมด 24 บิทแบ่งออกเป็นสองกลุ่มคือ Group A และ Group B Group A ประกอบด้วย Port A จำนวน 8 บิทและ 4 บิทบนของพอร์ท C ส่วน Group B ประกอบด้วย Port B จำนวน 8 บิทและ 4 บิทล่างของพอร์ท C พอร์ททั้งหมดจะสามารถกำหนดให้เป็นพอร์ทรับข้อมูลหรือส่งข้อมูลออกก็ได้ โดยการเขียนคำสั่งข้อมูลไปยังพอร์ทควบคุมจะทำให้ Port A, B และ C เป็นแบบใดดังรูปที่ 1.5



รูปที่ 1.5

เนื่องจากภายในของ 8255 ประกอบด้วยพอร์ทถึง 4 พอร์ทคือ Port A,B,C และ Control Port ดังนั้นการที่จะติดต่อกับพอร์ทใดจะต้องมีค่าตำแหน่งของพอร์ทนั้นเหมือนกับการชี้ตำแหน่งหน่วยความจำนั่นเอง โดยขา A0 และ A1 ที่ใช้เป็นค่าตำแหน่งสำหรับชี้พอร์ทที่ต้องการติดต่อด้วยดังนี้

A1	A0	Port
0	0	A
0	1	B
1	0	C
1	1	Control

การออกแบบให้ 8031 ติดต่อกับ 8255 นั้นจะต้องทำการออกแบบให้ 8031 มอง 8255 เหมือนกับเป็นหน่วยความจำสำหรับข้อมูลขนาด 4 ไบท์ที่ 8031 สามารถอ่านหรือเขียนข้อมูลได้

อุปกรณ์กำเนิด Pulse/Tone ในการหมุนหมายเลข

MC145412

ไอซีเบอร์ MC145412 เป็นวงจรรวมซึ่งจะเปลี่ยนอินพุทจากคีย์บอร์ดมาเป็น Pulse หรือ DTMF ได้ ที่เอาท์พุท

คุณสมบัติที่สำคัญ

- สามารถใช้กับคีย์บอร์ดได้ทั้งแบบ 3x4 หรือ 4x4
- มีขาที่ใช้สวิชท์ระหว่าง DTMF และแบบ Pulse 10 pps และ 20 pps
- ในโหมดของการใช้ Pulse DialingจะมีสัญญาณTone ความถี่ 500Hz ที่ขาTSO
- มีหน่วยความจำที่ใช้เก็บตัวเลข 18 หลัก 10 หมายเลขและสามารถเรียกหมายเลขซ้ำได้
- ใช้คริสตอลความถี่ 3.579545 MHz
- จ่ายไฟโดยทางสายโทรศัพท์
- เป็นวงจรรวม ชนิด Silicon Gate CMOS ใช้ไฟเลี้ยงต่ำ ประมาณ 1.7-5.5 V
- สามารถใช้ส่งสัญญาณ DTMF ได้ หรือใช้ส่งเป็น Pulse ของหมายเลขได้
- มีการ Mute เพื่อใช้แยกการรับเสียงออกจากการส่งหมายเลขที่เอาท์พุท
- สามารถจัดการหน่วยความจำภายในได้โดยใช้คีย์บอร์ด

ขาต่างๆที่ใช้งาน

V_{DD}, V_{SS} - Power Supply (ขา1,ขา6)

ไฟเลี้ยง DC จะป้อนเข้าที่ขาทั้งสองโดยที่ขา V_{DD} เป็นขั้วบวกและมีช่วงโวลท์เตจระหว่าง 1.7-5.5 V

MS - Mode Selection (ขา10)

ขา MS มี 3 สถานะอินพุทที่เปลี่ยนได้คือ DTMF, Pulse Dialing โหมด 10 ppsและ 20 pps การต่อขานำไปใช้งานโดย ถ้าขา MS ต่อกับ V_{DD} จะเป็นโหมด 20 ppsPulse Dialing, ถ้าปล่อย Open จะได้เป็นโหมด 10pps Pulse Dialing และถ้าต่อกับ V_{SS} จะเป็นโหมด DTMF Dialing

OH - On Hook (ขา12)

ถ้าขา OH ต่อกับ V_{DD} หรือปล่อยลอยไว้จะเป็นโหมด On-Hook แต่ถ้าขา OH ต่อกับ V_{SS} จะเป็นการเลือกโหมด Off-Hook

โดยในการใช้งานจะใช้ขานี้เป็นขา chip select สำหรับ MC145412

TSO - Tone Signal Output (ขา17)

TSO จะกำเนิดสัญญาณ Tone 500 Hz เมื่อมีการกดคีย์บอร์ดในโหมดของ Pulse และจะไม่ให้สัญญาณ 500 Hz เมื่ออยู่ในโหมด DTMF

DTMF Out - Dual Tone Multifrequency Output (ขา18)

เมื่อขา MS ถูกเซทที่ V_{SS}, ขา DTMF Out จะกำเนิดสัญญาณตามคีย์ที่กด ซึ่งจะสอดคล้องกับ Row และ Column ในโหมดของ Pulse Dialing (MS = V_{DD} หรือ Float) และ ระหว่างการ On-Hook นั้น ขา DTMF Out จะอยู่ในสภาพ High Impedance ถ้ามี Tone เอาท์พุทขานี้จะมี DC ไบอัส $(V_{DD}-V_{SS})/2$

$\overline{OPL}$ - Out Pulsing (ขา17)

ขานี้จะให้ Pulse 10 pps (เมื่อMS=Open) หรือ 20 pps (MS=V_{DD}) ไอซี MC145412 มีอัตราส่วน Make/Break เท่ากับ 40/60 สำหรับในโหมดของ DTMF Dialing (MS = V_{SS}) นั้น ขา $\overline{OPL}$ จะอยู่ในสภาวะ High Impedance และ ระหว่างการ On-Hook นั้น ขานี้จะไม่มี Pulse ออกไป

$\overline{MO}$ - Mute Output (ขา11)

ขา Mute Output ต่อแบบ Open-drain N-Cnannel Output ซึ่งจะ Pull กับV_{SS} ระหว่างที่ขา $\overline{OPL}$ มี Output ออกไป, ระหว่างที่ Off-Hook มีการกดคีย์ และในโหมด Memory Dialing ของโหมด DTMF

Keyboard Input (ขา2,3,4,5,13,14,15,16)

คีย์บอร์ดอินพุทจะใช้ชนิดหน้าสัมผัสเดียว (Class A) หรือคีย์บอร์ดมาตรฐาน 2-of-8 หรือ 2-of-7 ซึ่ง V_{SS} จะต่ออยู่กับ Common

OSC_{in}, OSC_{out} (ขา8,9)

ใช้คริสตอล 3.579545 MHz สำหรับเป็นตัวออสซิลเลเตอร์ การไบอัสคริสตอลจะกระทำโดยค่าความต้านทานและความจุไฟฟ้าภายในไอซี

ส่วนของ Speech Network

MC34014

ไอซี MC34014 ประกอบไปด้วยส่วนสำคัญๆคือวงจรรับสัญญาณ วงจรส่งสัญญาณ วงจรควบคุมความดังของเสียงพูดของตัวเอง (Sidetone Amplifier) วงจรเชื่อมต่อกับไฟกระแสตรง (DC Loop Interface) และวงจร Regulator วงจรควบคุมเสียงพูดที่เชื่อมต่อกับสายส่งสัญญาณจากวงจร Bridge Rectifier จะมีอุปกรณ์พวกความต้านทานภายนอกเพื่อปรับอัตราขยายของวงจรรับส่งสัญญาณ และวงจรควบคุมระดับความดังของเสียงพูดของตัวเอง ตลอดจนการปรับผลตอบสนองความถี่ด้วย

ไอซี MC34014 มีทั้งชนิด DIP 18ขา, Surface Mount PLCC 20ขา และ SOIC 20ขา

คุณสมบัติจำเพาะ

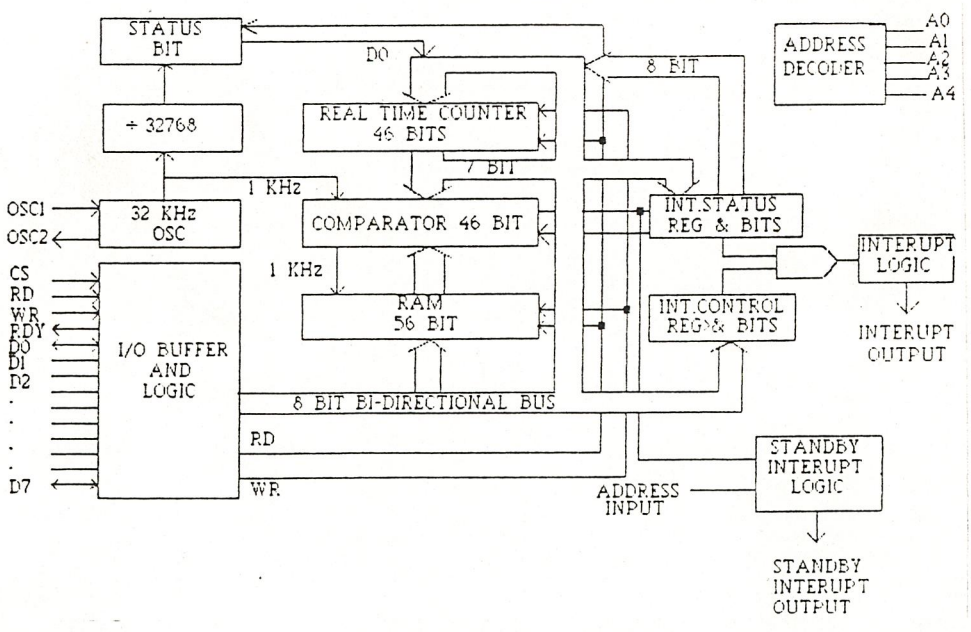
- ค่า Gain ของการส่ง,การรับและ Sidetone จะปรับโดยค่าความต้านทานภายนอก
- สามารถทำงานได้ที่โวลท์เตจต่ำสุด 1.5โวลท์ (V_+) ใน Speech Mode
- ตัว Speech Amplifier จะ Mute(เงียบ)ระหว่างที่มีการหมุนหมายเลขโดย Pulse หรือ DTMF
- การปรับระดับของ DTMF Output ทำได้โดยปรับความต้านทานเพียงตัวเดียว
- ใช้ได้กับไมโครโฟนชนิด 2-Terminal Electret Microphone
- ใช้ได้กับลำโพงรับเสียงที่มี Impedance 150 Ω หรือมากกว่า

สำหรับรายละเอียดของขาต่างๆดูได้จาก Data Sheet

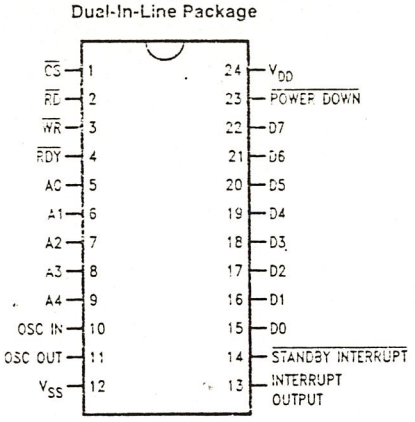
Real Time Clock (RTC)

ทฤษฎี

RTC คือ REAL TIME CLOCK หรือนาฬิกาบอกเวลาที่แท้จริง (เวลาที่เราใช้กันเป็นประจำ) ให้กับไมโครโปรเซสเซอร์ โดยการเชื่อมต่อเข้ากับ CPU ก็สามารถทำให้ CPU สามารถรู้เวลาได้ทุกขณะ โดยที่ CPU สามารถที่จะอ่านเวลาจากตัว RTC ได้ทุกครั้งที่ CPU ต้องการจึงจะทำให้ CPU สามารถไปทำงานอื่นได้ RTC นี้จะกินไฟน้อยมากเมื่อเราต่อ BATTERY สำรองให้กับมันจะทำให้ใช้ RTC ทำงานอยู่ได้นานเป็นเดือน แต่ก่อนที่จะทำให้ RTC ทำงานเราต้องให้ CPU เขียนข้อมูลเกี่ยวกับเวลาจริงให้กับมันเสียก่อน เมื่อเรียบร้อยแล้วเราเลิกจ่ายไฟเลี้ยง RTC ก็ยังทำงานเป็นนาฬิกาเหมือนทั่วๆไป (แต่ต้องต่อ BATTERY สำรองด้วย) คือสามารถบอกเวลาได้ตั้งแต่ วินาที, นาที, ชั่วโมง, วันในรอบสัปดาห์ (อาทิตย์-เสาร์), เดือน และปี ให้แก่ไมโครโปรเซสเซอร์ได้อย่างเที่ยงตรง โดยที่ CPU เพียงแต่ติดต่อกับ RTC เหมือนกับติดต่อกับหน่วยความจำหรือพอร์ทเท่านั้น RTC มีอยู่ด้วยกันหลายเบอร์ เช่น MM58167, MM58174, MM58274 และในที่นี้เราจะกล่าวถึงเบอร์ MM58167 ด้วยเหตุผลที่ว่าต่อกับ 8031ได้ง่ายโปรแกรมได้ง่ายและข้อมูลที่เขียนเข้าไปไม่สูญหายง่ายจากการเลิกจ่าย SUPPLY



รูปที่ 1.6 แสดงโครงสร้างภายในของ MM58167



รูปที่ 1.7 แสดงการจัดขาของ MM58167

หน้าที่ขาต่างๆ

$\overline{CS}$ CHIP SELECT เป็นขาเลือกให้ IC ทำการติดต่อกับ CPU หรือทำงานได้ ACTIVE 0

$\overline{RD}$ ACTIVE 0 สำหรับอ่านข้อมูลจาก RTC

$\overline{WR}$ ACTIVE 0 สำหรับเขียนข้อมูลเข้าไปยัง RTC

A0-A4 เป็นสาย ADDRESS ติดต่อกับ RTC เพื่อให้ CPU เขียนข้อมูลหรืออ่านข้อมูลในตำแหน่งที่มีหน้าที่นั้นๆ (จะกล่าวอีกทีในตารางหน้าที่)

OSCIN,OUT ใช้ต่อกับ X'TAL 32.768 KHz หรือจะนำความถี่จากภายนอกป้อนเข้าที่ OSCIN ก็ได้

INTERRUPT OUTPUT เป็นขาสัญญาณใช้ในการ INTERRUPT โดยการ INTERRUPT สั่งได้ทาง SOFTWARE ว่าจะ INTERRUPT ทุกๆเวลาเท่าไรสัญญาณออกเป็น HIGH

$\overline{\text{STANDARD INTERRUPT}}$ จะให้สัญญาณ LOW ก็ต่อเมื่อ RTC มีการเปรียบเทียบกันระหว่าง RAM กับตัวนับเวลาเมื่อเท่ากัน

D0-D7 สายข้อมูลที่จะทำการติดต่อ CPU

$\overline{\text{POWER DOWN}}$ ถือเป็นขา CHIP SELECT อีกขาหนึ่ง ACTIVE 0 คือมันจะทำให้ RTC ไม่ตอบสนองต่อสัญญาณภายนอกใดๆ ซึ่งเป็นการป้องกันเวลาสูญเสียจากสัญญาณอื่นๆ โดยเวลาที่ตั้งไว้จะยังคงเดินตามปกติและยังให้สัญญาณ $\overline{\text{STANDBY INT}}$ ด้วยถ้ามีการ SET

จากโปรแกรมไว้

Vss ไฟลบ

V_{DD} ไฟบวก MIN 2V POWER DOWN, 4V OUTPUTS ENABLE และ MAX 5.5 V

จากโครงสร้างส่วนที่สำคัญที่จะกล่าวถึงก็คือ RTC และ RAM

COUNTER ADDRESS	UNIT D0 D1 D2 D3	MAX BCD CODE	TENS D4 D5 D6 D7	MAX BCD CODE
1/10,000 OF SEC (00h)	- - - -	-	D4 D5 D6 D7	9
HUNDS & TENS SEC (01h)	D0 D1 D2 D3	9	D4 D5 D6 D7	9
SECOND (02h)	D0 D1 D2 D3	9	D4 D5 D6 -	5
MINUTE (03h)	D0 D1 D2 D3	9	D4 D5 D6 -	5
HOURS (04h)	D0 D1 D2 D3	9	D4 D5 - -	2
DAY OF THE WEEK (05h)	D0 D1 D2 -	7	- - - -	0
DAY OF THE MONTH (06h)	D0 D1 D2 D3	9	D4 D5 - -	3
MONTH (07h)	D0 D1 D2 D3	9	D4 - - -	1

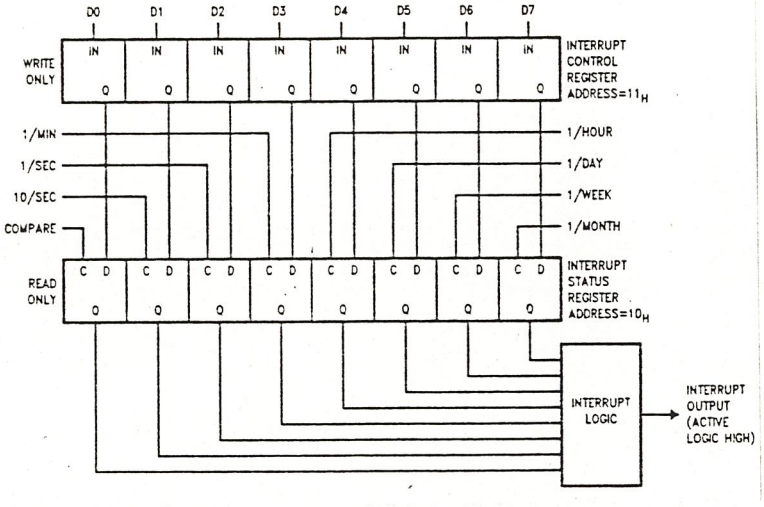
REAL TIME COUNTER เป็นตัวนับและจัดการเกี่ยวกับเวลาถูกแบ่งเป็นเลข BCD 2หลัก และบิทไหนที่ไม่ได้ใช้จะมีค่าเป็น 0 เช่น ที่ DAY OF WEEK (วันในสัปดาห์) คือในหนึ่งสัปดาห์มีแค่ 7 วัน เพราะฉะนั้นเลขไม่เกินหลักหน่วย หลักสิบจึงไม่ต้องสนใจ

RAM ใช้เก็บข้อมูลเมื่อไฟตก หรือใช้เก็บข้อมูลการตั้งปลุก เพื่อที่จะนำมาเปรียบเทียบ กับ REAL TIME COUNTER ซึ่ง RAM นี้จะถูกจัดให้มีรูปแบบเหมือนกับ REAL TIME COUNTER ดังตาราง

ตารางหน้าที่

A4	A3	A2	A1	A0	FUNCTION
0	0	0	0	0	COUNTER THOUSANDTHS OF SECONDS
0	0	0	0	1	COUNTER HUNDS AND TENTHS OF SECOND
0	0	0	1	0	COUNTER SECONDS
0	0	0	1	1	COUNTER MINUTES
0	0	1	0	0	COUNTER HOURS
0	0	1	0	1	COUNTER DAY OF WEEK
0	0	1	1	0	COUNTER DAY OF MONTH
0	0	1	1	1	COUNTER MONTH
0	1	0	0	0	RAM THOUSANDTHS OF SECONDS
0	1	0	0	1	RAM HUNDS AND TENTHS OF SECONDS
0	1	0	1	0	RAM SECONDS
0	1	0	1	1	RAM MINUTES
0	1	1	0	0	RAM HOURS
0	1	1	0	1	RAM DAY OF WEEK
0	1	1	1	0	RAM DAY OF MONTH
0	1	1	1	1	RAM MONTH
1	0	0	0	0	INTERUPT STATUS REGISTER
1	0	0	0	1	INTERUPT CONTROL REGISTER
1	0	0	1	0	COUNTER RESET
1	0	0	1	1	RAM RESET
1	0	1	0	0	STATUS BIT
1	0	1	0	1	"GO" COMMAND
1	0	1	1	0	STANDBY INTERUPT
1	1	1	1	1	TEST MODE

ตารางนี้จะมีส่วนของแอดเดรสที่จะติดต่อด้วยอยู่ด้านซ้าย และหน้าที่ของแต่ละตำแหน่งด้านขวา เช่น เราต้องการอ่านวินาทีเข้ามาที่ CPU จากที่กล่าวมาแล้วว่า ตัวกระทำการนับเวลาอยู่ที่ COUNTER ดังนั้นเราก็อ่านแอดเดรส 00010 COUNTER SECONDS คือ ADDRESS 02H นั่นเอง ก็เป็น IN A,(02H) และถ้าเราต้องการตั้งเวลาสำหรับปลุกบ้างก็เขียนเข้าไปที่ RAM เช่น กำหนดวินาทีไว้ที่ RAM ก็เป็น OUT RAM SECONDS = OUT (0AH),A



รูปที่ 1.8 แสดง INTERRUPT COMPARATOR

ส่วนมากแล้วในเวลาที่เราจะให้ CPU แสดงผลเวลาออกมา วิธีที่ง่ายก็คือให้ RTC ส่งสัญญาณ INTERRUPT มาทุกๆ 1 วินาที CPU ก็จะอ่านเวลาจาก RTC แล้ว OUT ออกไปที่ DISPLAY RTC ตัวนี้มี INTERRUPT 2 อย่างคือ INTERRUPT OUTPUT โดยสามารถโปรแกรมการ INTได้ 8 อย่างคือ 10Hz, 1Hz 1นาที/ครั้ง, 1ชั่วโมง/ครั้ง, 1วัน/ครั้ง, 1สัปดาห์/ครั้ง และเมื่อ RAM กับ REAL TIME COUNTER เกิดการเปรียบเทียบขึ้นก็จะเกิด STAND BY INTERRUPT วิธีการจะให้เกิดการ INTERRUPT ขึ้นอยู่กับการ

ให้ ENABLE LOGIC 1 ไปยัง INTERRUPT CONTROL REGISTER (ADDRESS 11H) ในบิทตรงกับความถี่ที่เราต้องการ เช่นให้ INT ทุกๆ 1 วินาที ก็ดูรูป 1/SEC ตรงกับบิทไหน ในที่นี้คือ D2 ก็เป็น

```
LD  A, 04H
OUT (11H), A
```

และเรายังสามารถ SET การ INTERRUPT ได้มากกว่า 1 บิท เช่นจะให้ INTERRUPT ทุกๆ วินาที และทุกๆชั่วโมงก็เป็น 14H

ส่วนการ STAND BY INTERRUPT (หรือการ INTERRUPT เมื่อการเปรียบเทียบ 8 ADDRESS ระหว่าง COUNTER กับ RAM ต้องเท่ากัน) ต้องให้บิท 1 เป็น 1 ที่แอดเดรส 16H

ส่วนการ DISABLE INTERRUPT ก็ส่ง 0 ไปที่บิทนั้น

INTERRUPT STATUS REGISTER ใช้สำหรับอ่านสัญญาณ INTERRUPT ว่ามาจากบิทใด และจะเป็นการ DISABLE การ INTERRUPT อีกด้วย

COUNTER AND RAM RESET โดยการเขียน OFFH ไปที่แอดเดรส 12H และ 13H เมื่อ COUNTER ถูก RESET ส่วนที่เปลี่ยนแปลงที่ใช้เวลาน้อยจะเป็น 0 ซึ่งขณะส่วนวันของสัปดาห์ วันของเดือน และเดือน จะถูกปรับเป็น 1 แต่ RAM จะเป็น 0 ทั้ง 8 ADDRESS

GO COMMAND โดยการให้ PULSE ของการเขียนไปที่ ADDRESS 15H เป็นการกำหนดให้นาที มีความถูกต้องก่อนที่จะเริ่มนับ COUNTER หรือเริ่ม CLOCK นั่นเอง

STATUS BIT

จะบอกถึงความถูกต้องของการอ่าน COUNTER โดย LOGIC ที่อ่านได้ที่บิท 0 จะเป็น 1 ส่วนบิทอื่นจะเป็น 0 หมด แต่ถ้าอ่านค่าจาก COUNTER มาไม่ถูกต้อง ที่บิท 0 นี้จะมีค่าเป็น 0

TEST MODE

ใช้ทดสอบ RTC ให้ทำงานสูงกว่าความถี่ปกติ

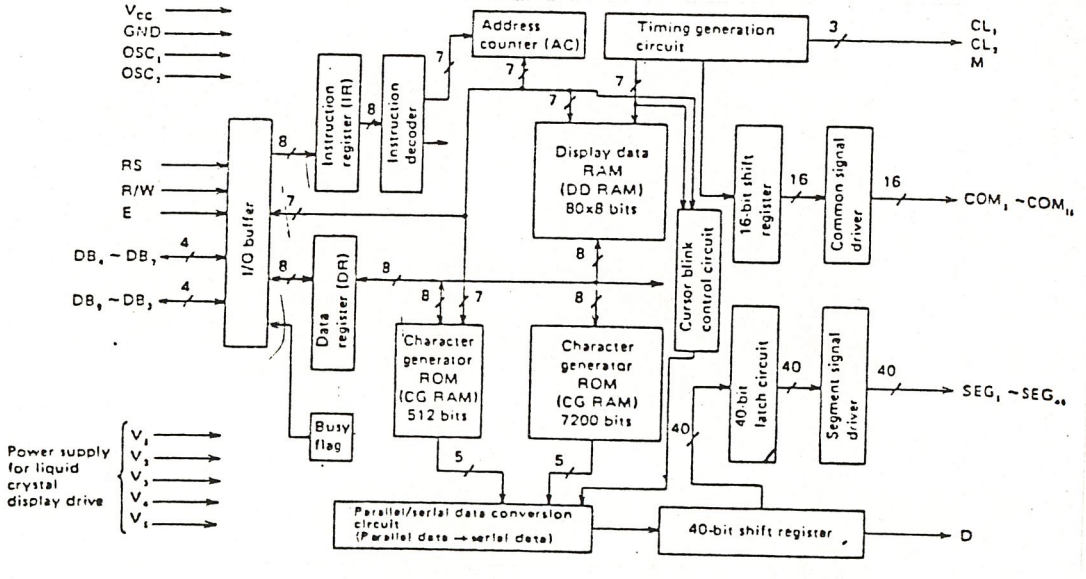
Dot Matrix LCD Module

อุปกรณ์ส่วนใหญ่ในปัจจุบันนิยมใช้ส่วนแสดงผลเป็นจอ LCD เนื่องจากใช้งานได้สะดวก โดยที่ Dot Matrix LCD Module แบ่งเป็นชนิดต่างๆได้ดังนี้

1. Character LCD Module
2. Graphic LCD Module
3. Segment Display Type LCD Module

ในแต่ละแบบจะประกอบด้วยส่วนใหญ่ๆดังนี้

1. Dot Matrix LCD เป็นตัวแสดงผลเปิด-ปิดตัวเองกับแสง
2. Driver เป็นตัวรับสัญญาณควบคุมมาขับ LCD นิยมใช้เบอร์ HD44100H
3. Controller เป็นตัวรับข้อมูลจากภายนอกและควบคุม LCD Module ให้แสดงผล เบอร์ที่นิยมใช้คือ HD44780 ซึ่งจะใช้ในแบบ Character LCD Module HD44780 เป็น LSIใช้ควบคุม LCD สามารถใช้งานแบบ 4 บิท หรือ 8 บิทได้



รูปที่ 1.9 โครงสร้างภายในของ HD44780

รายละเอียดคำสั่งของ HD44780

1. Clear Display เป็นการเขียนช่องว่าง(Space) ลง DD RAM ทั้งหมด และเซท DD RAM Addresser เป็นศูนย์ เคอร์เซอร์จะกลับไปอยู่ตำแหน่งบนซ้ายสุด

RS	$R/\overline{W}$	DB_7	DB_6	DB_5	DB_4	DB_3	DB_2	DB_1	DB_0
0	0	0	0	0	0	0	0	0	1

2. Return Home จะเซท DD RAM Addresser เป็นศูนย์ เคอร์เซอร์ไปอยู่ตำแหน่งบนซ้ายสุด จอภาพจะไม่เปลี่ยน

RS	$R/\overline{W}$	DB_7	DB_6	DB_5	DB_4	DB_3	DB_2	DB_1	DB_0	
0	0	0	0	0	0	0	0	1	*	(no effect)

3. Entry Mode Set

RS	$R/\overline{W}$	DB_7	DB_6	DB_5	DB_4	DB_3	DB_2	DB_1	DB_0
0	0	0	0	0	0	0	1	I/D	S

บิท I/D : เป็นตัวกำหนดให้เมื่อเขียนหรืออ่านข้อมูลจะทำให้ DD RAM Address เพิ่มหรือลดโดย 1=เพิ่ม 0=ลดลงหนึ่ง

บิท S : ถ้า S=1 เมื่อใส่ข้อมูลแล้วเคอร์เซอร์อยู่กับที่ ข้อมูลจะถูกดันไปทางซ้าย ถ้า S=0 ข้อมูลจะอยู่กับที่ เคอร์เซอร์ถูกดันไปทางขวา

4. Display ON/OFF Control

RS	$R/\overline{W}$	DB_7	DB_6	DB_5	DB_4	DB_3	DB_2	DB_1	DB_0
0	0	0	0	0	0	0	1	D	C	B

บิท D : จะเปิด/ปิดจอภาพโดย D=1จะ ON และ D=0 จะ OFF

บิท C : C=1 จะแสดงเคอร์เซอร์ C=0 จะไม่แสดงเคอร์เซอร์

บิท B : B=1 เคอร์เซอร์จะกระพริบ B=0 เคอร์เซอร์จะไม่กระพริบ

5. Cursor or Display Shift

RS	$R/\overline{W}$	DB_7	DB_6	DB_5	DB_4	DB_3	DB_2	DB_1	DB_0	
0	0	0	0	0	1	S/C	R/L	*	*	(no effect)

เป็นคำสั่งกำหนดให้เคอร์เซอร์หรือข้อมูลไปเกิดทางซ้ายหรือขวาโดยไม่ต้องใช้คำสั่งเขียนหรืออ่าน โดยที่

S/C	R/L	
0	0	ย้ายเคอร์เซอร์ไปทางซ้าย 1 ตำแหน่ง
0	1	ย้ายเคอร์เซอร์ไปทางขวา 1 ตำแหน่ง
1	0	ดันตัวอักษรไปทางซ้าย
1	1	ดันตัวอ้กษรไปทางขวา

6. Function Set

RS	R/$\overline{W}$	DB_7	DB_6	DB_5	DB_4	DB_3	DB_2	DB_1	DB_0	
0	0	0	0	1	DL	N	F	*	*	(no effect)

บิท DL : DL=0 ติดต่อแบบ 4 บิท DL=1 ติดต่อแบบ 8 บิท

บิท N : N=0 แสดงผล 1 บรรทัด N=1 แสดงผล 2 บรรทัดหรือมากกว่า

บิท F : F=0 Dot การแสดงผลแบบ 5x7 F=1 แสดงผลแบบ 5x10

7. Set CG RAM Address

RS	R/$\overline{W}$	DB_7	DB_6	DB_5	DB_4	DB_3	DB_2	DB_1	DB_0
0	0	0	1	A	A	A	A	A	A

ใน HD44780 มีหน่วยความจำสองชุดคือ Display Data RAM (DD RAM) 80x8บิท และ Character Generator RAM (CG RAM) 512 บิท และ7200 บิท คำสั่งนี้เซทแอดเดรส CG RAM ด้วย

8. Set DD RAM Address

RS	R/$\overline{W}$	DB_7	DB_6	DB_5	DB_4	DB_3	DB_2	DB_1	DB_0
0	0	1	A	A	A	A	A	A	A

เป็นค่าสั่งเซทค่าแอดเดรสใน DD RAM ในการเขียนหรืออ่านค่าจาก DD RAM (ส่วนแสดงผลหน้าจอ) โดยจำนวนแอดเดรสที่จะเกิดบนจอขึ้นกับการเซทค่า N ด้วย

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

ถ้า N=1 (2 บรรทัด) แอดเดรสจะอยู่ที่ 00H-27H (บรรทัดที่ 1)และ 40H-67H (บรรทัดที่ 2)

9. Read Busy Flag and Address

RS	R/$\overline{W}$	DB_7	DB_6	DB_5	DB_4	DB_3	DB_2	DB_1	DB_0
0	1	BF	A	A	A	A	A	A	A

เป็นคำสั่งอ่านค่า Busy Flag ซึ่งจะเป็นตัวบอกว่า HD44780 นี้อยู่ในขบวนการทำงานภายในหรืออยู่ในสภาพพร้อมจะรับข้อมูลโดย

BF=1 อยู่ในขบวนการทำงานภายในไม่พร้อมจะรับข้อมูลหรือคำสั่งได้

BF=0 พร้อมจะรับข้อมูลหรือคำสั่งได้ และนอกจากนี้ยังเป็นคำสั่งอ่านค่าข้อมูลแอดเดรสของ CG RAM หรือ DD RAM ด้วย

10. Write Data to CG or DD RAM

RS	$R/\overline{W}$	DB_7	DB_6	DB_5	DB_4	DB_3	DB_2	DB_1	DB_0
1	0	D	D	D	D	D	D	D	D

เป็นคำสั่งเขียนข้อมูลเข้า CG หรือ DD RAM โดยเมื่อเขียนข้อมูล และแอดเดรสจะเพิ่มหรือลดตามคำสั่งใน Entry Mode การที่จะรู้ว่าเป็นการเขียนข้อมูลของ CG RAM หรือ DD RAM ทำได้โดยเซทแอดเดรสของ CG RAM หรือ DD RAM ก่อนเขียนข้อมูล

11. Read Data from CG or DD RAM

RS	$R/\overline{W}$	DB_7	DB_6	DB_5	DB_4	DB_3	DB_2	DB_1	DB_0
1	1	D	D	D	D	D	D	D	D

เป็นคำสั่งอ่านข้อมูลจาก CG RAM หรือ DD RAM โดยควรเซทแอดเดรสก่อนเพื่อให้รู้ว่าข้อมูลที่อ่านได้นั้นเป็น DD RAM หรือ CG RAM

<u>ขาต่างๆในการต่อใช้งาน HD44780</u>

1. RS (Register Selection) เป็นขาเลือกรีจิสเตอร์ภายในซึ่งมี 2 ตัวคือ Instruction Register (IR) และ Data Register (DR) โดยถ้าเป็น 1 จะเลือก Data ถ้าเป็น 0 จะเลือก Instruction

2. $R/\overline{W}$ (Read/Write) เป็นตัวเลือกว่าจะเขียนหรืออ่านข้อมูลจากไอซี โดยอ่านข้อมูล=1, เขียนข้อมูล=0

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

RS	R/W	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.

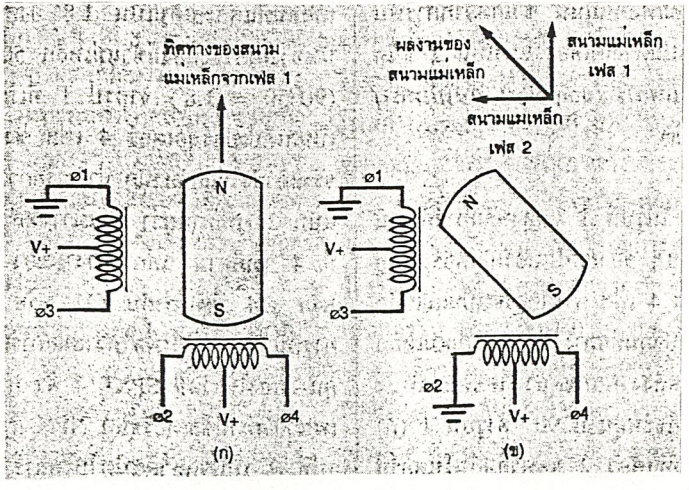
รูปที่ 1.10 แสดงความสัมพันธ์ระหว่าง RS, R/$\overline{W}$ และ E

3. E (Enable Signal) เป็นขากำหนดสภาพการเขียน/อ่านข้อมูล
4. DB_0-DB_7 เป็นขารับส่งข้อมูลจากไอซี
5. V_{DD} เป็นไฟเลี้ยงวงจร
6. V_{SS} เป็นขา GND
7. V_0 เป็นขาปรับแรงดันในการขับ LCD ให้มืด หรือสว่าง

Stepper Motor

ทฤษฎีการทำงาน

การทำงานของชิงโครนัสมอเตอร์คือ มีสนามแม่เหล็กหมุนเหมือนกันโดยชนิดของสเต็ปเปอร์มอเตอร์จะแบ่งตามสนามแม่เหล็กหมุนนี้ซึ่งเกิดจากการพันขดลวดบนตัวสเตเตอร์แบ่งเป็น2ชนิดคือยูนิโพลาร์(unipolar)กับไบโพลาร์(bipolar)ในรูปที่1.11แสดงหลักการทำงานแบบง่ายๆของสเต็ปเปอร์มอเตอร์แบบยูนิโพลาร์ 4เฟส ตัวโรเตอร์จะเป็นแม่เหล็กโดยจะเปลี่ยนทิศทางไปตามสนามแม่เหล็กการให้พลังงานแก่ขดลวดใดขดลวดหนึ่งโรเตอร์จะหมุนไป 90 ดังรูปที่ 1.11(ก) แต่ถ้าให้ทีเดียว 2ขดพร้อมกันโรเตอร์ก็จะหมุนไป 45 ดังรูปที่1.11(ข)ซึ่งแบบหลังจะสร้างแรงบิดได้ดีกว่าแบบแรก สเต็ปเปอร์มอเตอร์จะมีมุมของการเคลื่อนที่แต่ละสเต็ปเป็น 1.8 องศาดังนั้นที่ขั้วแม่เหล็ก 50 ขั้ว(90 /50 =1.8) จากรูป1.11ที่เรียกว่าเป็นสเต็ปเปอร์มอเตอร์ 4เฟสที่จริงแล้วไม่ถูกต้องนักน่าจะเรียกเป็นแบบ 2 เฟสมากกว่าถึงแม้ว่าจะมีขดลวด 4 ขดก็ตามแต่การทำงานของเฟส 3หรือเฟส 4 มีค่าเท่ากับเฟส 1 หรือเฟส 2 การที่มี 4 ขดก็เพื่อให้ง่ายต่อการควบคุมเพียงใช้สวิตช์ spst4ตัวหรือใช้เพาเวอร์ทรานซิสเตอร์ชนิดNPNส่วนในรูปที่1.12เป็นขดลวดชนิดไบโพลาร์ เมื่อขดลวดAและBในรูปมีกระแสไหลผ่านสเตเตอร์จะเกิดขั้วแม่เหล็กตามรูปเป็นผลให้โรเตอร์ที่มีขั้วแม่เหล็กต่างกันสเตเตอร์ถูกดูด ต่อมาเมื่อกระแสที่ไหลในขดลวด A เปลี่ยนทิศทางกลับจึงเป็นผลให้ขั้วแม่เหล็กที่แกน A เปลี่ยนขั้วจาก S เป็น N และเปลี่ยนจาก Nเป็น Sโรเตอร์จึงถูกผลักให้หมุนทวนเข็มไป 90 ลำดับการหมุนใน 1รอบเป็นสเต็ปดังนี้ AB→AB→AB→AB→AB มี 4 สเต็ปๆละ90



รูปที่ 1.11 เป็นชนิด4เฟส เมื่อϕ1ทำงานโรเตอร์จะเป็นดังรูป (ก) และเมื่อϕ1และ ϕ2ทำงานพร้อมกันโรเตอร์จะเป็นดังรูป (ข)

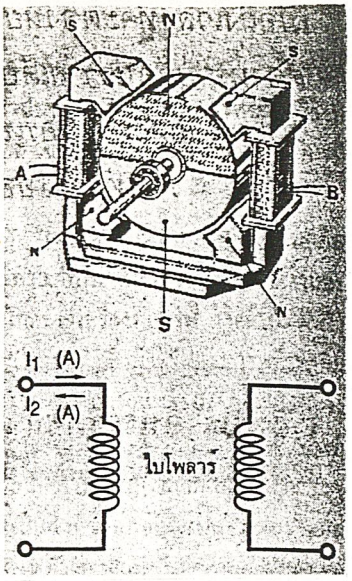
สเต็ปเปอร์มอเตอร์	ดีซีเซอร์โวมอเตอร์
มีการควบคุมที่ซับซ้อน ไม่ต้องมีการป้อนกลับ	การควบคุมง่าย จำเป็นต้องมีการป้อนกลับ(อาศัยตัวต้านทานปรับค่าได้, กำเนิดการนับรอบ)
กำลังงานเมื่อเทียบกับขนาดรูปร่างยังไม่เหมาะสม	กำลังเมื่อเทียบกับขนาดรูปร่างเหมาะสม
แข็งแรงสึกหรือต่ำ คุณสมบัติในการบล็อกกิ้งดี	การสึกหรอมาก เพราะใช้แปรงถ่าน การบล็อกกิ้งต้องอาศัยการเบรคที่พิเศษ

ตารางที่ 1.1 แสดงความแตกต่างของสเตปเปอร์มอเตอร์กับดีซีเซอร์โวมอเตอร์

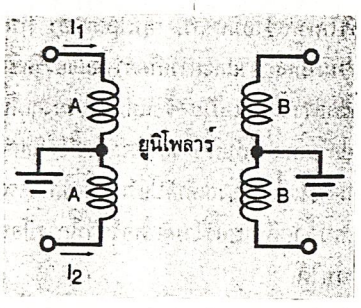
จะสังเกตได้ว่าเมื่อเวลากลับขั้วแม่เหล็กในแต่ละเฟสจะต้องมีการหยุดกระแสก่อนแล้วกระแสจึงค่อยเปลี่ยนทิศทางจึงสรุปเป็นสเต็ปได้คือ AB→B→AB→A→AB→B →AB→A→AB การทำงานเป็นแบบกึ่งสเต็ปนี้เป็นผลให้ค่าโมเมนต์มีค่าน้อยกว่าปกติเพราะมีช่วงเวลาที่กระแสไหลแค่เฟสเดียว ส่วนสเต็ปเปอร์มอเตอร์แบบยูนิโพลาร์ก็คล้ายๆกับไบโพลาร์โดยคิดเพียงขดเดียว ในแต่ละเฟสของยูนิโพลาร์จะมีแทปกลาง ซึ่งจะแบ่งเป็น 2ขดดังรูปที่1.13ดังนั้นเมื่อสนามแม่เหล็กเปลี่ยนแปลงกระแสจะไม่เปลี่ยนทิศทางการไหลเป็นที่แน่นอนว่าถ้าจำนวนขดของยูนิโพลาร์พันเหมือนแบบไบโพลาร์ แต่ยูนิโพลาร์มีแทปจึงเป็นผลให้แอมแปร์-เทิร์นซึ่งเป็นค่าฟลั๊กแม่เหล็กมีค่าน้อยกว่าไบโพลาร์เพราะฉนั้นสนามแม่เหล็กที่ได้ก็น้อยตามแรงบิดที่ขึ้นกับสนามแม่เหล็กก็น้อยกว่าด้วยเมื่อเทียบกับไบโพลาร์ขนาดเดียวกัน

ความต้องการในการให้มันมีการหมุนที่เที่ยงตรงและถูกต้องการหมุนในแต่ละรอบต้องมีสเต็ปมากขึ้น เราจึงต้องสร้างตัวโรเตอร์และสเตเตอร์ให้มีหลายขั้นโดยแยกขดลวดแต่ละเฟสออกจากกันซึ่งในเวลาการทำงานเฟสแต่ละเฟสต้องต่างเฟสเล็กน้อย

จำนวนสเต็ปที่มากที่สุดของมอเตอร์ถูกกำหนดโดยส่วนประกอบของโรเตอร์ที่เป็นแม่เหล็ก ซึ่งเกี่ยวข้องกับการเหนี่ยวนำแรงดันของขดลวดในสเตเตอร์นิยมใช้เหล็กอ่อนเป็นตัวโรเตอร์มีขั้วแม่เหล็กน้อยกว่าสเตเตอร์และเป็นแบบยูนิโพลาร์ดังแสดงในรูปที่ 1.13



รูปที่ 1.12 ไบโพลาร์สเต็ปเปอร์มอเตอร์แบบ 2 เฟส สนามแม่เหล็กจะเปลี่ยนเมื่อกลับทิศทางการไหลของกระแส



รูปที่ 1.13 ยูนิโพลาร์สเต็ปเปอร์มอเตอร์ การเปลี่ยนขั้วแม่เหล็กใช้การไหลของกระแสที่ต่างขดกัน กระแสจะไม่ไหลที่เดียวพร้อมกัน 2 ขด ในตัวสเตเตอร์เดียวกัน

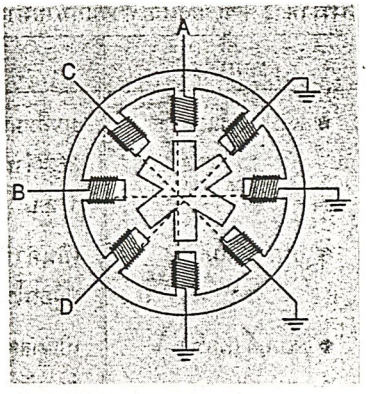
คำจำกัดความ

ก่อนที่จะทราบถึงหลักทางปฏิบัติของสเต็ปเปอร์มอเตอร์ควรจะทราบถึงลักษณะคุณสมบัติของตัวมอเตอร์ก่อนดังในตารางที่1.2จะบอกถึงความหมายของข้อมูลแต่ละอย่างของมอเตอร์แบ่งประเภทของความหมายได้ 2 ประเภทคือทางไฟฟ้าและทางกล ข้อมูลทางกลเป็นข้อมูลที่เราต้องการใช้ส่วนข้อมูลทางไฟฟ้าใช้สำหรับในการออกแบบวงจรอิเลคทรอนิกส์ควบคุม ตัวแปรที่มีความสำคัญที่จะต้องการทราบคือ pull-in rate (เป็นค่ามากสุดที่ยอมให้เกิดอัตราเร่งสเต็ป) ซึ่งจะมีความสัมพันธ์กับค่าโมเมนต์ความเฉื่อยของตัวโรเตอร์ ซึ่งในทางปฏิบัติแล้วค่าโมเมนต์ความเฉื่อยของตัวโรเตอร์ซึ่งในทางปฏิบัติแล้วค่าโม

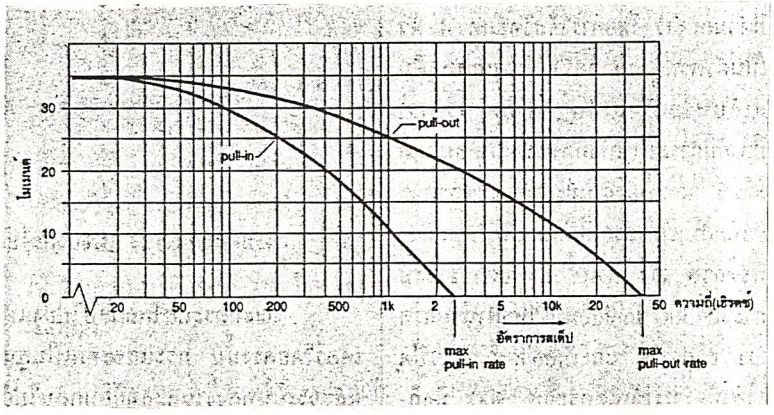
เมนต์ความเฉื่อยจะเพิ่มขึ้นได้ด้วยการถูกหมุนโดยตัวมอเตอร์แล้วผลที่ตามมาก็จะทำให้ pull-in rateลดลง ดังในรูปที่1.15เป็นกราฟลักษณะคุณสมบัติระหว่างโมเมนต์กับความถี่ จะเห็นได้ว่าเมื่อความถี่เพิ่มขึ้นค่าโมเมนต์จะลดลง ที่เป็นเช่นนี้ก็เพราะว่าเมื่อความถี่ที่เข้ามาสูงขึ้น จะทำให้ค่าอินดักแตนซ์ที่ขดลวดบนสเตเตอร์สูงขึ้นกระแสจะไหลได้น้อยลงและเป็นผลให้ค่าสนามแม่เหล็กน้อยลงด้วย นอกจากนี้กระแสที่ไหลในขดลวดสเตอร์ก็ไม่สามารถเปลี่ยนแปลงได้อย่างรวดเร็วด้วยซึ่งจากกราฟได้แสดงถึงค่าโมเมนต์ 2 โมเมนต์คือ กราฟ pull-in และ กราฟ pull-out กราฟ pull-in ควรจะใช้เมื่อขับมอเตอร์ด้วยความถี่คงที่ค่าของโมเมนต์ก็จะอยู่ที่ค่าหนึ่ง ส่วนกราฟ pull-outใช้กับการเร่งและหน่วงความเร็วที่ราบรื่นไม่กระตุก ซึ่งค่าโมเมนต์จะสูงกว่ากราฟ pull-in แต่วงจรควบคุมซับซ้อนกว่า

ตารางที่ 1.2 ชื่อเรียกและความหมายทางกลและทางไฟฟ้าของสเต็ปเปอร์มอเตอร์

ชื่อทางกล	ความหมาย	ชื่อทางไฟฟ้า	ความหมาย
● สเต็ปเปอร์แองเกิล	มุมที่หมุนไปใน 1 สเต็ป มีค่า 360/จำนวนสเต็ปในการหมุนไป 1 รอบ	● ยูนิโพลาร์กับไบโพลาร์	เป็นชนิดของการพันขดลวดบนตัวสเตเตอร์
● เบรกกิ้งโมเมนต์	เป็นค่าโมเมนต์มากสุดในการบล็อกโรเตอร์ไม่ให้หมุน	● ค่าความเหนี่ยวนำ (L)	เป็นตัวกำหนดขนาดของกระแสที่อัตราความเร็วสเต็ปสูงซึ่งสัมพันธ์กับฟลักซ์แม่เหล็ก
● โมเมนต์ (ทอร์ค)	เป็นผลคูณระหว่างระยะทางที่ตั้งฉากกับแรงที่มากระทำ	● ค่าความต้านทาน (R)	เป็นตัวจำกัดกระแสที่ขดลวดบนสเตเตอร์กับที่โรเตอร์
● pull-in rate	ความถี่ที่เริ่มสตาร์ท โดยที่ยังไม่มีแรงสูญเสีย	● กระแสสเตเตอร์มากสุด	ขึ้นอยู่กับขนาดของขดลวดที่พัน
● pull-out rate	อัตราของสเต็ปปิ้งเมื่อความเร่งคงที่แล้ว		
● โมเมนต์เฉื่อย (*I*)	เป็นการวัดแรงต้านของวัตถุต่อความเร่งเชิงมุม		



รูปที่ 1.14 สเต็ปเปอร์มอเตอร์ที่ใช้แกนเหล็กอ่อน ซึ่งเป็นสิ่งจำเป็นเมื่อใช้เป็นมอเตอร์ที่ความถี่ 50 Hz และขั้วแม่เหล็กบนโรเตอร์น้อยกว่าขั้วบนสเตเตอร์



รูปที่ 1.15 กราฟคุณสมบัติระหว่างโมเมนต์กับความถี่

บทที่ 2 วงจรที่ใช้งานและการเชื่อมต่อ

การเชื่อมต่อทางฮาร์ดแวร์

การเชื่อมต่อ EPROM

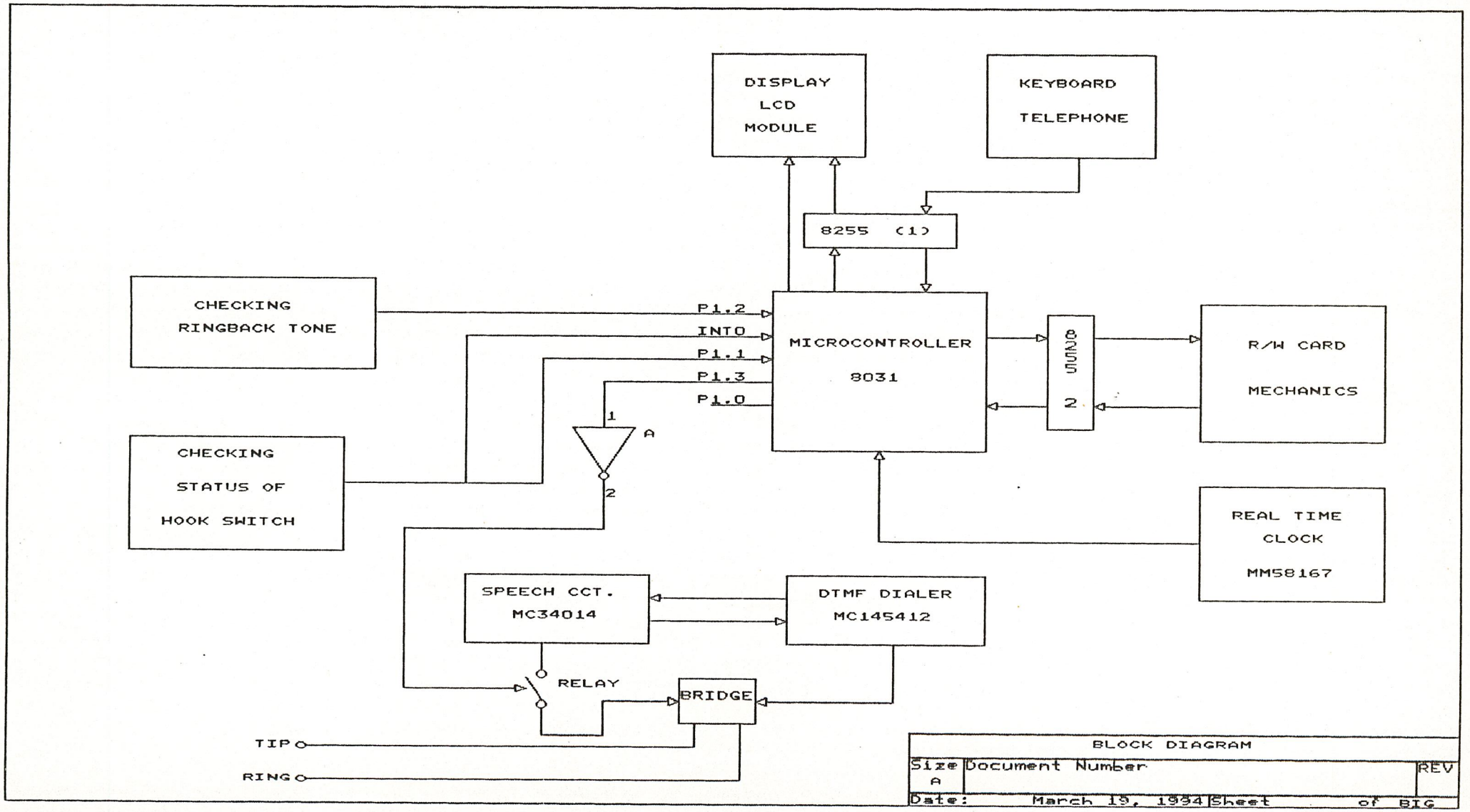
เนื่องจาก EPROM 2732 ถูกใช้เป็นหน่วยความจำสำหรับโปรแกรม ดังนั้น 8031 จึงต้องมีการเข้าถึงหน่วยความจำโดยการอ้างแอดเดรสไบท์ต่ำและ data ผ่าน Port 0 ซึ่งใช้การ Multiplex ส่วนแอดเดรสไบท์สูงจะอ้างโดยผ่าน Port 2 ดังนั้นจึงต้องใช้ 74LS373 เพื่อทำการ Latch แอดเดรสไบท์ต่ำแล้วนำเอาท์พุทของ74LS373 มาต่อเข้ากับขาแอดเดรสของ EPROM 2732 ส่วนขา data ของ 2732 จะต่อมาจาก Port 0 ของ 8031 โดยตรง มีการใช้ขาควบคุมการอ่านคำสั่ง (ขาPSEN) เมื่อต้องการอ่านคำสั่ง (Fetch Instruction) โดยต่อกับขา$\overline{OE}$ ของ 2732 และขา $\overline{CE}$ ของ2732 จะต่อกับขา 15 ของ 74LS138 ซึ่งทำการถอดรหัสจากการเข้ารหัส 4 บิทบนของขาแอดเดรสที่ออกทาง Port 2 ของ 8031

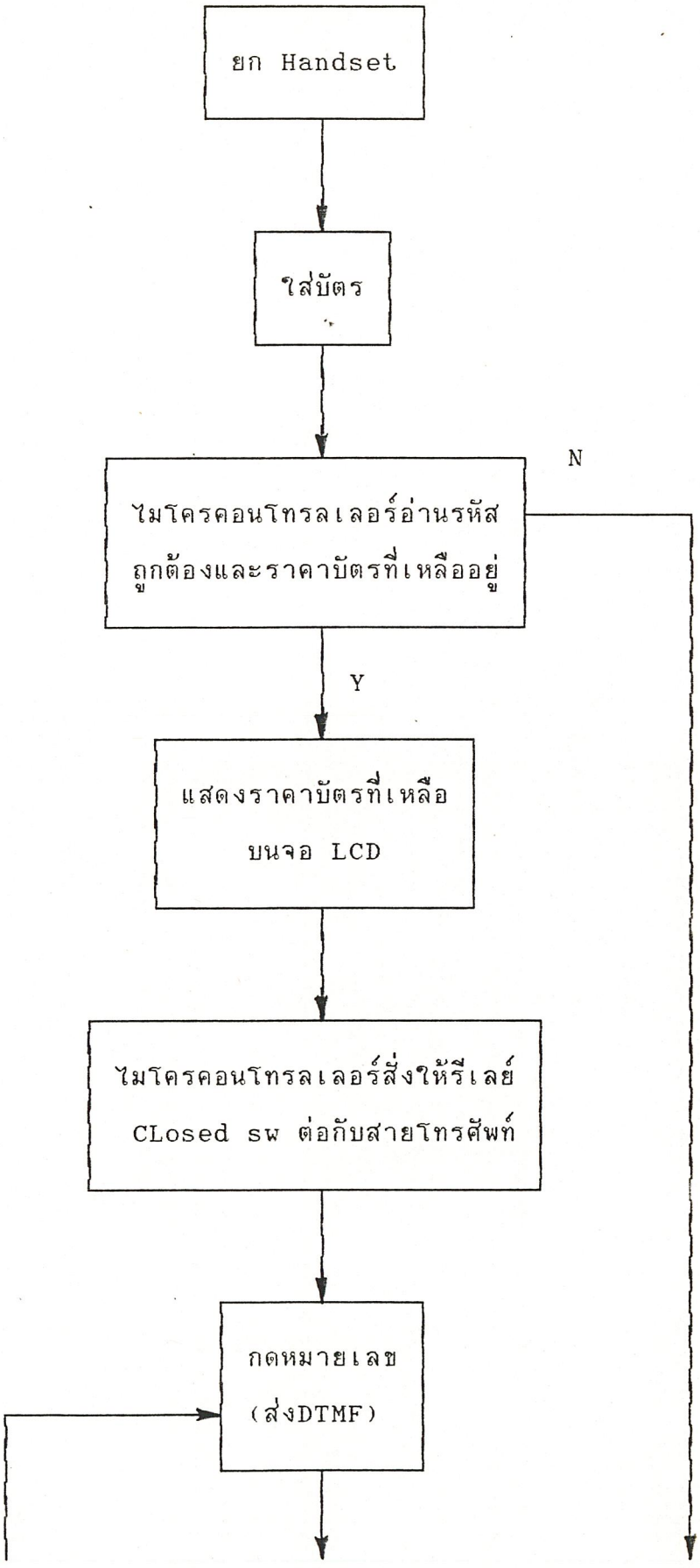
การเชื่อมต่อ RAM

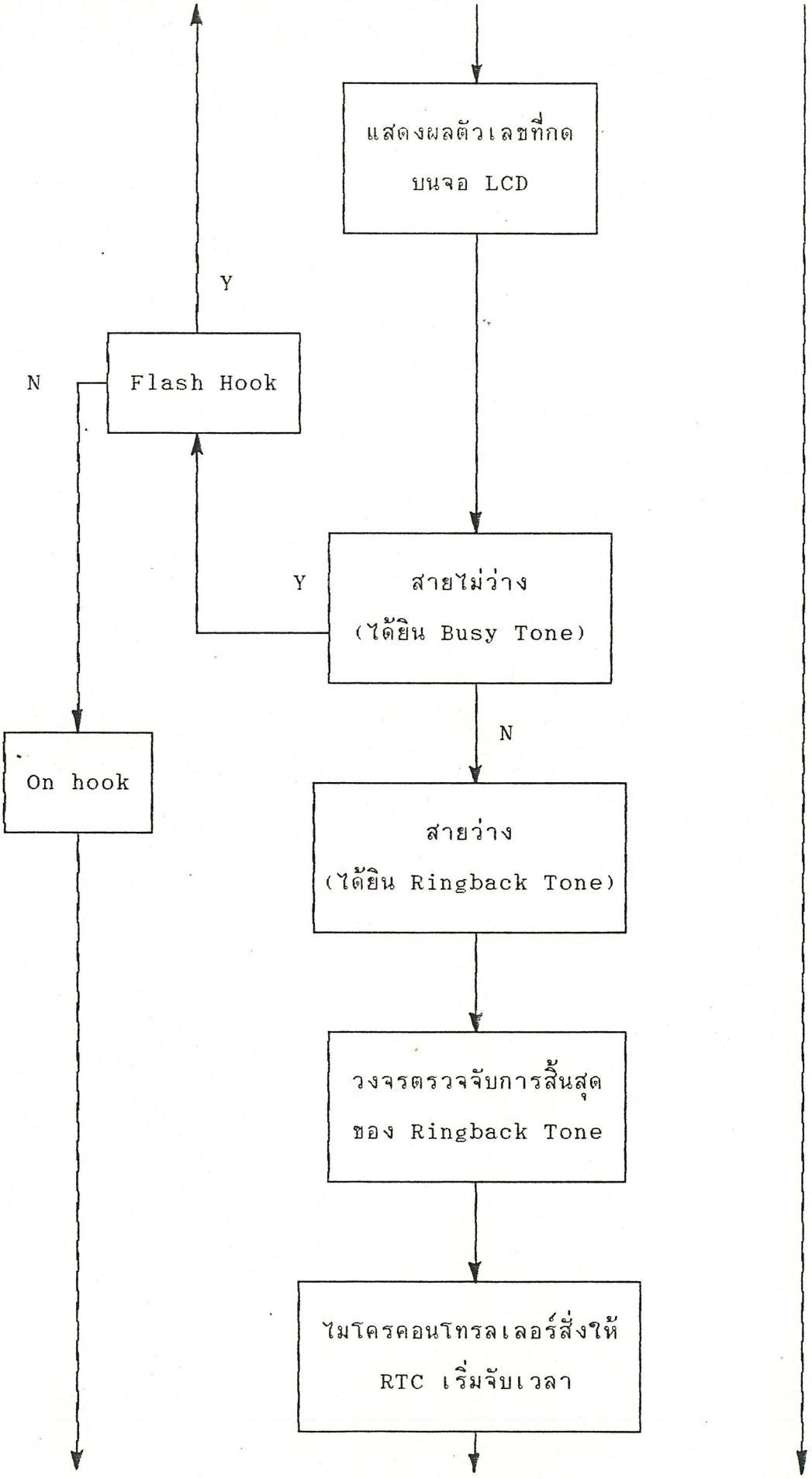
ใน 8031 มีลักษณะการเข้าถึงหน่วยความจำโดยการอ้างแอดเดรสไบท์ต่ำและdata ผ่านPort 0 ซึ่งใช้การ Multiplex ส่วนแอดเดรสไบท์สูงจะอ้างโดยผ่าน Port 2 ดังนั้นในการออกแบบต้องใช้74LS373 เพื่อทำการ Latch แอดเดรสไบท์ต่ำ แล้วนำเอาท์พุทของ 74LS373 มาต่อเข้ากับขาแอดเดรสของ RAM 6264 ส่วนขา data ของ RAM จะต่อมาจาก Port 0 ของ 8031โดยตรง มีการใช้ขาควบคุมการเขียนและอ่าน(ขา16 ของ8031)ต่อไปยังขา $\overline{WE}$(ขา27) ของ RAM 6264,ขา $\overline{RD}$ ของ 8031 ต่อกับขา $\overline{OE}$ ของ RAM 6264,ขา14 ของ74LS138 ซึ่งได้จากการเข้ารหัส 4 บิทบนของขาแอดเดรสที่ออกทาง Port 2 ของ 8031 ต่อเข้ากับขา $\overline{CS_1}$ ของ6264 และขา CS_2 ต่อกับ V_{cc}

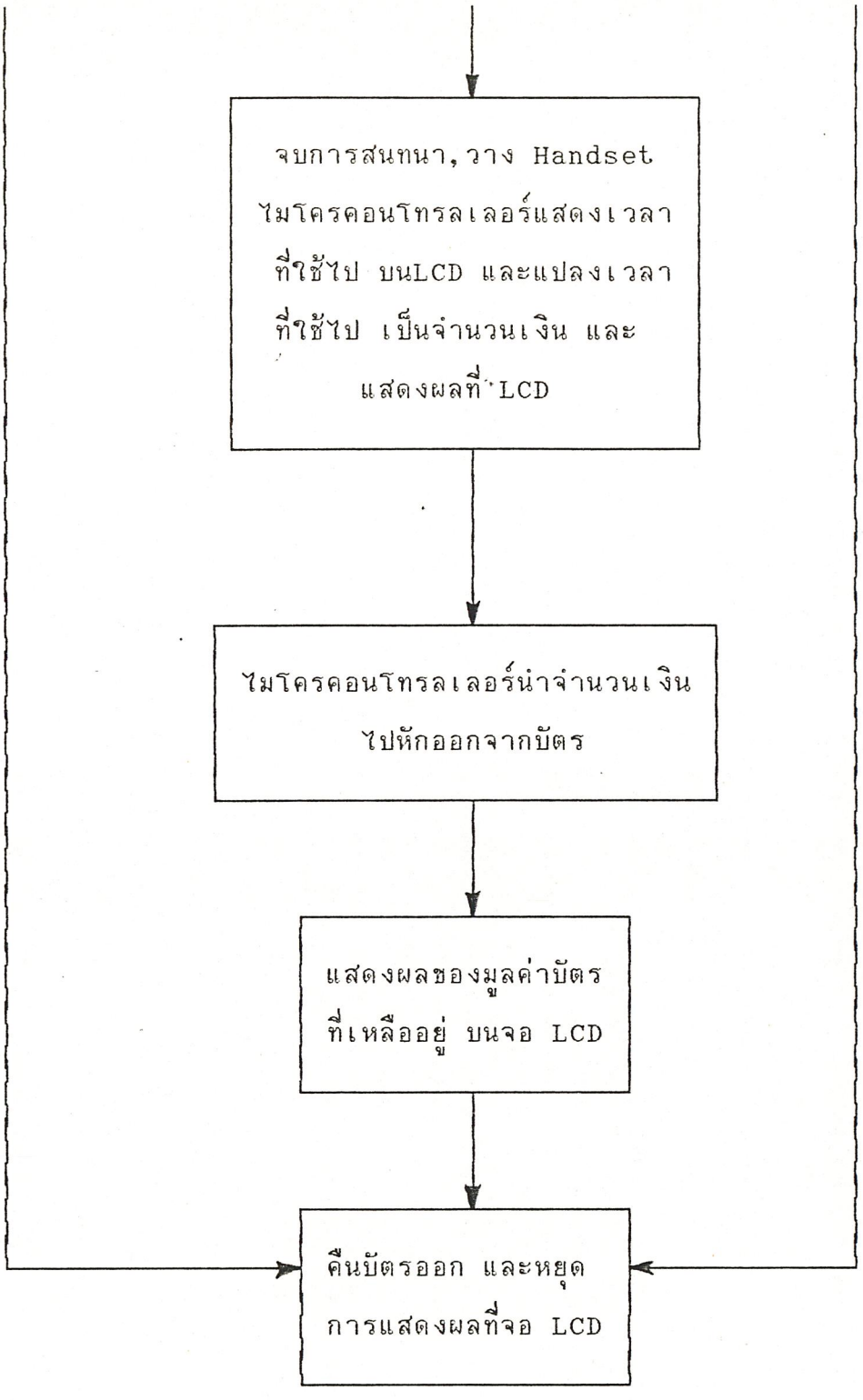
การเชื่อมต่อชิพ RTC MM58167

ในการใช้งานของชิพ RTC จะต่อขาแอดเดรสของ RTC กับขา แอดเดรสที่ถูกแลทช์จาก 8031 คือต่อกับขา Q_0-Q_4 ของ 74LS373 และขา data ของ RTC จะต่อกับพอร์ท 0 ของ 8031ได้โดยตรง ส่วนขา $\overline{RD}$ และ $\overline{WR}$ จะต่อกับขา $\overline{RD}$ และ $\overline{WR}$ ของ 8031 ขา $\overline{CS}$ จะต่อเข้ากับขา 13 ของ 74LS138 ดังรูป ซึ่งมาจากการถอดรหัสขา พอร์ท 2 ของ 8031 อีกครั้งหนึ่ง


DISPLAY
LCD
MODULE
KEYBOARD
TELEPHONE
8255 (1)
CHECKING
RINGBACK TONE
P1.2
INTO
P1.1
P1.3
P1.0
MICROCONTROLLER
8031
ULN2003
R/W CARD
MECHANICS
1
A
2
CHECKING
STATUS OF
HOOK SWITCH
REAL TIME
CLOCK
MM58167
SPEECH CCT.
MC34014
DTMF DIALER
MC145412
RELAY
BRIDGE
TIP
RING
BLOCK DIAGRAM
Size
A
Document Number
REV
Date:
March 19, 1994
Sheet
of
BIG

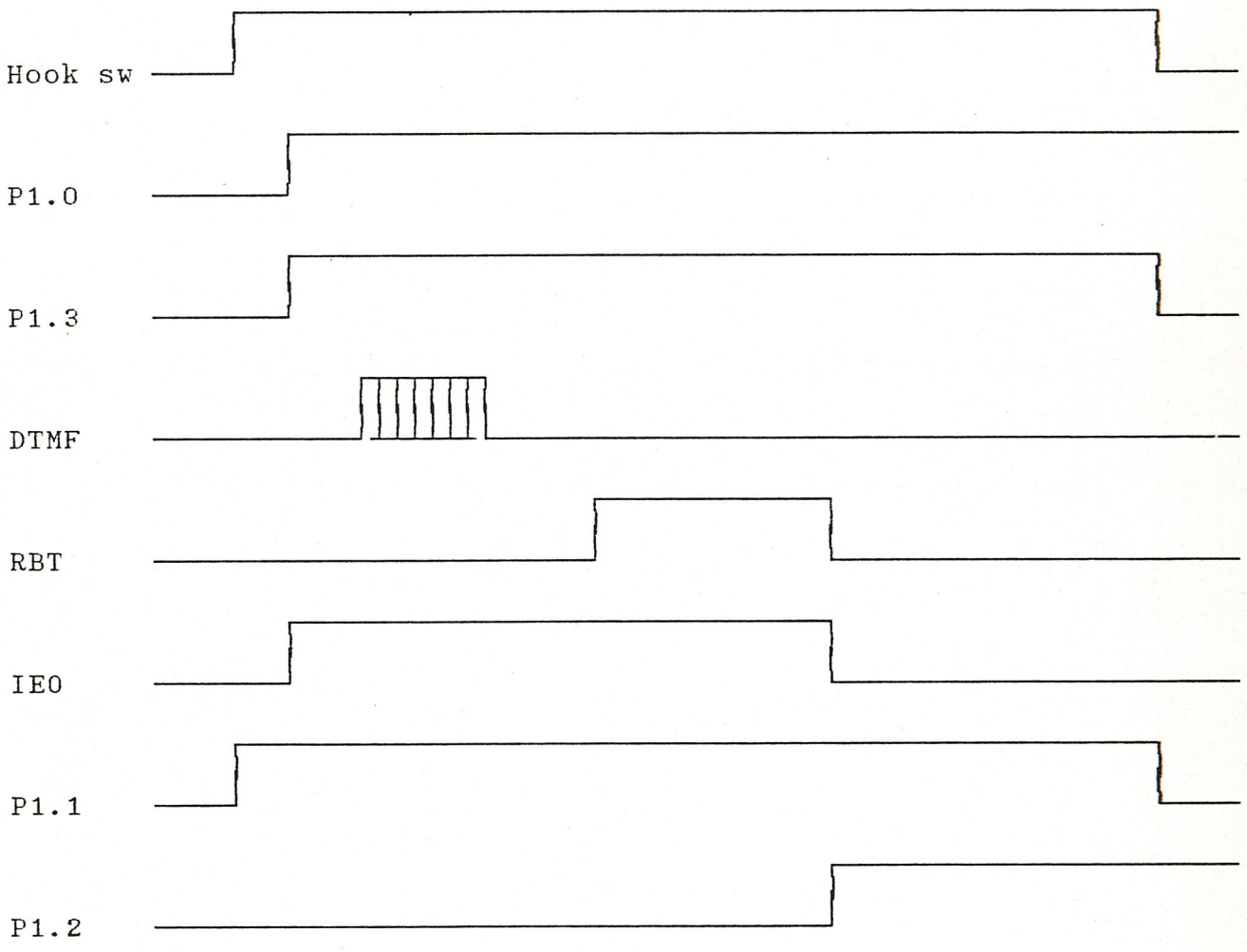

ยก Handset
ใส่บัตร
ไมโครคอนโทรลเลอร์อ่านรหัส
ถูกต้องและราคาบัตรที่เหลืออยู่
N
Y
แสดงราคาบัตรที่เหลือ
บนจอ LCD
ไมโครคอนโทรลเลอร์สั่งให้รีเลย์
CLosed sw ต่อกับสายโทรศัพท์
กดหมายเลข
(ส่งDTMF)


แสดงผลตัวเลขที่กด
บนจอ LCD
Y
Flash Hook
N
Y
สายไม่ว่าง
(ได้ยิน Busy Tone)
N
On hook
สายว่าง
(ได้ยิน Ringback Tone)
วงจรตรวจจับการสิ้นสุด
ของ Ringback Tone
ไมโครคอนโทรลเลอร์สั่งให้
RTC เริ่มจับเวลา

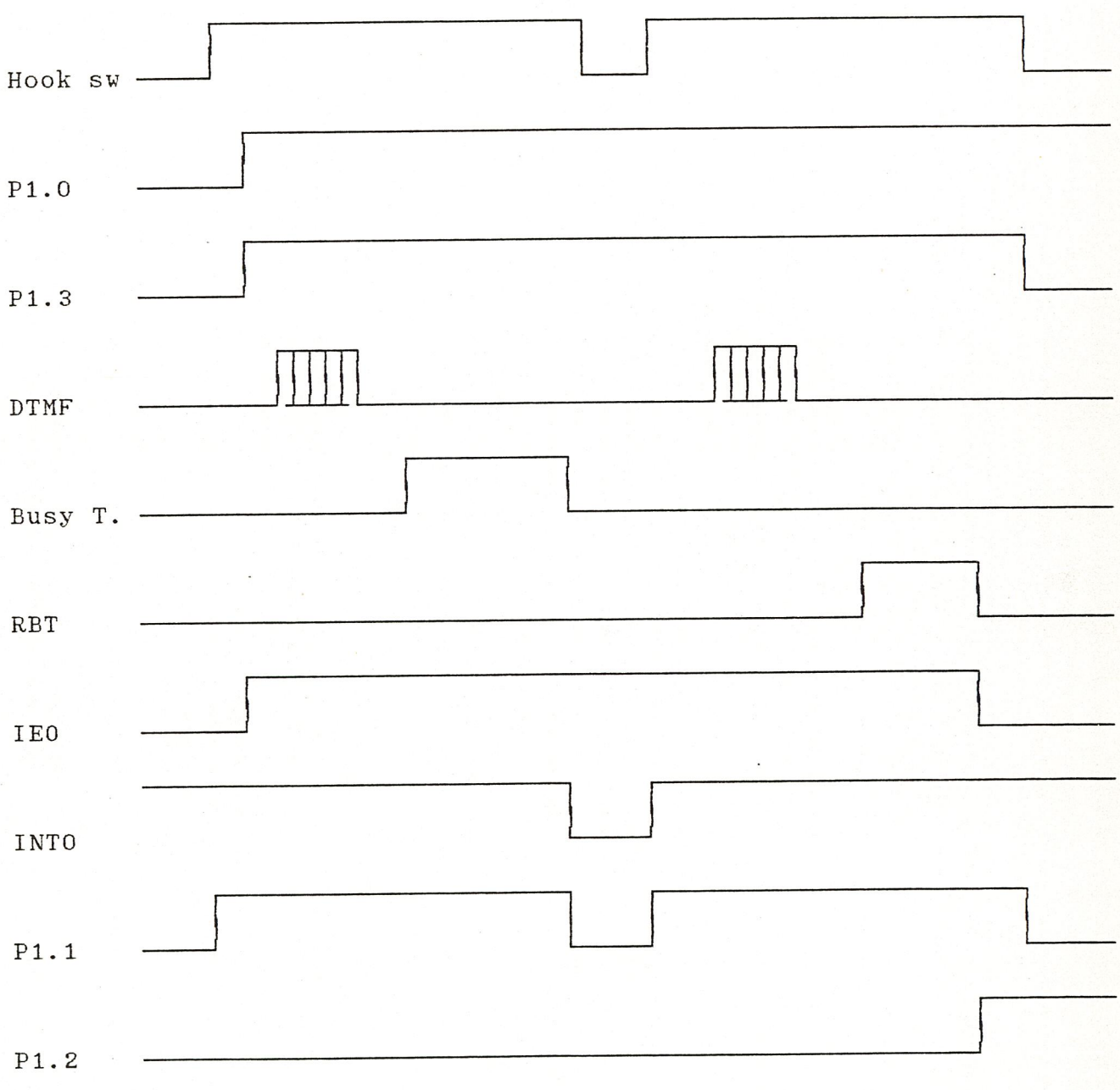

จบการสนทนา, วาง Handset
ไมโครคอนโทรลเลอร์แสดงเวลา
ที่ใช้ไป บนLCD และแปลงเวลา
ที่ใช้ไป เป็นจำนวนเงิน และ
แสดงผลที่ LCD
ไมโครคอนโทรลเลอร์นำจำนวนเงิน
ไปหักออกจากบัตร
แสดงผลของมูลค่าบัตร
ที่เหลืออยู่ บนจอ LCD
คืนบัตรออก และหยุด
การแสดงผลที่จอ LCD

กรรมวิธีการเรียก

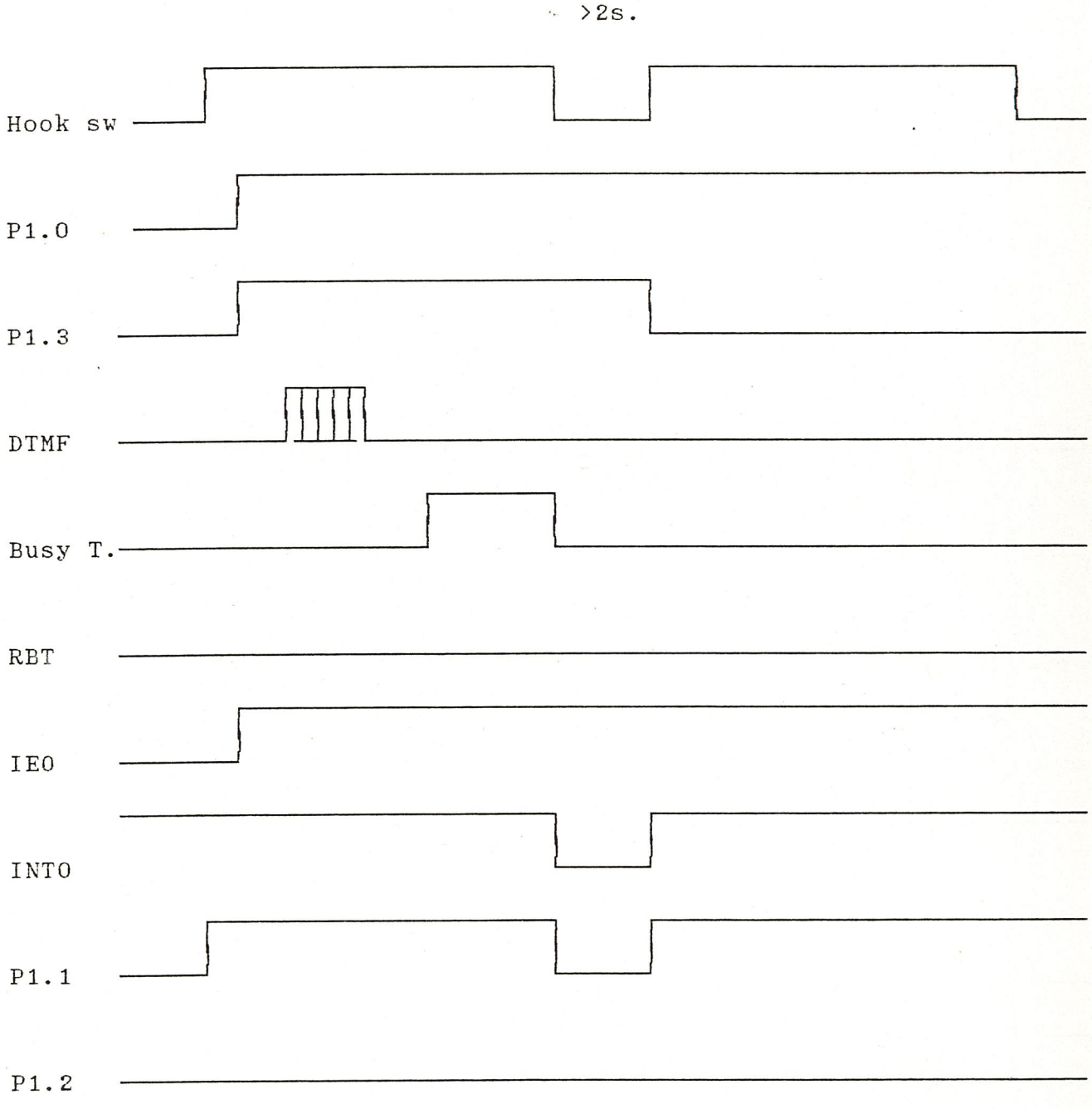
กรณีที่ 1 ยกหู→ใส่บัตร→กดDTMF→สายว่าง→สนทนา→On-Hook→คืนบัตร



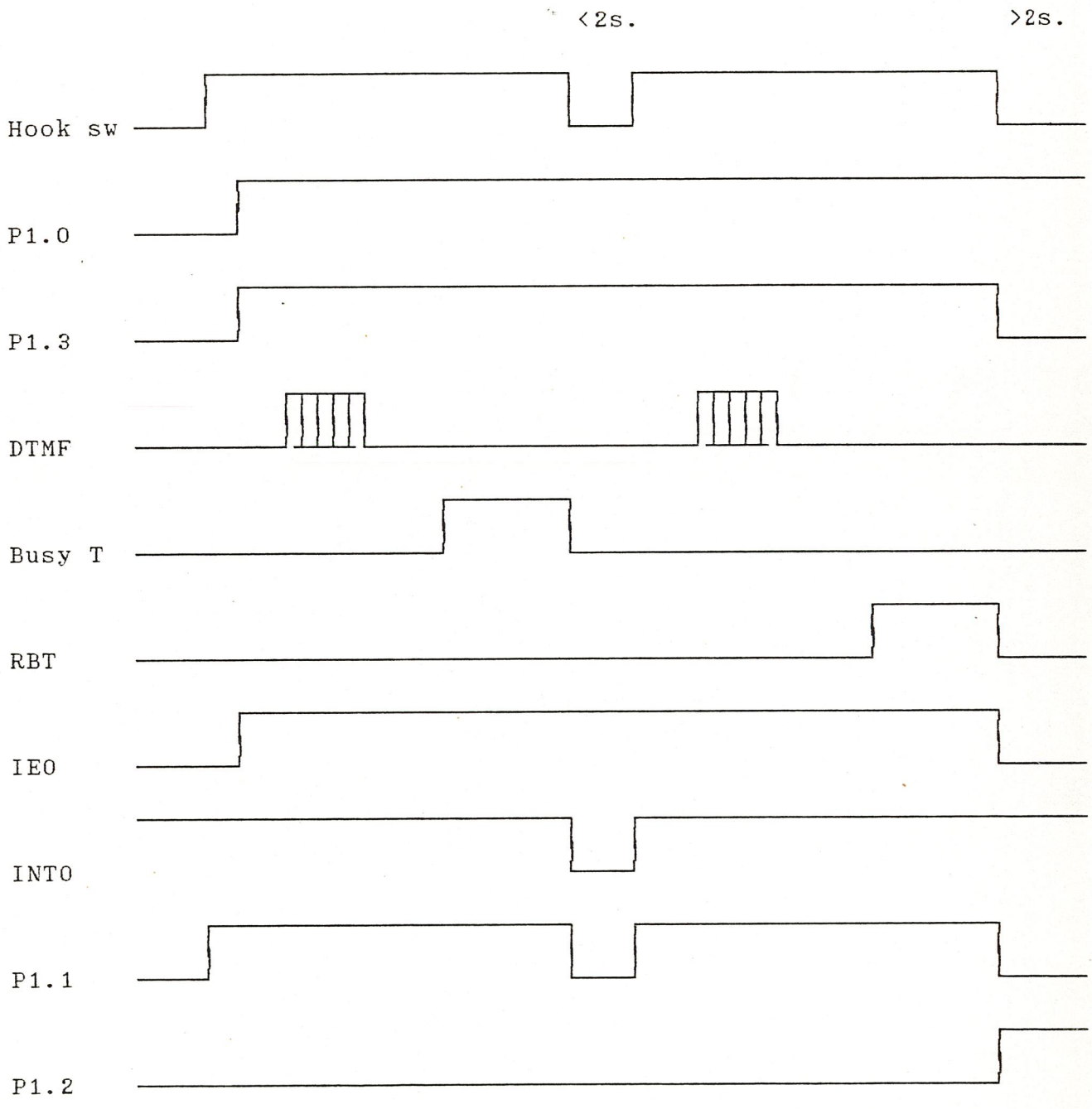
กรณีที่ 2 ยกหู→ใส่บัตร→กดDTMF→สายไม่ว่าง→Flash-Hook→กดDTMF
→สายว่าง→สนทนา→On-Hook→คืนบัตร

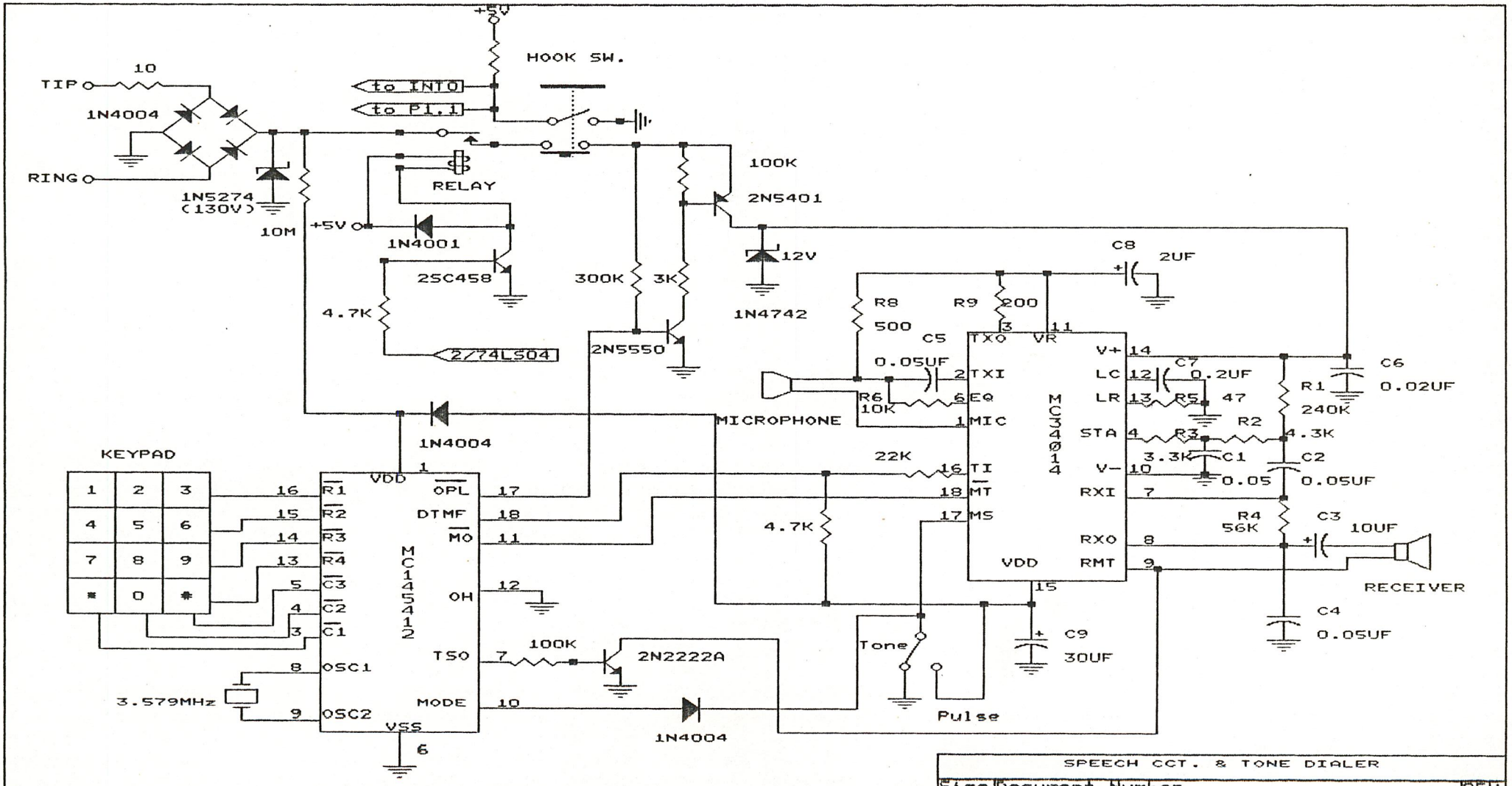


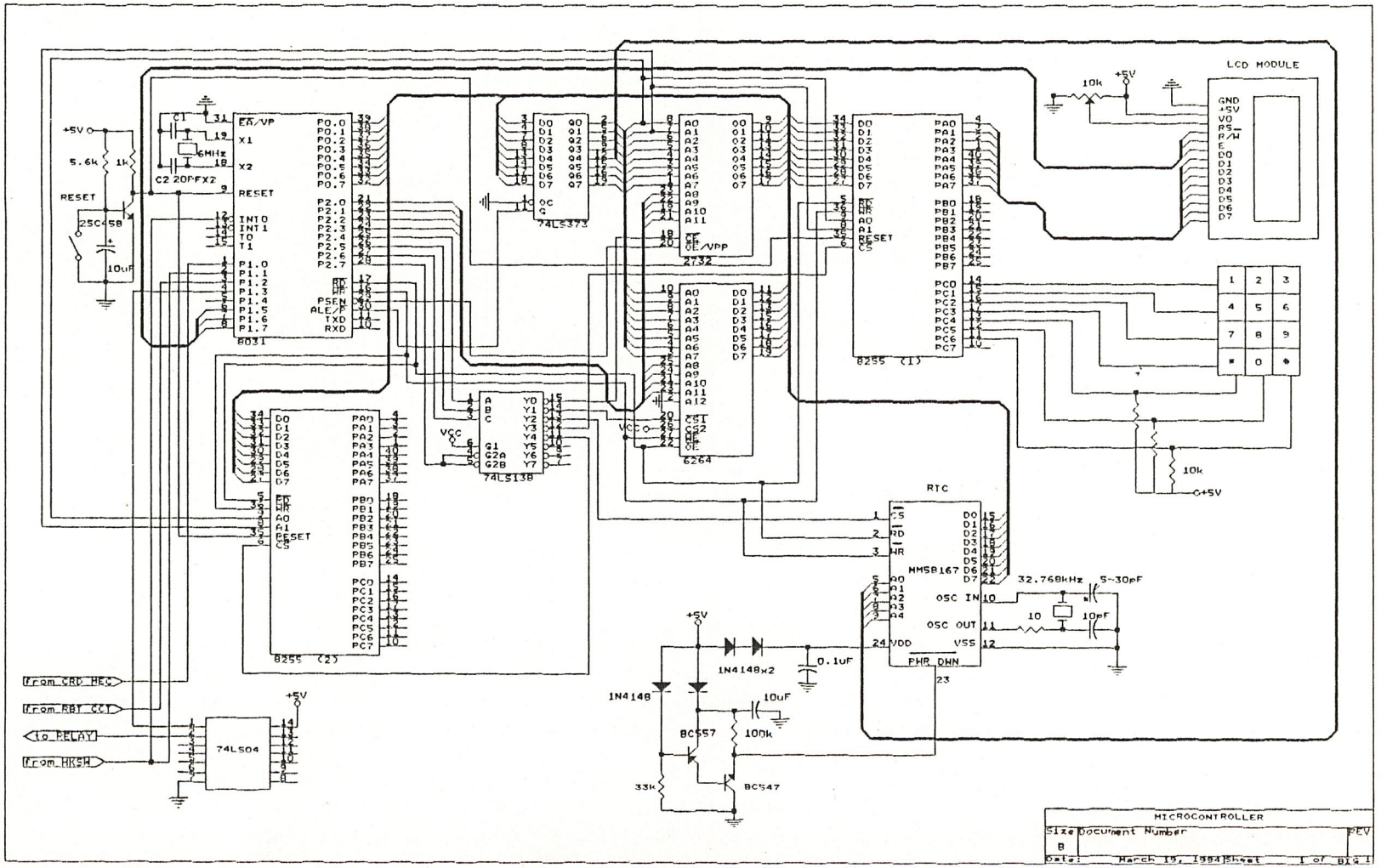
กรณีที่ 3 ยกหู→ใส่บัตร→กดDTMF→สายไม่ว่าง→On-Hook→คืนบัตร
(ยังไม่ได้สนทนา)

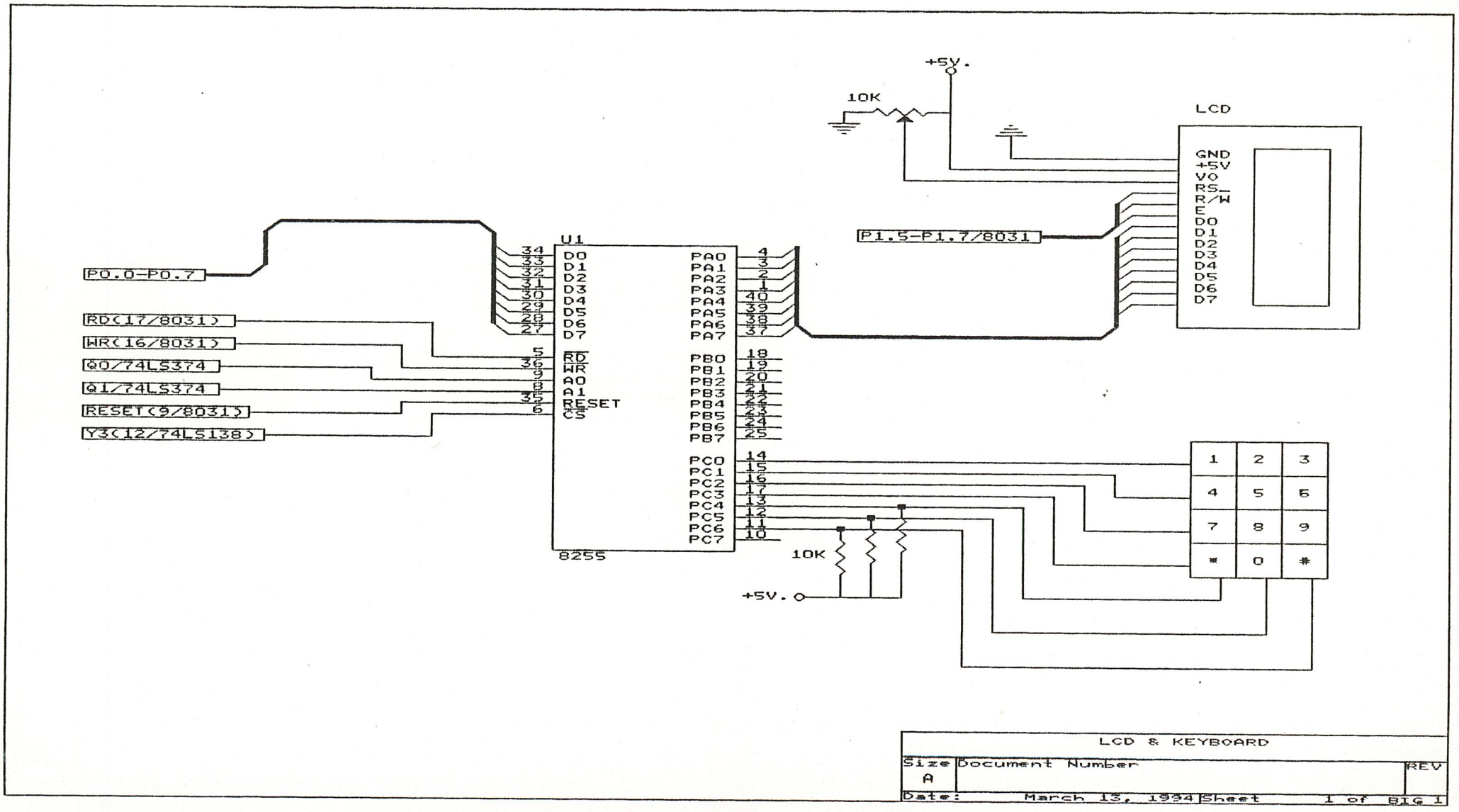


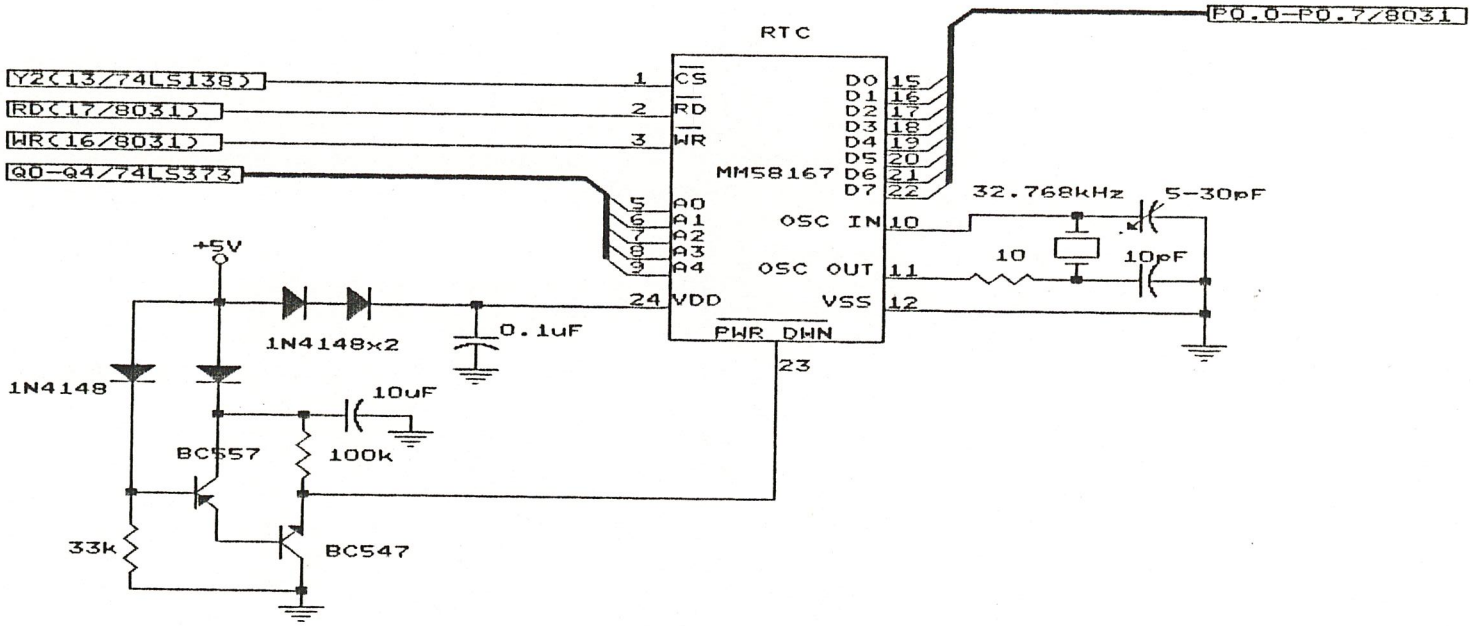
กรณีที่ 4 ยกหู→ใส่บัตร→กดDTMF→สายไม่ว่าง→Flash-Hook→กดDTMF
→สายว่าง→On-Hook→คืนบัตร (ยังไม่ได้สนทนา)



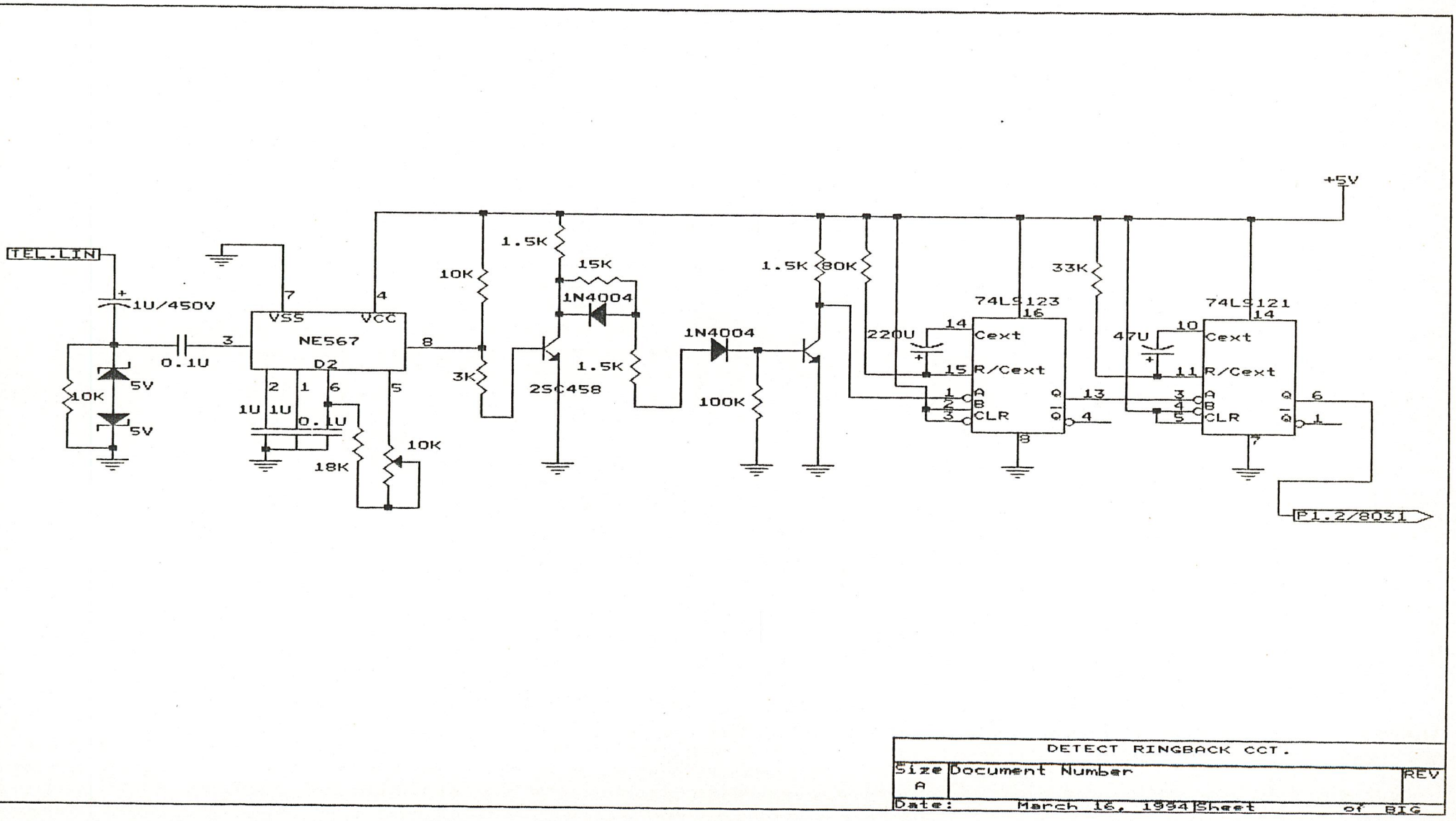

+5V
HOOK SW.
TIP
10
1N4004
RING
to INT0
to P1.1
1N5274
(130V)
10M
RELAY
+5V
1N4001
2SC458
4.7K
2/74LS04
300K
3K
2N5550
100K
2N5401
12V
1N4742
C8
2UF
R8
500
R9
200
C5
0.05UF
MICROPHONE
R6
10K
1N4004
KEYPAD
1 2 3
4 5 6
7 8 9
* 0 #
VDD
MC14412
R1
R2
R3
R4
C3
C2
C1
OSC1
OSC2
VSS
OPL
DTMF
MO
OH
TSO
MODE
3.579MHz
100K
2N2222A
1N4004
22K
4.7K
MC34014
TXO
VR
TXI
EQ
MIC
TI
MT
MS
VDD
V+
LC
LR
STA
V-
RXI
RXO
RMT
C7
0.2UF
R5
47
R2
R3
3.3K
C1
0.05
R1
240K
4.3K
C2
0.05UF
C6
0.02UF
R4
56K
C3
10UF
RECEIVER
C4
0.05UF
Tone
Pulse
C9
30UF
SPEECH CCT. & TONE DIALER
Size
A
Document Number
REV
Date:
March 13, 1994
Sheet
of
BIG


LCD MODULE
GND
+5V
VO
RS
R/W
E
DB
D1
D2
D3
D4
D5
D6
D7
10k
+5V
8031
EA/VP
X1
X2
RESET
INT0
INT1
T0
T1
P1.0
P1.1
P1.2
P1.3
P1.4
P1.5
P1.6
P1.7
P2.0
P2.1
P2.2
P2.3
P2.4
P2.5
P2.6
P2.7
PSEN
ALE/P
TXD
RXD
6MHz
C2 20PFX2
5.6k
1k
25C458
10uF
74LS373
2732
6264
8255 (1)
8255 (2)
74LS138
74LS04
RTC
MM58167
OSC IN
OSC OUT
VDD
VSS
PWR DWN
32.768kHz
5-30pF
10
10pF
0.1uF
IN4148x2
IN4148
BC557
BC547
100k
33k
10uF
10k
C+5V
from CRD REQ
from RBT CCT
to RELAY
from HKSW
MICROCONTROLLER
Size
Document Number
REV
Date: March 19, 1984
Sheet 1 of 1


+5V.
10K
LCD
GND
+5V
VO
RS
R/W
E
DO
D1
D2
D3
D4
D5
D6
D7
P1.5-P1.7/8031
U1
P0.0-P0.7
RD(17/8031)
WR(16/8031)
Q0/74LS374
Q1/74LS374
RESET(9/8031)
Y3(12/74LS138)
D0
D1
D2
D3
D4
D5
D6
D7
RD
WR
A0
A1
RESET
CS
PA0
PA1
PA2
PA3
PA4
PA5
PA6
PA7
PB0
PB1
PB2
PB3
PB4
PB5
PB6
PB7
PC0
PC1
PC2
PC3
PC4
PC5
PC6
PC7
8255
10K
+5V.
1
2
3
4
5
6
7
8
9
*
0
#
LCD & KEYBOARD
Size A
Document Number
REV
Date: March 13, 1994
Sheet 1 of BIG 1



RTC		
Size A	Document Number	REV
Date: March 13, 1994	Sheet of BIG	


TEL.LIN
1U/450V
0.1U
10K
5V
5V
NE567
VSS
VCC
D2
1U
1U
0.1U
18K
10K
3K
10K
1.5K
15K
1N4004
2SC458
1.5K
1N4004
100K
1.5K
80K
220U
74LS123
Cext
R/Cext
CLR
33K
47U
74LS121
Cext
R/Cext
CLR
+5V
P1.2/8031
DETECT RINGBACK CCT.
Size
A
Document Number
REV
Date:
March 16, 1994
Sheet
of
BIG

ในการสร้างความถี่ของวงจรฐานเวลาจะเลือกใช้ X-tal 32.768 KHz จึงต้องเลือกใช้ความต้านทานที่ขา OSC_{out} เท่ากับ 10 Ω และ C 10 pF ส่วนขา OSC_{in} จะมีค่า C ซึ่งแปรผันค่าได้ เพื่อให้เวลาผิดพลาดน้อยที่สุด

ในส่วนจ่ายไฟเลี้ยงจะใช้แหล่งจ่ายไฟจาก power supply

การเชื่อมต่อกับ LCD Module

ใช้ LCD Module รุ่น HDM-16416H ซึ่งจะมีขนาด 16 ตัวอักษร 4 บรรทัด โดยควบคุมผ่านพอร์ท 8255 1 พอร์ท และ พอร์ท P1.5-P1.7 อีก 1 พอร์ท พอร์ท 8255 ใช้เป็นพอร์ท data ซึ่งต่อมาจากพอร์ท PA ที่มีแอดเดรส 3000H ส่วนอีกพอร์ทเป็นพอร์ท Control ที่ต่อมาจากพอร์ท P1.5-P1.7 โดยมีขา RS, R/$\overline{W}$ และE ต่อเข้ากับขา P1.5, P1.6 และP1.7 ตามลำดับ

วงจรตรวจจับสัญญาณ Ringback Tone

วงจรตรวจจับสัญญาณเรียกกลับจะใช้ในการตรวจสอบสัญญาณเรียกกลับ เมื่อมีสัญญาณเรียกกลับแสดงว่าการติดต่อสำเร็จ แต่ผู้รับปลายทางยังไม่ได้ยก Handset ซึ่งสถานะนี้สัญญาณที่ออกจากขา Output Check จะเป็น 0 จนกระทั่งมีการยก Handset เอาท์พุทจะเป็น 1 ชั่วระยะเวลาหนึ่ง ซึ่งจะใช้สถานะนี้ร่วมกับ Status Flag จากโปรแกรมหลักในการตัดสินสถานะในขณะเวลานั้นๆว่าปลายทางยกหูขึ้นจริงหรือไม่ เมื่อตรวจสอบแล้วพบว่าปลายทางยกหูขึ้นจริงจึงเริ่มจับเวลาในการคิดค่าบริการจนกระทั่งเลิกสนทนาก็จะมีการตรวจสอบ Flag อีกครั้ง เพื่อให้ทราบว่าเลิกสนทนาแล้วจึงหยุดจับเวลา

วงจรนี้ใช้ไอซี PLL เบอร์ NE 567 จะมีลอจิก 1 ในสถานะปกติและจะตกเป็น 0 (Pulse ขอบขาลง) เมื่อมีความถี่ 400 Hz เข้ามาที่อินพุท เอาท์พุทของไอซีจะส่งผ่านทรานซิสเตอร์สองตัวมีหน้าที่สวิชชิ่งก่อนส่งผ่านไปให้ไอซี 74LS123 (Retriggable Monostable) โดยเซทให้ทริกที่ขอบขาลง ซึ่งเอาท์พุทของ 74LS123 จะเป็น 1 ตลอดจนกว่าสัญญาณเรียกกลับจะหยุดไป เมื่อสัญญาณเรียกกลับหมดไปจะมีพัลส์ขอบขาลงไปกระตุ้นให้ 74LS123 ทำงาน มีเอาท์พุทลอจิก 1 ชั่วขณะเวลาตามที่ได้ตั้งไว้

จากรูปเนื่องจาก Ringback Tone เป็นสัญญาณที่มีความถี่ 400 Hz ดัง 1 วินาที หยุด 4 วินาที เราจึงต้องเซทเวลาของไอซี 74LS123 ให้เป็นพัลส์ที่เป็นบวกเป็นเวลาตามสูตร $t_w = 0.29\ R_T C_{ext}(1+0.7/RT)$ โดยที่ R_T ในวงจรมีค่า 80 KΩ

และ C_{ext} มีค่า 220 μF ทำให้ได้ค่า t_w = 5.184 วินาที ในขณะที่มีพัลส์จาก NE 567 ที่มีระยะห่างระหว่างพัลส์น้อยกว่า t_w เอาท์พุทเป็น 1 ต่อเนื่องกันไปตลอดช่วงที่มีสัญญาณ จนกว่าสัญญาณเรียกกลับจะหายไป จะเกิดพัลส์เอาท์พุทที่ 74LS121 เป็นเวลา 0.7 $R_T C_{ext}$ ซึ่งเราเลือก R_T ให้มีค่าเท่ากับ 33 KΩ C_{ext} ให้มีค่าเท่ากับ 47 μF ทำให้ได้พัลส์บวกยาวประมาณ 1.085 วินาที ซึ่งนานพอที่จะตรวจสอบสถานะได้ด้วยซอฟแวร์ได้ โดยการต่อขา 13 ของ 74LS121 ไปเข้ากับขา P1.2 (ขา3) ของ 8031 เพื่อนำไปตรวจสอบสถานะ

<u>การเชื่อมต่อกันระหว่าง Speech cct.กับ Tone Dialerและไมโครคอนโทรลเลอร์</u>

การเชื่อมต่อกันระหว่างวงจรส่งรหัสหมายเลข (Pulse/Tone Dialer) กับวงจรควบคุมเสียงพูด (Speech circuit) ทำโดยการต่อขา 17 ของ MC34014 กับขา 10 ของ MC145412 ซึ่งขา MS นี้จะเป็นการเลือกโหมดให้อยู่ในโหมดของ DTMF Dialing โดยการต่อขา MS กับ V_{ss} ส่วนสัญญาณควบคุมการตัดเสียง (Mute) $\overline{MO}$ (ขา11) ของ MC145412 จะต่อเข้ากับขา $\overline{MT}$ (ขา18) ของ MC34014 ได้โดยตรงเลย, ขา DTMF ของ MC145412 จะต่อผ่านความต้านทานมาเข้าขา TI ของ MC34014 และขา OH ของ MC145412 จะต่อกับ V_{ss} เพื่อให้เป็นการอินาเบิ้ลไอซี MC145412 สำหรับรีเลย์ในวงจร จะใช้ควบคุมโดยไมโครคอนโทรลเลอร์ 8031 เพื่อใช้ในการตัดหรือต่อวงจรกับสายโทรศัพท์ โดยต่อกับขา P1.3 ของ 8031 ผ่านความต้านทานและทรานซิสเตอร์ 2SC458 เพื่อขับรีเลย์ ส่วนสวิชท์ในวงจรนี้ใช้เป็น Hook Switch ซึ่งจะมีส่วนที่เชื่อมต่อกับ 8031 เพื่อเป็นการตรวจสอบสถานะการยกหู หรือการวางหู โดยต่อกับขา P1.1 ของ 8031 และต่อกับขา INT0 ของ 8031

การทำงานของเครื่องอ่าน-เขียนบัตรแม่เหล็ก

(1) เมื่อเสียบบัตรแม่เหล็กเข้าที่ช่องเสียบ ตัวตรวจจับที่ทางเข้าช่องเสียบจะทำงานทำให้มอเตอร์เดินหน้า(หมุนตามเข็มนาฬิกา) ป้อนบัตรเข้าไปในเครื่อง

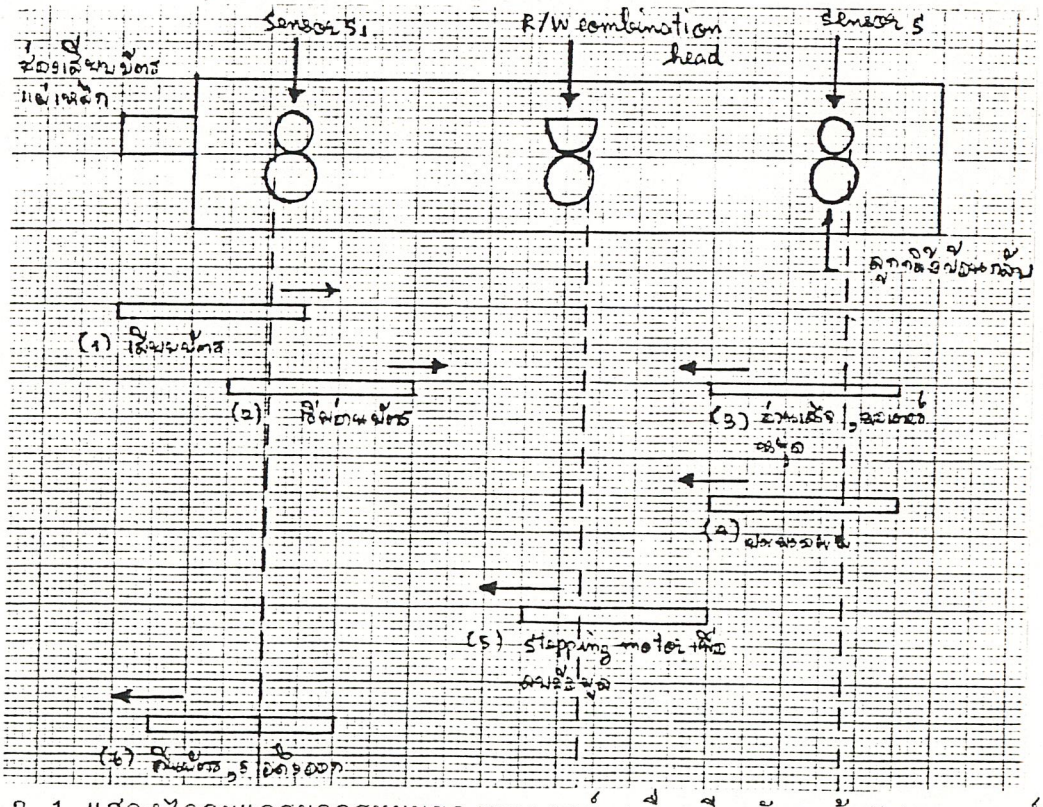
(2) เมื่อบัตรเคลื่อนมาถึงตำแหน่งหัวอ่าน-เขียน จะมีสัญญาณ READ GATE ทำให้วงจรอ่านทำงาน ข้อมูลที่บันทึกอยู่บนบัตรจะถูกอ่านด้วยหัวอ่านของ combination head

(3) เมื่อบัตรเคลื่อนผ่านหัวอ่านจนหมดสัญญาณ READ GATE จะหายไป ตัวตรวจจับที่ปลายทางจะทำงานเป็นผลให้มอเตอร์หยุดหมุน (หมุนตามเข็มนาฬิกา)

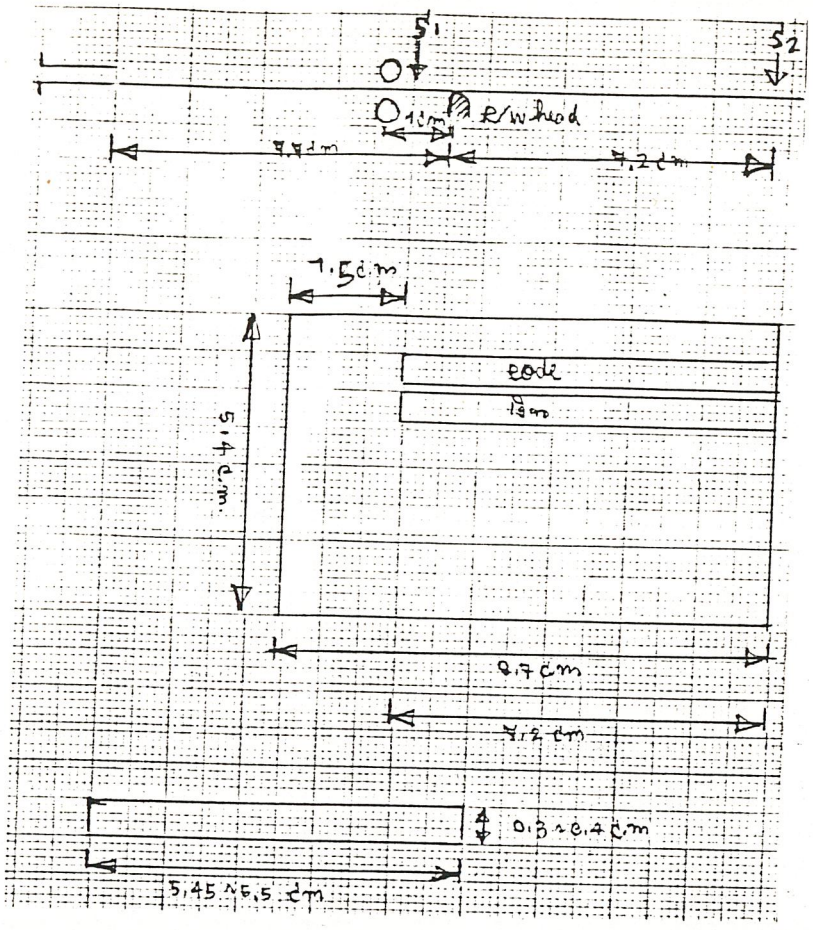
(4) ข้อมูลที่อ่านได้จะถูกตรวจสอบด้วยอุปกรณ์ภายนอกเครื่อง และจะนำไปประมวลผล ค่าที่เป็นจำนวนเงินของบัตรจะถูกแสดงที่ส่วน display

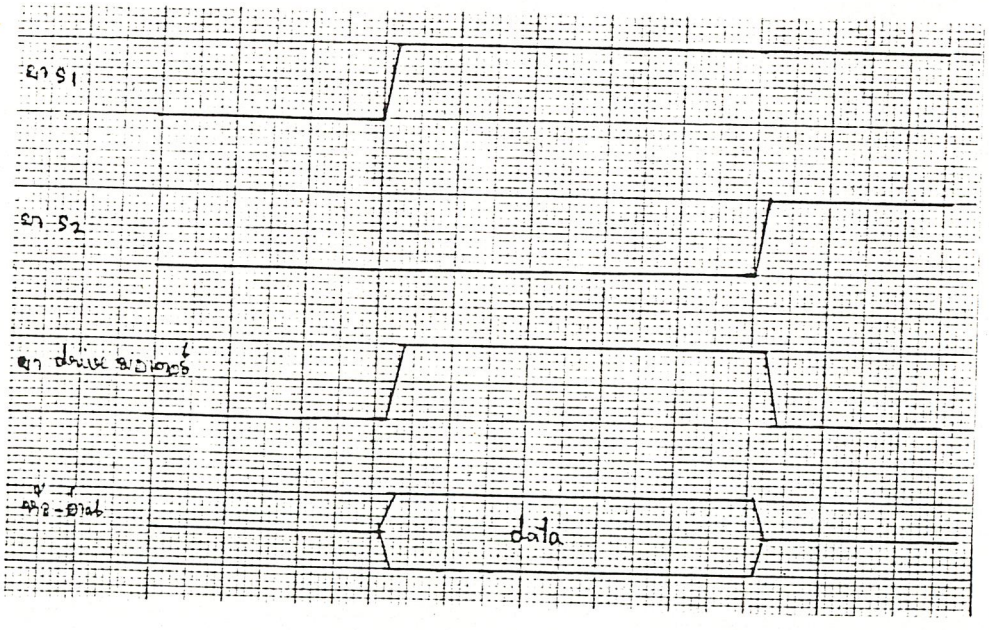
(5) เมื่อสิ้นสุดการสนทนา มอเตอร์จะหมุนทวนเข็มนาฬิกาเป็นสเต็ปเพื่อทำการลบข้อมูลที่ใช้แล้วออกไป

(6) มอเตอร์จะหมุนเพื่อทำการคืนบัตรออกมาทางช่องเสียบ เพื่อรอการดึงออก

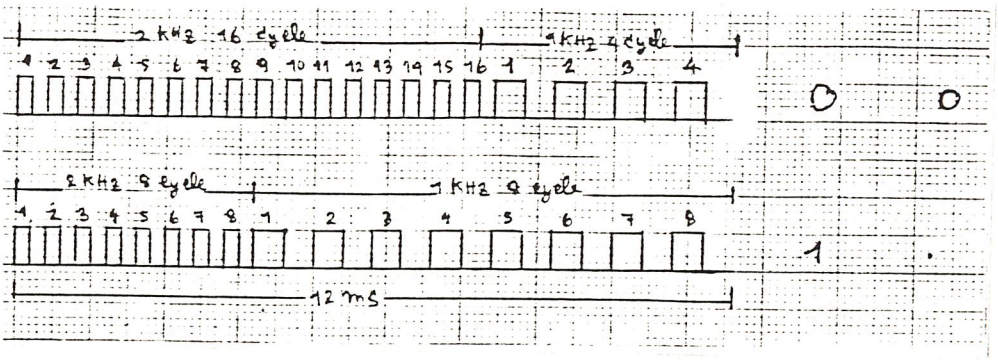


รูปที่ 2.1 แสดงไดอะแกรมการหมุนของมอเตอร์ เมื่อเสียบบัตรเข้าไป มอเตอร์จะหมุนเมื่อสัญญาณขาที่ออกจาก S.1 เปลี่ยนจาก 0 เป็น 1 มอเตอร์จะหยุดหมุนเมื่อสัญญาณขาที่ออกจาก S.1 และ S.2 เป็น 1 และเมื่อมอเตอร์หมุน(ตามเข็มนาฬิกา) หัวอ่านก็จะเริ่มอ่านข้อมูล





รูปที่ 2.2 แสดง Timing Diagram



รูปที่ 2.3 แสดง Bit Format

รูปที่ 2.4 แสดง Byte Format

การเขียนข้อมูลบนเนื้อเทปนั้นเราจะใช้การเก็บข้อมูลลงในหน่วยความจำก่อนแล้วจึงค่อยบันทึกลงในเนื้อเทปตามแอดเดรสที่เรากำหนดว่าจากตำแหน่งใดถึงตำแหน่งใด โดยเนื้อเทปที่เราใช้ จะใช้เพียง 2 แทรก แทรกหนึ่งใช้เป็นรหัสข้อมูลที่เป็นความลับ อีกแทรกหนึ่งใช้เป็นรหัสข้อมูลที่เป็นจำนวนเงินที่เหลือ เมื่อบัตรถูกอ่านโดยหัวอ่านรหัสข้อมูลข้อมูลทั้งสองจะถูกอ่านพร้อมกัน โดยรหัสข้อมูลที่เป็นความลับจะต้องตรงกับรหัสข้อมูลที่เราเก็บไว้ในหน่วยความจำประเภท ROM นอกจากนี้แล้วรหัสข้อมูลที่แสดงจำนวนเงินจะต้องมีจำนวนเงินที่เหลือเพียงพอด้วย ไมโครคอนโทรลเลอร์จึงจะสั่งการให้เชื่อมต่อกับสายโทรศัพท์ได้

เมื่อสิ้นสุดการสนทนา เวลาที่ใช้ในขณะสนทนาทั้งหมดที่นับโดยวงจร RTC จะถูกเปลี่ยนเป็นจำนวนเงินที่ใช้ แล้วนำไปลบกับค่าสูงสุด ได้ค่าเท่าไร สเต็ปปิ้งมอเตอร์จะหมุนตามจำนวนนั้น แล้วหัวอ่านจึงค่อยลบ เช่นการ์ดมีจำนวนเงินเหลือ 4 บาท เมื่อสิ้นสุดการสนทนาใช้ไปทั้งสิ้น 3 บาท นำไปลบกัน ได้ค่าเท่ากับ 1 สเต็ปปิ้งมอเตอร์จะหมุนไป 1 สเต็ป แล้วหัวอ่านจึงค่อยทำการลบ พร้อมกับมอเตอร์หมุนเพื่อเลื่อนบัตรไปทางด้านออก

การออกแบบโปรแกรม

การออกแบบโปรแกรมที่นำมาใช้งานสำหรับการควบคุมระบบทั้งหมดนั้นจะใช้โปรแกรมภาษาแอสเซมบลี้ (ASM51) โดยใช้ตัวคอมไพล์คือ CROSS 32 คอมไพเลอร์ ในการเขียนโปรแกรมควบคุมการทำงานนั้นจะแบ่งเป็นโปรแกรมย่อยๆต่างๆ เพื่อความสะดวกในการใช้งานซึ่งบระกอบด้วยส่วนต่างๆดังต่อไปนี้

- init_RAM เป็นโปรแกรมย่อยที่ใช้ในการย้ายข้อมูลจากหน่วยความจำชนิด EPROM ของ 8031 ไปเก็บไว้ยังหน่วยความจำชนิด RAM ที่อยู่ภายนอก เนื่องจากในการใช้งานของโปรแกรมซึ่งมีการใช้ข้อมูลซึ่งเป็นลักษณะของข้อความซึ่งต้องมีการเก็บไว้ใน EPROM ดังนั้นในการเริ่มต้นใช้งานจึงต้องมีการโอนย้ายข้อมูลเหล่านี้ไปเก็บไว้ในหน่วยความจำภายนอกก่อนที่จะเริ่มการใช้งาน

- init_LCD เป็นโปรแกรมย่อยที่จะทำการกำหนดสถานะเริ่มต้นของไอซีที่ใช้เป็นตัวขับ LCD ซึ่งจะต้องเซตค่าต่างๆสำหรับการเริ่มต้นการใช้งาน LCD

- init_RTC เป็นโปรแกรมย่อยซึ่งจะกำหนดสถานะเริ่มต้นของไอซีที่ทำหน้าที่เหมือนนาฬิกาในการจับเวลา และการตั้งเวลาเริ่มต้นให้กับนาฬิกา พร้อมทั้งให้นาฬิกาเริ่มทำงาน

- WriteLn เป็นการนำข้อมูลจากหน่วยความจำมาแสดงผลที่จอ LCD โดยที่ข้อมูลเหล่านี้จะถูกส่งผ่านพอร์ท 8255 ซึ่งต่อเข้ากับ LCD ส่วนขา control ทั้ง 3 ขาของ LCD นั้นจะส่งผ่านพอร์ท P1.5-P1.7 ของ 8031 และจะต้องมีโปรแกรมโมดูลส่วนที่ทำการสร้างพัลส์ (epulse) เพื่อทำให้สถานะการเขียนข้อมูลของ LCD ถูกต้องด้วย

- Get_Time เป็นโปรแกรมย่อยที่ทำการอ่านค่าเวลาซึ่งประกอบด้วย วินาที, นาที, ชั่วโมง, วัน, เดือน จาก RTC มาเก็บไว้ที่หน่วยความจำ

- Disp_Time เป็นการนำข้อมูลซึ่งเป็นค่าของเวลาซึ่งที่อยู่ในหน่วยความจำในโปรแกรมย่อย Get_Time มาแสดงผลที่จอ LCD ด้วยโปรแกรมย่อย WriteLn

- Diff_Time เป็นโปรแกรมย่อยที่ทำการคำนวณเวลาที่ใช้ไปในการสนทนาแต่ละครั้ง โดยจะนำเวลาตอนเริ่มต้นในการสนทนา มาลบออกจากเวลาที่การสนทนาเสร็จสิ้นแล้ว ซึ่งในโปรแกรมนี้จะมีโปรแกรมย่อย BCD2Bin ซึ่งเป็นการแปลงเวลาที่เป็นเลข BCD ให้เป็นเลขไบนารีก่อนแล้วจึงทำการลบกัน

- DIS เป็นโปรแกรมย่อยซึ่งจะนำเวลาที่ใช้ไปในการสนทนามาแสดงผลที่จอ LCD ซึ่งจะมีโปรแกรมย่อย Bin2BCD ซึ่งเป็นการแปลงเวลาจากเลขไบนารีให้เป็น BCD

ก่อน แล้วจึงนำเวลาที่ใช้ ไปเก็บไว้ในหน่วยความจำ จากนั้นจึงนำไปแสดงผลที่จอ LCD ด้วยโปรแกรมย่อย WriteLn

- PLY เป็นการนำเวลาที่ใช้ไปในการสนทนามาแปลงเป็นจำนวนเงินที่ใช้สำหรับการสนทนา คือเป็นการคำนวณเป็นราคา โดยกำหนดไว้ว่าการสนทนาเป็นเวลา 3 นาที คิดเป็นจำนวณเงิน 1 บาท เมื่อคำนวณเวลาให้เป็นจำนวนเงินแล้วก็จะใช้โปรแกรมย่อย Bin2BCD เปลี่ยนจากจำนวนเงินที่เป็นไบนารี่ ให้เป็นเลข BCD แล้วจึงเก็บไว้ที่หน่วยความจำ จากนั้นจึงนำไปแสดงผลที่จอ LCD โดยโปรแกรมย่อย WriteLn

- SCANKB เป็นการตรวจสอบการกดคีย์บอร์ดซึ่งเป็นตัวเลขต่างๆ แล้วแสดงหมายเลขที่กดนั้นๆบนจอ LCD โดยใช้พอร์ท 8255 พอร์ท PC_0-PC_7 โดยการให้ PC_0-PC_3 เป็นพอร์ทซึ่งใช้ในการส่งข้อมูลเพื่อตรวจสอบ และ PC_4PC_7 เป็นพอร์ทรับข้อมูลการกดคีย์ เพื่อนำมาตรวจสอบว่าคีย์ใดถูกกด โดยที่การแสดงผลที่จอ LCD นั้นจะแสดงหมายเลขที่กดเพียง 7 ตัวเลขเท่านั้น

- INT เป็นโปรแกรมย่อยการบริการอินเตอร์รัพท์ (interrupt service routine) ซึ่งเป็นการตรวจสอบระยะเวลาที่ฮุคสวิทช์ ON หมายถึงว่า ถ้าฮุคสวิทช์ ON นานน้อยกว่า 2 วินาที ก็จะเป็นการ flash hook ถ้าฮุคสวิทช์ ON นานเกิน 2 วินาที แสดงว่าเป็นการ on hook โดยจะทำการตรวจสอบบิตของฮุคสวิทช์ ในกรณีที่เป็น flash hook โปรแกรมย่อยจะทำการรับหมายเลขเข้ามาใหม่ เพื่อใช้ในการโทรครั้งใหม่ แต่ถ้าเป็น on hook โปรแกรมย่อยจะสั่งให้คืนบัตร และหยุดการทำงาน

บทที่ 3 ผลการทดลอง

วงจร Speech cct.&Tone Dialer

จากการทดลองโดยการต่อวงจรตามรูป แล้วนำมาลองใช้งานจริงปรากฏว่ายังไม่สามารถส่งสัญญาณ DTMF ออกไปได้ ดังนั้นจึงทำการวัดดูสัญญาณที่ขา TI ของ MC34014 เทียบกับสัญญาณที่ขา DTMF ของ MC145412 ปรากฏว่าระดับสัญญาณที่รับได้ที่ขา TI มีระดับต่ำกว่าที่ขา DTMF ดังนั้นจึงทำการปรับค่าความต้านทานจาก 100KΩ แล้วดูรูปคลื่นสัญญาณตอนที่กดคีย์ส่ง DTMFปรับจนกระทั่งรูปคลื่นสัญญาณไม่ถูกตัด(clip) จนได้ระดับสัญญาณสูงสุด ปรากฏว่าได้ความต้านทาน 4.7 KΩ

วงจรไมโครคอนโทรลเลอร์

สำหรับการทดลองใช้งานไมโครคอนโทรลเลอร์นั้น ทดลองโดยการเขียนโปรแกรมทดสอบการทำงานโดยเขียนข้อมูลผ่านออกพอร์ท 8255 ปรากฏว่าข้อมูลที่กำหนดในโปรแกรมกับข้อมูลที่วัดได้ที่แต่ละพอร์ทของ 8255 นั้นตรงกัน ยกเว้นเพียงบิทเดียวคือ PA_4 ซึ่งจะเป็นลอจิก 0 ตลอด หลังจากนั้นจึงทำการเปลี่ยน 8255 ตัวใหม่ ผลปรากฏว่าข้อมูลที่ได้จากพอร์ทมีค่าตรงกับที่กำหนดไว้ในโปรแกรมทุกบิท

การเชื่อมต่อวงจรไมโครคอนโทรลเลอร์กับ LCD module

การต่อ LCD module กับไมโครคอนโทรลเลอร์ โดยผ่านพอร์ท 8255 นั้น จากการทดลองครั้งแรกโดยการต่อขา data ของ LCD กับพอร์ท PA_0-PA_7 ของ 8255 และขา control ของ LCD กับพอร์ท PB_0-PB_2 นั้น ปรากฏว่าจอ LCD ไม่สามารถแสดงผลได้ ดังนั้นจึงทำการแก้ไขโปรแกรมเล็กน้อย และลองเปลี่ยนจากการต่อขา control ของ LCD ไปต่อกับพอร์ทแลทช์ P1.5-P1.7 ของ 8031 แทน ผลปรากฏว่า LCD ก็สามารถแสดงผลได้อย่างถูกต้อง

การชื่อมต่อวงจรไมโครคอนโทรลเลอร์กับ RTC

ในการต่อ RTC เบอร์ MM58167 กับวงจรไมโครคอนโทรลเลอร์นั้น ในการทดลองครั้งแรกปรากฏว่า RTC ซึ่งทำงานเป็นนาฬิกานั้น ยังไม่สามารถทำงานได้ ดังนั้นจึงมาตรวจสอบดูที่โปรแกรมในการอ่านค่าเวลาจาก RTC แล้วแก้ไขโปรแกรมซึ่งเป็นบิท

control บางบิทของ RTC ปรากฏว่านาฬิกาเริ่มทำงานได้ แต่ยังเดินไม่เที่ยงตรง จึงต้องทำการปรับเปลี่ยนค่า R และ C ที่ขา OSC_{in} และ OSC_{out} จนกระทั่งเวลาเดินได้อย่างเที่ยงตรงแม่นยำ

วงจรตรวจจับ Ringback Tone

ในส่วนของวงจรตรวจจับการสิ้นสุดของ Ringback Tone นั้นต้องมีการปรับค่าความต้านทาน จึงต้องใช้ความต้านทานชนิด Trimmer ซึ่งต้องการการปรับอย่างละเอียดเพื่อใช้ตรวจจับความถี่ 400 Hz ในวงจร PLL และต้องเลือกค่าตัวเก็บประจุและตัวต้านทาน ในส่วนของวงจรโมโนสเตเบิ้ลให้ได้ค่าเวลาที่เหมาะสมตามต้องการ

```
************************** PROGRAM CONTROL ****************************

;---------------------------- Define Part -------------------------------
b:                  equ     0F0h
acc:                equ     0E0h
PSW:                equ     0D0h
IP:                 equ     0B8h
P3:                 equ     0B0h
IE:                 equ     0A8h
P2:                 equ     0A0h
SBUF:               equ      99h
SCON:               equ      98h
P1.0:               equ      90h
P1.1:               equ     P1.0+1
P1.2:               equ     P1.0+2
P1.3:               equ     P1.0+3
P1.4:               equ     P1.0+4
P1.5:               equ     P1.0+5
P1.6:               equ     P1.0+6
P1.7:               equ     P1.0+7
TH1:                equ      8Dh
TH0:                equ      8Ch
TL1:                equ      8Bh
TL0:                equ      8Ah
TMOD:               equ      89h
TCON:               equ      88h
PCON:               equ      87h
DPH:                equ      83h
DPL:                equ      82h
SP:                 equ      81h
P0:                 equ      80h
acc.0:              equ     ACC+0
acc.1:              equ     ACC+1
acc.2:              equ     ACC+2
acc.3:              equ     ACC+3
acc.4:              equ     ACC+4
acc.5:              equ     ACC+5
acc.6:              equ     ACC+6
acc.7:              equ     ACC+7

BegRAM:              equ    1000h              ; RAM Address of 8031
Ctr_8255:            equ    3003h              ; Port Address of 8255
PDC8255:             equ    3002h
PDB8255:             equ    3001h
LCDD:                equ    3000h              ; LCD Data Address

RTC_Base:            equ    2000h              ; Real Time Clock Address MAP
RTC_Sec1:            equ    RTC_Base+0
RTC_Sec2:            equ    RTC_Base+1
RTC_Sec3:            equ    RTC_Base+2
RTC_Min:             equ    RTC_Base+3
RTC_Hour:            equ    RTC_Base+4
RTC_DayW:            equ    RTC_Base+5
RTC_DayM:            equ    RTC_Base+6
RTC_Month:           equ    RTC_Base+7
Int_S_R:             equ    RTC_Base+10
Int_C_R:             equ    RTC_Base+11
Count_Rst:           equ    RTC_Base+12
RAM_Rst:             equ    RTC_Base+13
```

```
Stats_B:          equ     RTC_Base+14
Go_Com:           equ     RTC_Base+15
Stby_Int:         equ     RTC_Base+16
CallStartTime:    equ     1600h
CallStopTime:     equ     1602h
CallDiffTime:     equ     1604h

PtrHi:            equ     07Dh                ; High byte of RAMPtr
PtrLo:            equ     07Eh                ; Low byte of RAMPtr
Hund:             equ      31h                ; Hundredth digit

;---------------------------- Macro Part ----------------------------
out:      macro   PORTNUMBER, DATA    ; OutPort macro from Data to PortNo
          mov     a,DATA
          mov     DPTR, PORTNUMBER
          movx    @DPTR,a
           endm

outa:     macro   PORTNUMBER              ; OutPort macro from A to PortNo
          mov     DPTR, PORTNUMBER
          movx    @DPTR,a
           endm

inpa:     macro   PORTNUMBER              ; InPort macro from PortNo to A
          mov     DPTR, PORTNUMBER
          movx    a, @DPTR
           endm

pushDPTR: macro                           ; Push DPTR macro
          push    DPH
          push    DPL
            endm

popDPTR:  macro                           ; Pop DPTR macro
          pop     DPL
          pop     DPH
           endm

pushall:  macro                           ; Push all register macro
          push    acc
          push    b
          push    DPH
          push    DPL
           endm

popall:   macro                           ; Pop all register macro
          pop     DPL
          pop     DPH
          pop     b
          pop     acc
           endm

delay:    macro   TIME
          mov     r0, TIME
d1:       mov     r1, 0
d2:       djnz    r1, d2
          djnz    r0, d1
           endm

movb:     macro   SOURCE, DEST        ; Copy one byte from Source to Dest
```

```
        push    acc
        push    DPH
        push    DPL
        mov     DPTR, SOURCE
        movx    a,@DPTR
        mov     DPTR, DEST
        movx    @DPTR,a
        pop     DPL
        pop     DPH
        pop     acc
          endm

movw:    macro  SOURCE, DEST        ; Copy one word from Source to Dest
        movb    SOURCE, DEST
        movb    SOURCE+1, DEST+1
         endm

movDPTRPtr: macro                          ; Copy DPTR to RAMPtr
        mov     PtrHi, DPH
        mov     PtrLo, DPL
            endm

movPtrDPTR: macro                          ; Copy RAMPtr to DPTR
        mov     DPH, PtrHi
        mov     DPL, PtrLo
            endm

;----------------------------  Main Part  ----------------------------

        CPU     "8051.TBL"
        HOF     "INT8"
        org     0000H              ; ROM Code start up at 0000h
        ljmp    ST
        org     0003h              ; Interrupt Service Routine
        ljmp    INT

ST:     setb    P1.3               ; Disconnect with telephone line
        acall   Init_RAM           ; Initial External RAM
        out     #Ctr_8255, #88h    ; out 88h to 8255 Port
        acall   Init_LCD           ; Initialize LCD Module
        acall   Init_RTC           ; Initialize RTC time
        mov     PSW, #00000000B

        jb      P1.0, STOP         ; If code in card correct then
        clr     P1.3               ; connect with telephone line

TO:     mov     IE, #10000001B     ; Enabel interrupt INT0
        clr     01h
KOK:    jb      01h, STOP

        lcall   RETEL              ; Call Processing

MNO:    jb      P1.2, MNO      ; Jump loop wait for end of ringbacktone
        mov     IE, #00000000B     ; Disable interrupt

        acall   Get_Time
        movb    #RTC_Hour, #CallStartTime       ; Get start HOUR in RAM
        movb    #RTC_Min,  #CallStartTime+1     ; Get start MIN in RAM
LMN:    acall   Disp_Time
        jb      P1.1, LMN                   ; If hook switch "HOOK ON"

        movb    #RTC_Hour, #CallStopTime        ; Get stop HOUR in RAM
```

```
        movb    #RTC_Min,  #CallStopTime+1      ; Get stop MIN in RAM
        acall   Diff_Time
                                        ; Compute the time of conversation
        acall   DIS                     ; Disply time of conversation
        acall   PLY                     ; Disply fee of conversation
        setb    P1.3                    ; Disconnect telephone line
        ajmp    ENDD                    ; Jump to end of program

;--------------------------- Procedure Part ----------------------------
STOP:   mov     a, #4
        mov     DPTR, #CdOut
        acall   WriteLn
        setb    P1.3
        ajmp    ENDD

DIS:    inpa    #CallDiffTime
        lcall   Bin2BCD
        push    acc
        anl     a, #0Fh
        add     a, #30h
        outa    #Time_Used+8
        pop     acc
        swap    a
        anl     a, #0Fh
        add     a, #30h
        outa    #Time_Used+7

PO:     mov     a, #3
        mov     DPTR, #Time_Used
        acall   Writeln
        ret

PLY:    inpa    #CallDiffTime
        mov     b ,#3
        div     ab
        mov     r5, b
        cjne    r5,#00h,ABC
        ajmp    DEF
ABC:    inc     a
DEF:    lcall   Bin2BCD
        push    acc
        anl     a, #0Fh
        add     a, #30h
        outa    #Price+8
        pop     acc
        swap    a
        anl     a, #0Fh
        add     a, #30h
        outa    #Price+7
TR:     mov     a, #4
        mov     DPTR, #Price
        acall   WriteLn
        ret

Diff_Time:
        pushall
        inpa    #CallStartTime          ; Get Start'HH to R0
        acall   BCD2Bin                 ; Convert to Binary
        mov     R0, a
        inpa    #CallStopTime           ; Get Stop'HH to A
```

```
        acall   BCD2Bin                 ; Convert to Binary
        clr     c                       ; clear carry flag befor use subb
        subb    a, R0                   ; A = Stop'HH - Start'HH
        jnc     NoAdjust                ; A < 0 ?
        add     a, #24                  ; Yes, adjust Hour by 24
NoAdjust:
        mov     b, #60                  ; Convert A to Minute
        mul     ab
        mov     R0, a                   ; Save LSB product in R0
        inpa    #CallStopTime+1         ; Get Stop'MM to A
        acall   BCD2Bin                 ; Convert to Binary
        add     a, R0                   ; add to A (MM)
        jnc     NoCarry                 ; A > 255 ?
        inc     b                       ; Yes, adjust MSB too
NoCarry:
        push    acc
        inpa    #CallStartTime+1        ; Get Start'MM to A
        acall   BCD2Bin                 ; Convert to Binary
        mov     R0, a
        pop     acc
        clr     c                       ; clear carry flag befor use subb
        subb    a, R0                   ; A = MM - Start'MM
        jnc     NoBorrow                ; Set Borrow ?
        dec     b                       ; Yes, Decrement MSB
NoBorrow:
        inc     a                       ; Round off A
        outa    #CallDiffTime           ; and Save to CalDiffTime
        popall
        ret

BCD2Bin:                                ; Convert BCD in A to Bin in A
        mov     R7, a
        anl     a, #0F0h
        rr      a
        mov     R6, a
        rr      a
        rr      a
        add     a, R6
        mov     R6, a
        mov     a, R7
        anl     a, #0Fh
        add     a, R6
        ret

Bin2BCD:                                ; Convert Bin in A to BCD in A
        mov     b, #100
        div     ab
        mov     hund, a
        mov     a, #10
        xch     a, b
        div     ab
        swap    a
        add     a, b
        ret

Disp_Time:
        mov     R0, 20
Pi1:    mov     R1, 0
Pi2:    djnz    R1, Pi2
        djnz    R0, Pi1
```

```
        acall   Get_Time                ; Disply DATE
        mov     a, #3
        mov     DPTR, #Date_Mess
        acall   WriteLn
        mov     a, #4                   ; Disply TIME
        mov     DPTR, #Time_Mess
        acall   WriteLn
        ret

Get_Time:
        inpa    #RTC_DayM               ; Get Day of Month
        push    acc
        anl     a, #0Fh
        add     a, #30h
        outa    #Date_Mess+7            ; then put to Date_Mess Dx-MM-19YY
        pop     acc
        swap    a
        anl     a, #0Fh
        add     a, #30h
        outa    #Date_Mess+6            ; put to Date_Mess xD-MM-19YY

        inpa    #RTC_Month              ; Get Month
        push    acc
        anl     a, #0Fh
        add     a, #30h
        outa    #Date_Mess+10           ; then put to Date_Mess DD-Mx-19YY
        pop     acc
        swap    a
        anl     a, #0Fh
        add     a, #30h
        outa    #Date_Mess+9            ; put to Date_Mess DD-xM-19YY

        inpa    #RTC_Hour               ; Get Hour
        push    acc
        anl     a, #0Fh
        add     a, #30h
        outa    #Time_Mess+7            ; then put to Time_Mess H-:MM:SS
        pop     acc
        swap    a
        anl     a, #0Fh
        add     a, #30h
        outa    #Time_Mess+6            ; put to Time_Mess -H:MM:SS

        inpa    #RTC_Min                ; Get Minute
        push    acc
        anl     a, #0Fh
        add     a, #30h
        outa    #Time_Mess+10           ; then put to Time_Mess HH:M-:SS
        pop     acc
        swap    a
        anl     a, #0Fh
        add     a, #30h
        outa    #Time_Mess+9            ; put to Time_Mess HH:-M:SS

        inpa    #RTC_Sec3               ; Get Second
        push    acc
        anl     a, #0Fh
        add     a, #30h
        outa    #Time_Mess+13           ; then put to Time_Mess HH:MM:S-
```

```
            pop     acc
            swap    a
            anl     a, #0Fh
            add     a, #30h
            outa    #Time_Mess+12          ; put to Time_Mess HH:MM:-S
            ret

WriteLn:
            acall   ConvLine
            pushDPTR
            outa    #LCDD
            clr     P1.6
            clr     P1.5
            acall   epulse
            popDPTR

            mov     R0, #16
WRL:        movx    a, @DPTR
            mov     R2, a
            mov     122, R0
            acall   wrbyte
            mov     R0, 122
            inc     DPTR
            djnz    R0, WRL
            ret

wrbyte:                                    ; Used with WriteLn procedure
            pushDPTR
            setb    P1.5
            out     #LCDD, R2
            acall   epulse
            popDPTR
            clr     P1.5
            ret

epulse:                                    ; Used with WriteLn procedure
            pushall
            setb    P1.7
            mov     R0, 50
Pm1:        mov     R1, 0
Pm2:        djnz    R1, Pm2
            djnz    R0, Pm1
            clr     P1.7
            popall
            ret

ConvLine:                                  ; Convert Line to used in LCD
            movc    a,@a+PC
            ret
            dfb     080h                   ; Line 1
            dfb     0C0h                   ; Line 2
            dfb     090h                   ; Line 3
            dfb     0D0h                   ; Line 4

Init_LCD:                                  ; Initial LCD Module
            mov     R0, 50
WW1:        mov     R1, 0
WW2:        djnz    R1, WW2
            djnz    R0, WW1
            clr     P1.5
```

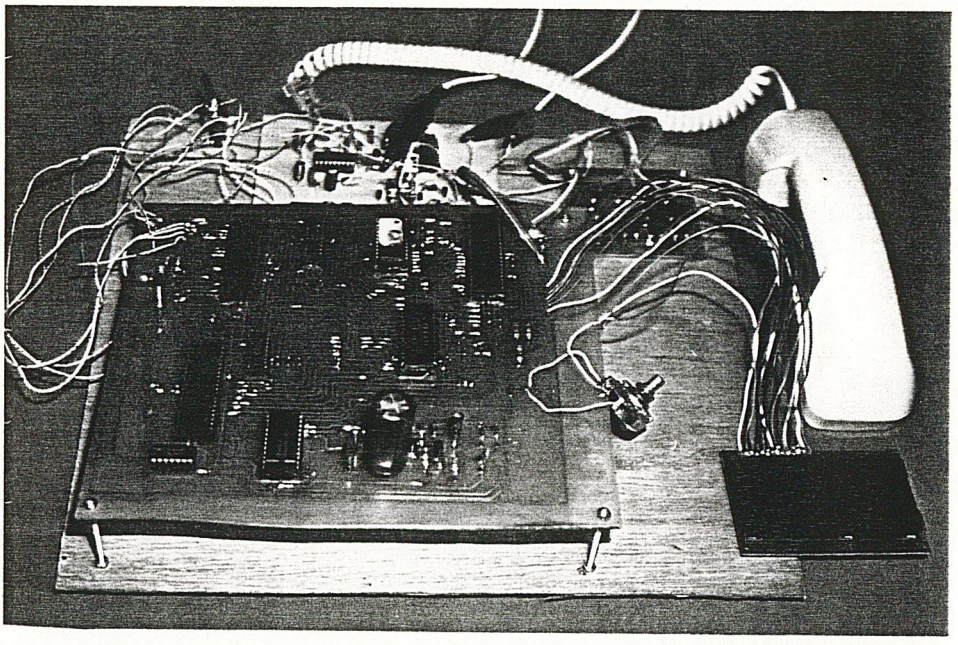
```
        clr     P1.6
        clr     P1.7
        out     #LCDD, #00111000B  ; fn set 38h DL=1 8 bit,N=1 1/16
        acall   epulse             ; Enable signal pulse

        mov     R0, 20
F1:     mov     R1, 0
F2:     djnz    R1, F2
        djnz    R0, F1             ; delay 5 mSec

        out     #LCDD, #00001111B  ; display on/off(D=On,C=On,B=Blink)
        acall   epulse
        mov     R0, 20
SS1:    mov     R1, 0
SS2:    djnz    R1, SS2
        djnz    R0, SS1
        out     #LCDD, #00000110B ; Entry mode set I/D=1 Inc,S=0 Right
        acall   epulse
        mov     R0, 20
NN1:    mov     R1, 0
NN2:    djnz    R1, NN2
        djnz    R0, NN1
        out     #LCDD, #00000001B  ; Clear all display
        acall   epulse

        mov     R0, 20
G1:     mov     R1, 0
G2:     djnz    R1, G2
        djnz    R0, G1
        ret

Init_RAM:
        mov     DPTR, #BegRAM
        movDPTRPtr
        mov     DPTR, #BegROM           ; Point to ROM Data
        mov     R0, #{EndROM-BegROM}    ; Calculate Length of ROM Data
iLoop:  clr     a
        movc    a, @a+DPTR              ; Transfer data
        pushDPTR
        movPtrDPTR
        movx    @DPTR, a
        inc     DPTR
        movDPTRPtr
        popDPTR
        inc     DPTR
        djnz    R0, iLoop
        ret

Init_RTC:
        out     #Int_C_R , #04H
        out     #RTC_Sec1, #00H         ; Set Time to 00:00:00
        out     #RTC_Sec2, #00H
        out     #RTC_Sec3, #00H         ; Set Date to 31-12-1994
        out     #RTC_Min , #00H
        out     #RTC_Hour, #00H
        out     #RTC_DayM, #31H
        out     #RTC_Month,#12H
        out     #Go_Com,   #01H         ; Start !

        mov     a, #1
```

```
        mov     DPTR, #Set_Ti_Mess
        acall   WriteLn
        ret

SCANKB:         mov     R5, #00H        ; Scan keyboard
                mov     R4, #07H
SCANKEY:        mov     a, #0FEH
NLOOP:          xch     a, R7
                out     #PDC8255, R7

                mov     R0, 20
CV1:            mov     R1, 0
CV2:            djnz    R1, CV2
                djnz    R0, CV1

                inpa    #PDC8255
                mov     R6, a
                orl     a, #0FH
                cpl     a
                jnz     KEYIN
                xch     a, R7
                rl      a
                cjne    a,#0DFH,NLOOP

                mov     R0, 50
VC1:            mov     R1, 0
VC2:            djnz    R1, VC2
                djnz    R0, VC1

                ljmp    SCANKEY

KEYIN:          mov     a, R7
                jnb     ACC.0, NPE
                jnb     ACC.1, NPD
                jnb     ACC.2, NPB
                jnb     ACC.3, NP7

                mov     R0, 50
KL1:            mov     R1, 0
KL2:            djnz    R1, KL2
                djnz    R0, KL1

                ljmp    SCANKEY

NPE:            mov     a, R6
                jnb     ACC.4, PP1
                jnb     ACC.5, PP2
                jnb     ACC.6, PP3
                jnb     ACC.7, PP4
                ljmp    SCANKEY
PP1:            ljmp    PR1
PP2:            ljmp    PR2
PP3:            ljmp    PR3
PP4:            ljmp    PR4

NPD:            mov     a, R6
                jnb     ACC.4, PP5
                jnb     ACC.5, PP6
                jnb     ACC.6, PP7
                jnb     ACC.7, PP8
```

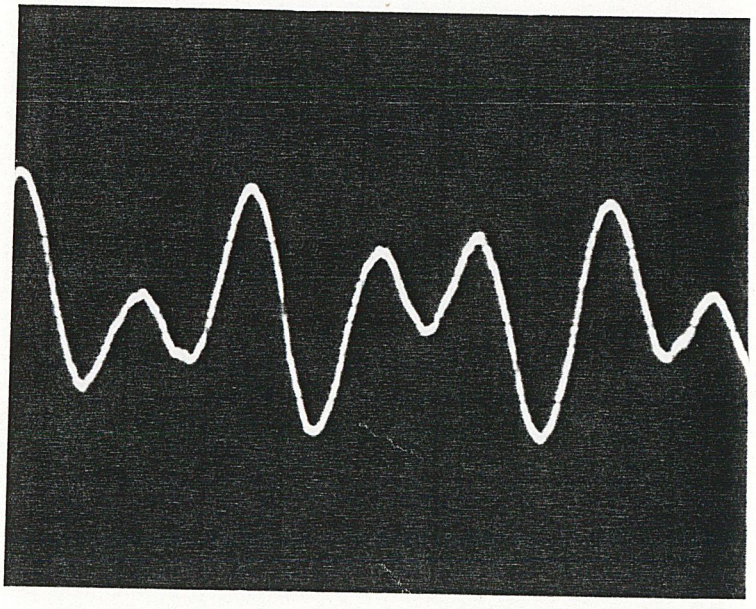
```
                ljmp    SCANKEY
PP5:            ljmp    PR5
PP6:            ljmp    PR6
PP7:            ljmp    PR7
PP8:            ljmp    PR8

NPB:            mov     a, R6
                jnb     ACC.4, PP9
                jnb     ACC.5, PP10
                jnb     ACC.6, PP11
                jnb     ACC.7, PP12
                ljmp    SCANKEY
PP9:            ljmp    PR9
PP10:           ljmp    PR10
PP11:           ljmp    PR11
PP12:           ljmp    PR12

NP7:            mov     a, R6
                jnb     ACC.4, PP13
                jnb     ACC.5, PP14
                jnb     ACC.6, PP15
                jnb     ACC.7, PP16
                ljmp    SCANKEY
PP13:           ljmp    PR13
PP14:           ljmp    PR14
PP15:           ljmp    PR15
PP16:           ljmp    PR16

PR1:            mov     a, #01H
                ljmp    END1
PR2:            mov     a, #02H
                ljmp    END1
PR3:            mov     a, #03H
                ljmp    END1
PR4:            mov     a, #04H
                ljmp    END1
PR5:            MOV     a, #04H
                ljmp    END1
PR6:            mov     a, #05H
                ljmp    END1
PR7:            MOV     a, #06H
                ljmp    END1
PR8:            mov     a, #08H
                ljmp    END1
PR9:            mov     a, #07H
                ljmp    END1
PR10:           MOV     a, #08H
                ljmp    END1
PR11:           mov     a, #09H
                ljmp    END1
PR12:           mov     a, #0BH
                ljmp    END1
PR13:           mov     a, #0CH
                ljmp    END1
PR14:           mov     a, #00H
                ljmp    END1
PR15:           mov     a, #0EH
                LJMP    END1
PR16:           mov     a, #0FH
                ljmp    END1
```

```
END1:       cjne    R5,#00H,XYZ1
            mov     DPTR, #Phone_Mess+8
            ljmp    XYZ2
XYZ1:       mov     DPL, 124
            mov     DPH, 123
XYZ2:       push    acc
            anl     a, #0FH
            add     a, #30H
            movx    @DPTR, a
            pop     acc
            inc     DPTR
            mov     124, DPL
            mov     123, DPH
            mov     a, #2
            mov     DPTR, #Phone_Mess
            acall   WriteLn
            dec     R5
            djnz    R4, SCANAGAIN
            ret

SCANAGAIN:  ljmp    SCANKEY

INT:                                    ; Interrupt service routine
            mov     R7, #0CH
DELY:       mov     R0, #0FFH           ; Delay about 2 second
DEL0:       mov     R1, #0FFH
DEL1:       djnz    R1, DEL1
            djnz    R0, DEL0
            djnz    R7, DELY

            mov     IE, #00000000B      ; Disable interrupt
            jnb     P1.1, CARDOUT       ; If hook switch "FLASH HOOK"

                                        ; then recall
            mov     a, #20h             ; IF hook switch "ON HOOK"

            mov     DPTR, #Phone_Mess+8 ; then end of processing
            movx    @DPTR, a
            mov     R3, #07h
RTR:        inc     DPTR
            movx    @DPTR, a
            djnz    R3, RTR

            mov     a, #1
            mov     DPTR, #Space
            acall   WriteLn
            mov     a, #2
            mov     DPTR, #Space
            acall   WriteLn

            mov     R1, 081h
            mov     @R1, #00h
            dec     R1
            mov     @R1, #1Ch
            inc     R1
            mov     081h, R1
            reti

RETEL:      mov     a, #1                       ; Recall processing
            mov     DPTR, #Line1
            acall   WriteLn
```

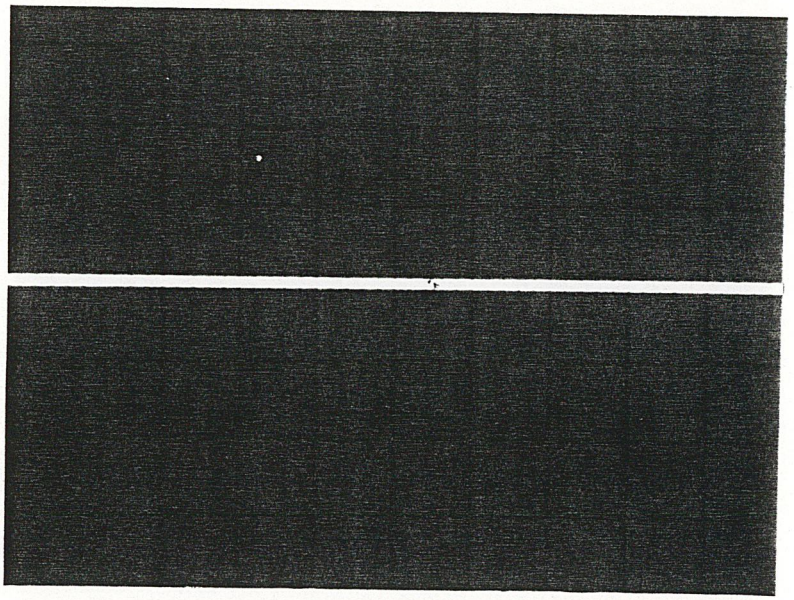
```
                acall   SCANKB
                ret

CARDOUT:        setb    01h
                mov     R1, 081h
                mov     @R1, #00h
                dec     R1
                mov     @R1, #21h
                inc     R1
                mov     081h, R1
                reti

;---------------------- eXternal RAM data Part ----------------------
BegROM:          equ    $
Time_Mess:              equ $-BegROM+BegRAM
                dfb     "Time: HH:MM:SS  "
Date_Mess:              equ $-BegROM+BegRAM
                dfb     "Date: DD/MM/1994"
Phone_Mess:             equ $-BegROM+BegRAM
                dfb     " Number         "
Set_Ti_Mess:            equ $-BegROM+BegRAM
                dfb     " Set Clock OK ! "
Line1:                  equ $-BegROM+BegRAM
                dfb     "  Push Number  "
Time_Used:              equ $-BegROM+BegRAM
                dfb     " Used XX  Min "
Price:                  equ $-BegROM+BegRAM
                dfb     " Paid XX Baht "
Deposite:               equ $-BegROM+BegRAM
                dfb     "The Rest XX Baht"
CdOut:                  equ $-BegROM+BegRAM
                dfb     "  .CARD...OUT.  "
Space:                  equ $-BegROM+BegRAM
                dfb     "                "
EndROM:          equ    $

ENDD:           END
```



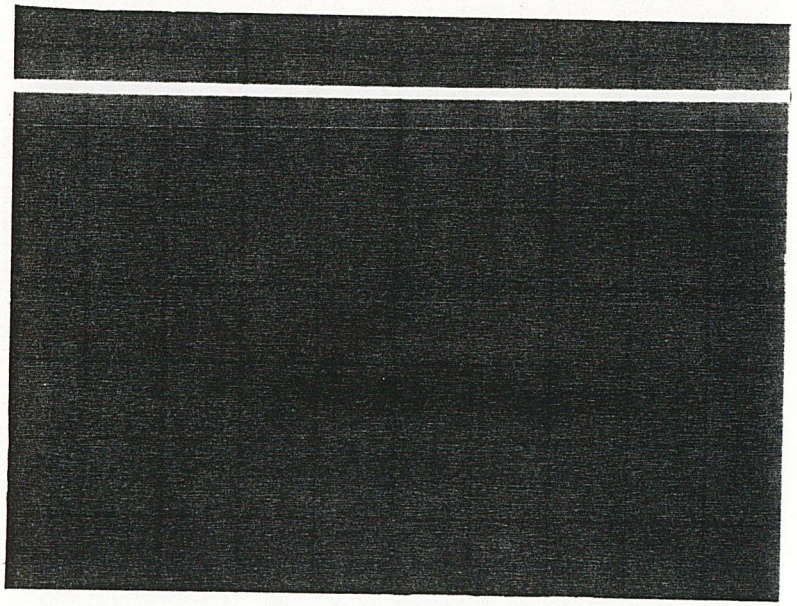
รูปที่ 3.1 แสดงวงจรทั้งหมดที่ใช้งาน



รูป 3.2 แสดงสัญญาณที่ขา 16 ของ MC34014 เมื่อมีการกดหมายเลข



รูปที่ 3.3 แสดงสัญญาณที่ขา 6 ของ 74LS121 ขณะที่มี ringback tone



รูปที่ 3.4 แสดงสัญญาณที่ขา 6 ของ 74LS121 ขณะที่สิ้นสุด ringback tone

บทที่ 4 สรุปผลการทดลอง

การเชื่อมต่อ LCD module กับวงจรไมโครคอนโทรลเลอร์

ในการใช้งานบิท control ของ LCD นั้นควรจะต่อกับพอร์ทแลทช์ของ 8031 โดยตรง เพื่อที่ว่าในกรณีของการส่งบิทเป็นพัลส์ของ 8031 ไปยังขา control ของ LCD นั้น จะทำให้พัลส์และช่วงเวลาของพัลส์มีความแน่นอนมากขึ้น

การเชื่อมต่อ RTC กับวงจรไมโครคอนโทรลเลอร์

ในการเชื่อมต่อ RTC กับวงจรไมตครคอนโทรลเลอร์นั้น ค่อนข้างจะไม่เป็นปัญหาเท่าใดนัก สำหรับทางฮาร์ดแวร์ เพราะขาข้อมูล และขาแอดเดรส ของ RTC นั้นแยกส่วนกัน ทำให้ไม่เกิดปัญหาเรื่องการมัลติเพล็กซ์กันของขาแอดเดรส และขาข้อมูล สำหรับ RTC เบอร์อื่นๆบางตัว แต่สำหรับ RTC เบอร์ MM58167 ที่ใช้งานอยู่นั้นจะมีปัญหาเพียงแค่การปรับค่าของ R และ C ที่ขา OSC_{in} และ OSC_{out} ให้ได้ค่าที่ถูกต้องที่สามารถทำให้นาฬิกาเดินเที่ยงตรงที่สุด

การควบคุมเครื่องอ่านบัตร

ในโครงงานนี้ยังไม่สามารถที่จะทำการทดลองเกี่ยวกับวิธีการอ่านบัตร หรือการควบคุมเครื่องอ่านบัตรได้ เนื่องจากข้อจำกัดทางด้านระยะเวลาของโครงงาน และความสลับซับซ้อนของกระบวนการและกรรมวิธีของเครื่องอ่านบัตร

โปรแกรมที่ใช้ในการควบคุม

ในการใช้งานโปรแกรมที่ทำการควบคุมการทำงานของระบบนั้น จะใช้ภาษาแอสเซมบลี้ ASM51 ซึ่งใช้ชุดคำสั่งของไมโครคอนโทรลเลอร์ ตระกูล 8051 โดยทำการออกแบบโปรแกรม และเขียนลงใน editor จากนั้นก็จะทำการคอมไพล์ สำหรับโปรแกรมในโครงงานนี้ใช้ CROSS 32 เป็นตัวคอมไพเลอร์ โดยเปลี่ยนจากภาษาแอสเซมบลี้ให้เป็นเลขฐาน 16 ในการใช้คอมไพเลอร์ CROSS 32 นั้น ไม่เหมาะสมอย่างยิ่งที่จะใช้งาน เนื่องจากมีความไม่สะดวกในการกำหนด(define) ตัวแปรต่างๆที่ใช้งาน และมี syntax ที่ไม่สะดวกในการใช้งาน จึงสมควรที่จะเปลี่ยนไปใช้คอมไพเลอร์ชนิดอื่นแทน

เอกสารอ้างอิง

[1] บริษัท ETT จำกัด "MCS-51 Microcontrollers": บริษัท ETT จำกัด, 1989

[2] บริษัท ETT จำกัด "Dot Matrix LCD Module": บริษัท ETT จำกัด, 1989

[3] ถวิล พึ่งมา และ มนูญ สุขเกษม "โครงการพัฒนาอุปกรณ์โทรคมนาคม เครื่องช่วยบันทึกตอบรับและแจ้งภัยทางโทรศัพท์" กรุงเทพฯ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหาร ลาดกระบัง 2531

[4] สุเจตน์ จันทรังษ์ "ไมโครคอนโทรลเลอร์ชิพเดียว 8051": สำนักพิมพ์วิทยาลัยมหานคร 2535

PRELIMINARY

8031AH/8051AH
SINGLE-COMPONENT 8-BIT MICROCOMPUTER

- 8031AH — Control-Oriented CPU with RAM and I/O
- 8051AH — An 8031AH with Factory Mask-Programmable ROM
- Fabricated with Intel's HMOS II Process

- 4K × 8 ROM (8051AH only)
- 128 × 8 RAM
- 32 I/O Lines (Four 8-Bit Ports)
- Two 16-Bit Timer/Counters
- Programmable Full-Duplex Serial Channel
- 128K Accessible External Memory
- Boolean Processor
- 4 μs Multiply and Divide
- 218 User Bit-Addressable Locations
- 100 mA Typical Supply Current

The 8031AH/8051AH is Intel's HMOS II version of the high performance 8-bit 8031/8051 microcomputer. While the 8031AH/8051AH features the same powerful architecture and instruction set as its HMOS I predecessor, it offers the additional benefit of lower power supply current.

The 8031AH/8051AH provides a cost-effective solution for those controller applications requiring up to 64K bytes of program and/or 64K bytes of data storage. Specifically, the 8031AH contains 128 bytes of read/write data memory; 32 I/O lines configured as four 8-bit parallel ports; two 16-bit timer/counters; a five-source, two-priority level, nested interrupt structure; a programmable serial I/O port; and an on-chip oscillator with clock circuitry. The 8051AH has all of these 8031AH features plus 4K bytes of nonvolatile read only program memory. Both microcomputers can use standard TTL compatible memories and most byte-oriented MCS®-80 and MCS-85 peripherals for additional I/O and memory capabilities.

The 8031AH/8051AH microcomputer, characteristic of the entire MCS-51 family, is efficient in both control and computational type applications. This results from extensive BCD/binary arithmetic and bit-handling facilities. The 8031AH/8051AH also makes efficient use of its program memory space with an instruction set consisting of 44% one-byte, 41% two-byte, and 15% three-byte instructions. At 12MHz CPU operation, over half of the instructions execute in just 1.0μs, while the longest instructions, multiply and divide, require only 4μs.

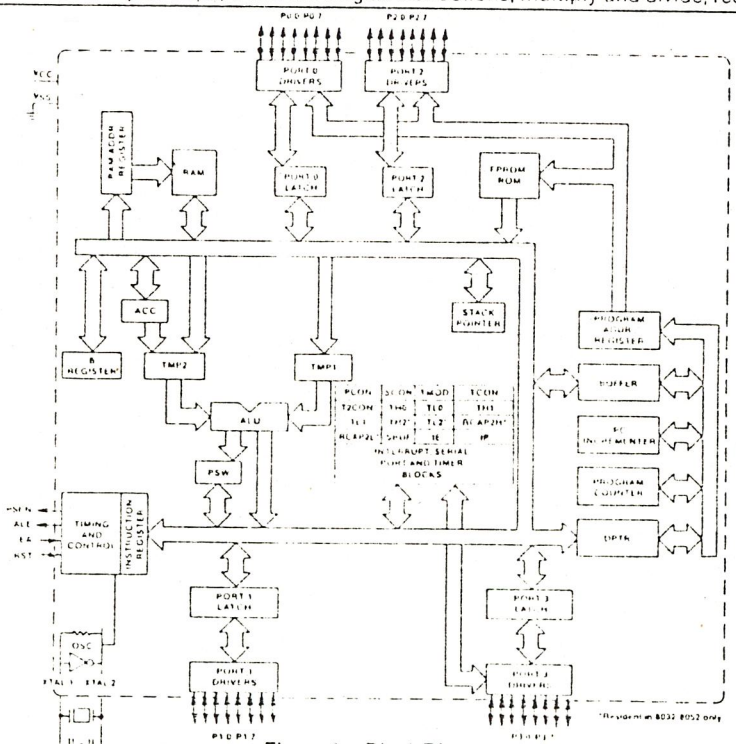


Figure 1. Block Diagram

8031AH/8051AH PIN DESCRIPTIONS

V_{SS}
Circuit ground potential.

V_{CC}
5V power supply input for normal operation and program verification.

Port 0
Port 0 is an 8-bit open drain bidirectional I/O port. It is also the multiplexed low-order address and data bus when using external memory. It is used for data output during program verification. Port 0 can sink/source eight LS TTL loads.

Port 1
Port 1 is an 8-bit quasi-bidirectional I/O port. It is also used for the low-order address byte during program verification. Port 1 can sink/source four LS TTL loads.

Port 2
Port 2 is an 8-bit quasi-bidirectional I/O port. It also emits the high-order address byte when accessing external memory. It is used for the high-order address and the control signals during program verification. Port 2 can sink/source four LS TTL loads.

Port 3
Port 3 is an 8-bit bidirectional I/O port with internal pullups. It also serves the functions of various special features of the MCS-51 Family, as listed below:

Port Pin	Alternate Function
P3.0	RXD (serial input port)
P3.1	$\overline{TXD}$ (serial output port)
P3.2	$\overline{INT0}$ (external interrupt)
P3.3	$\overline{INT1}$ (external interrupt)
P3.4	T0 (Timer/counter 0 external input)
P3.5	T1 (Timer/counter 1 external input)
P3.6	$\overline{WR}$ (external Data Memory write strobe)
P3.7	$\overline{RD}$ (external Data Memory read strobe)

The output latch corresponding to a secondary function must be programmed to a one (1) for that function to operate. Port 3 can sink/source four LS TTL loads.

RST/V_{PD}
A high on this pin for two machine cycles while the oscillator is running resets the device. A small external pulldown resistor (≈8.2kΩ) from RST/VPD to V_{SS} permits power-on reset when a capacitor (≈10 μf) is also connected from this pin to V_{CC}.

If RST/V_{PD} is held within the V_{PD} spec while V_{CC} drops below its spec, RST/V_{PD} will provide standby power to the RAM. When RST/V_{PD} is low, the RAM's current is drawn from V_{CC}.

ALE
Address Latch Enable output for latching the low byte of the address during accesses to external memory. ALE is activated at a constant rate of 1/6 the oscillator frequency except during an external data memory access at which time one ALE pulse is skipped. ALE can sink/source 8 LS TTL inputs.

$\overline{PSEN}$
The Program Store Enable output is a control signal that enables the external Program Memory to the bus during external fetch operations. It is activated every six oscillator periods except during external data memory access. $\overline{PSEN}$ remains high during internal program execution.

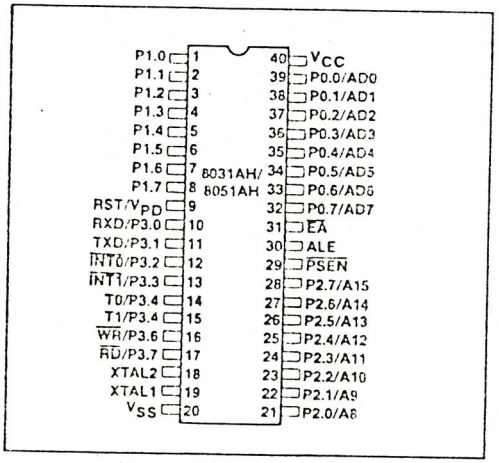


Figure 2. Pin Configuration

$\overline{EA}$

When held at a TTL high level, the 8051AH executes instructions from the internal ROM when the PC is less than 4096. When held at a TTL low level, the 8031AH/8051AH fetches all instructions from external Program Memory. Do not float $\overline{EA}$ during normal operation.

XTAL1

Input to the inverting amplifier that forms part of the oscillator. This pin should be connected to ground when an external oscillator is used.

XTAL2

Output of the inverting amplifier that forms part of the oscillator, and input to the internal clock generator. XTAL2 receives the oscillator signal when an external oscillator is used.

ABSOLUTE MAXIMUM RATINGS*

Ambient Temperature Under Bias . . . 0°C to 70°C

Storage Temperature . . . . . -65°C to +150°C

Voltage on Any Pin With Respect to Ground (V_{SS}) . . . . . -0.5V to +7V

Power Dissipation . . . . . 1 Watt

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

DC CHARACTERISTICS (T_A = 0°C to 70°C; V_{CC} = 4.5V to 5.5V; V_{SS} = 0V)

Symbol	Parameter	Min	Max	Unit	Test Conditions
VIL	Input Low Voltage	-0.5	0.8	V	
VIH	Input High Voltage (Except RST/V_{PD} and XTAL2)	2.0	V_{CC} + 0.5	V	
VIH1	Input High Voltage to RST/V_{PD} For Reset, XTAL2	2.5	V_{CC} + 0.5	V	XTAL1 to V_{SS}
V_{PD}	Power Down Voltage to RST/V_{PD}	4.5	5.5	V	V_{CC} = 0V
VOL	Output Low Voltage Ports 1, 2, 3 (Note 1)		0.45	V	IOL = 1.6mA
VOL1	Output Low Voltage Port 0, ALE, $\overline{PSEN}$ (Note 1)		0.45	V	IOL = 3.2mA
VOH	Output High Voltage Ports 1, 2, 3	2.4		V	IOH = -80μA
VOH1	Output High Voltage Port 0, ALE, $\overline{PSEN}$	2.4		V	IOH = -400μA
IIL	Logical 0 Input Current Ports 1, 2, 3		-800	μA	Vin = 0.45V
IIL2	Logical 0 Input Current for XTAL 2		-2.5	mA	XTAL1 = V_{SS}, Vin = 0.45V
ILI	Input Leakage Current To Port 0, $\overline{EA}$		±10	μA	0.45V < Vin < V_{CC}
IIH1	Input High Current to RST/V_{PD} For Reset		500	μA	Vin < V_{CC} - 1.5V
ICC	Power Supply Current		125	mA	All outputs disconnected
IPD	Power Down Current		10	mA	V_{CC} = 0V
CIO	Capacitance of I/O Buffer		10	pF	f_C = 1MHz, T_A = 25°C

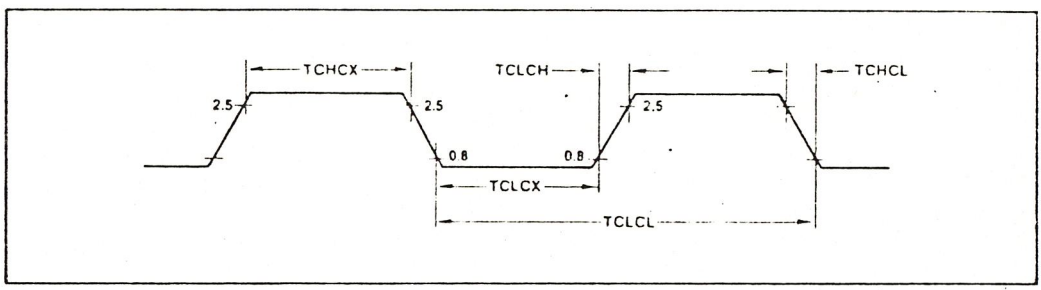
See page 4 for Notes

Note 1: VOL is degraded when the 8031AH/8051AH rapidly discharges external capacitance. This AC noise is most pronounced during emission of address data. When using external memory, locate the latch or buffer as close to the 8031AH/8051AH as possible.

Datum	Emitting Ports	Degraded I/O Lines	VOL (peak) (max)
Address	P2, P0	P1, P3	0.8V
Write Data	P0	P1, P3, ALE	0.8V

EXTERNAL CLOCK DRIVE CHARACTERISTICS (XTAL2)

Symbol	Parameter	Variable Clock freq = 1.2 MHz to 12 MHz		Unit
		Min	Max	
TCLCL	Oscillator Period	83.3	833.3	ns
TCHCX	High Time	20		ns
TCLCX	Low Time	20		ns
TCLCH	Rise Time		20	ns
TCHCL	Fall Time		20	ns



AC CHARACTERISTICS (T_A = 0°C to 70°C, V_{CC} = 5V ± 10%, V_{SS} = 0V, C_L for Port 0, ALE and $\overline{PSEN}$ Outputs = 100 pF; C_L for all other outputs = 80 pF)

EXTERNAL PROGRAM MEMORY CHARACTERISTICS

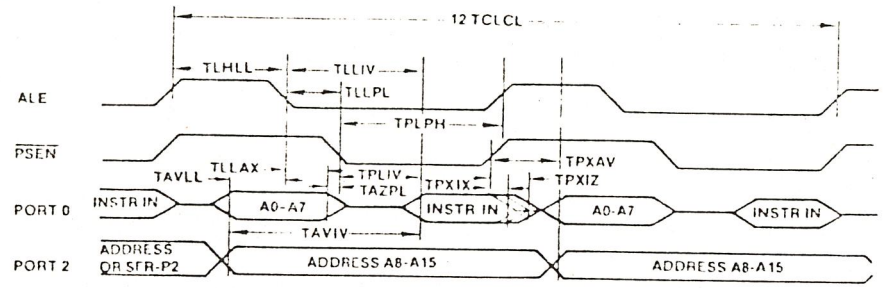
Symbol	Parameter	12 MHz Clock			Variable Clock 1/TCLCL = 1.2 MHz to 12 MHz		
		Min	Max	Unit	Min	Max	Unit
TLHLL	ALE Pulse Width	127		ns	2TCLCL-40		ns
TAVLL	Address Setup to ALE	43		ns	TCLCL-40		ns
TLLAX	Address Hold After ALE	48		ns	TCLCL-35		ns
TLLIV	ALE to Valid Instr In		233	ns		4TCLCL-100	ns
TLLPL	ALE To $\overline{PSEN}$	58		ns	TCLCL-25		ns
TPLPH	$\overline{PSEN}$ Pulse Width	215		ns	3TCLCL-35		ns
TPLIV	$\overline{PSEN}$ To Valid Instr In		125	ns		3TCLCL-125	ns
TPXIX	Input Instr Hold After $\overline{PSEN}$	0		ns	0		ns
TPXIZ	Input Instr Float After $\overline{PSEN}$		63	ns		TCLCL-20	ns
TPXAV	Address Valid After $\overline{PSEN}$	75		ns	TCLCL-8		ns
TAVIV	Address To Valid Instr In		302	ns		5TCLCL-115	ns
TAZPL	Address Float To $\overline{PSEN}$	0		ns	0		ns

EXTERNAL DATA MEMORY CHARACTERISTICS

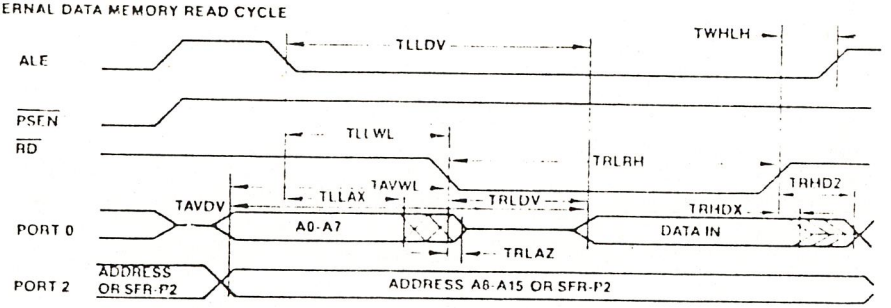
Symbol	Parameter	12 MHz Clock			Variable Clock 1/TCLCL = 1.2 MHz to 12 MHz		
		Min	Max	Unit	Min	Max	Unit
TRLRH	$\overline{RD}$ Pulse Width	400		ns	6TCLCL-100		ns
TWLWH	$\overline{WR}$ Pulse Width	400		ns	6TCLCL-100		ns
TLLAX	Address Hold After ALE	48		ns	TCLCL-35		
TRLDV	$\overline{RD}$ To Valid Data In		250	ns		5TCLCL-165	ns
TRHDX	Data Hold After $\overline{RD}$	0		ns	0		ns
TRHDZ	Data Float After $\overline{RD}$		97	ns		2TCLCL-70	ns
TLLDV	ALE To Valid Data In		517	ns		8TCLCL-150	ns
TAVDV	Address To Valid Data In		585	ns		9TCLCL-165	ns
TLLWL	ALE To $\overline{WR}$ or $\overline{RD}$	200	300	ns	3TCLCL-50	3TCLCL + 50	ns
TAVWL	Address To $\overline{WR}$ or $\overline{RD}$	203		ns	4TCLCL-130		ns
TWHLH	$\overline{WR}$ or $\overline{RD}$ High To ALE High	43	123	ns	TCLCL-40	TCLCL + 40	ns
TDVWX	Data Valid To $\overline{WR}$ Transition	23		ns	TCLCL-60		ns
TQVWH	Data Setup Before $\overline{WR}$	433		ns	7TCLCL-150		ns
TWHQX	Data Hold After $\overline{WR}$	33		ns	TCLCL-50		ns
TRLAZ	Address Float After $\overline{RD}$		0	ns		0	ns

AC TIMING DIAGRAMS

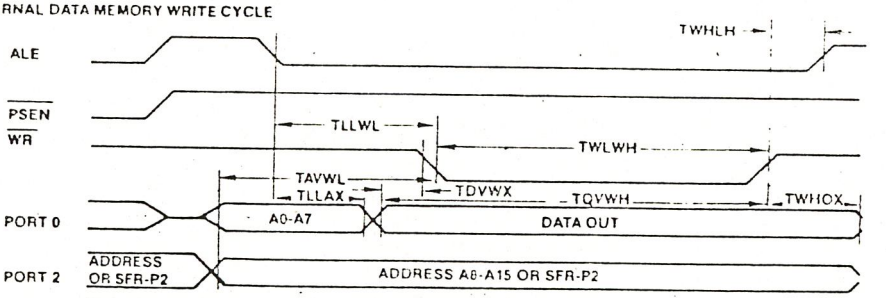
EXTERNAL PROGRAM MEMORY READ CYCLE



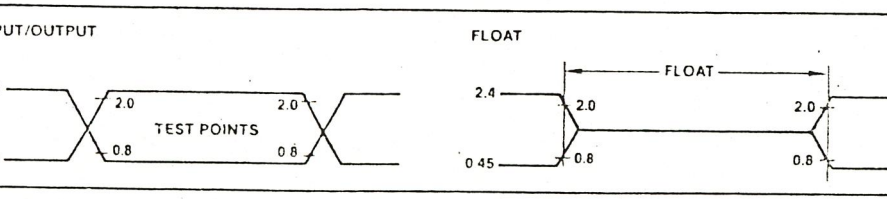
EXTERNAL DATA MEMORY READ CYCLE



EXTERNAL DATA MEMORY WRITE CYCLE

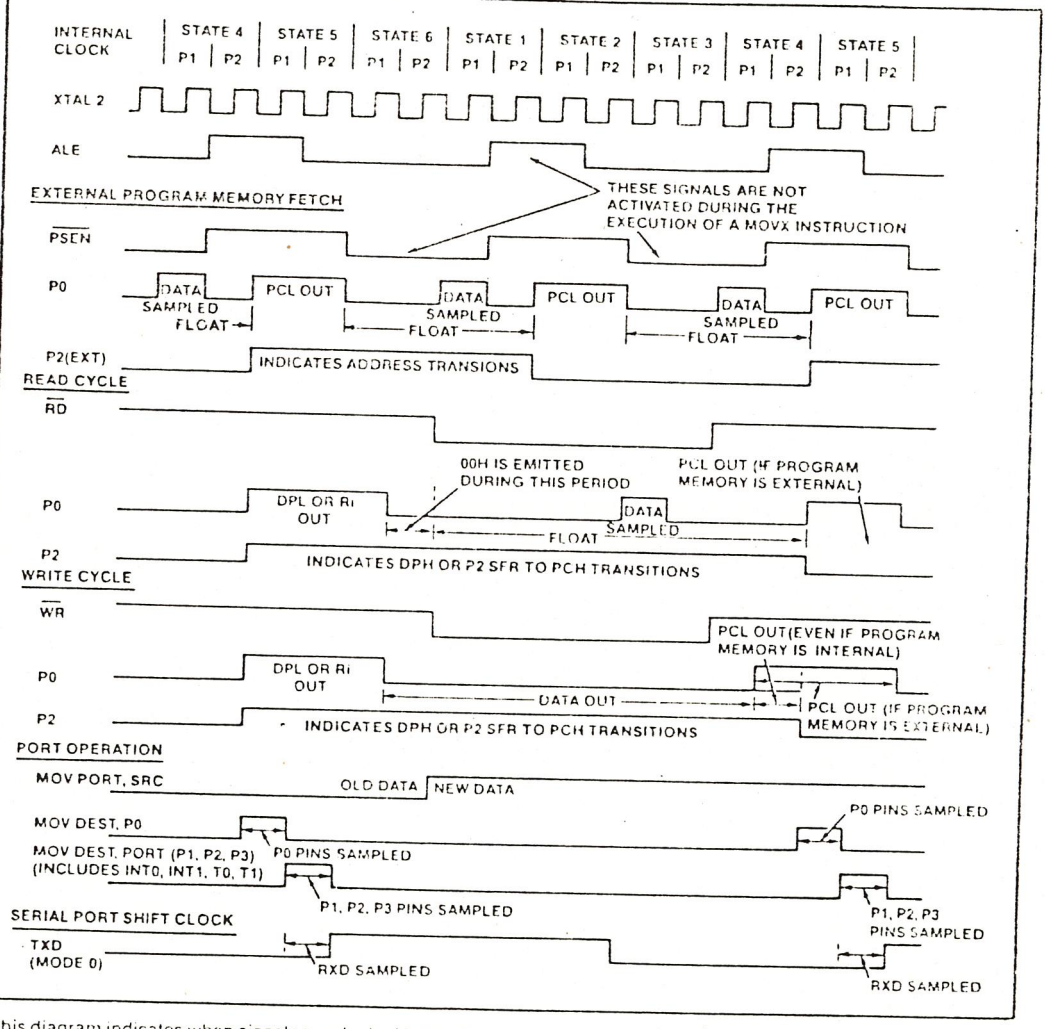


AC TESTING INPUT/OUTPUT, FLOAT WAVEFORMS



AC inputs during testing are driven at 2.4V for a logic "1" and 0.45V for a logic "0". Timing measurements are made at 2.0V for a logic "1" and 0.8V for a logic "0". For timing purposes, the float state is defined as the point at which a P0 pin sinks 2.4mA or sources 400 μA at the voltage test levels.

CLOCK WAVEFORMS



This diagram indicates when signals are clocked internally. The time it takes the signals to propagate to the pins, however, ranges from 25 to 125 ns. This propagation delay is dependent on variables such as temperature and pin loading. Propagation also varies from output to output and component to component. Typically though, (T_A = 25°C, fully loaded) RD and WR propagation delays are approximately 50 ns. The other signals are typically 85 ns. Propagation delays are incorporated in the AC specifications.

Table 1. MCS®-51 Instruction Set Description

ARITHMETIC OPERATIONS

Mnemonic		Description	Byte	Cyc
ADD	A,Rn	Add register to Accumulator	1	1
ADD	A,direct	Add direct byte to Accumulator	2	1
ADD	A,@Ri	Add indirect RAM to Accumulator	1	1
ADD	A,#data	Add immediate data to Accumulator	2	1
ADDC	A,Rn	Add register to Accumulator with Carry	1	1
ADDC	A,direct	Add direct byte to A with Carry flag	2	1
ADDC	A,@Ri	Add indirect RAM to A with Carry flag	1	1
ADDC	A,#data	Add immediate data to A with Carry flag	2	1
SUBB	A,Rn	Subtract register from A with Borrow	1	1
SUBB	A,direct	Subtract direct byte from A with Borrow	2	1
SUBB	A,@Ri	Subtract indirect RAM from A with Borrow	1	1
SUBB	A,#data	Subtract immed data from A with Borrow	2	1
INC	A	Increment Accumulator	1	1
INC	Rn	Increment register	1	1
INC	direct	Increment direct byte	2	1
INC	@Ri	Increment indirect RAM	1	1
INC	DPTR	Increment Data Pointer	1	2
DEC	A	Decrement Accumulator	1	1
DEC	Rn	Decrement register	1	1
DEC	direct	Decrement direct byte	2	1
DEC	@Ri	Decrement indirect RAM	1	1
MUL	AB	Multiply A & B	1	4
DIV	AB	Divide A by B	1	4
DA	A	Decimal Adjust Accumulator	1	1

LOGICAL OPERATIONS

Mnemonic		Destination	Byte	Cyc
ANL	A,Rn	AND register to Accumulator	1	1
ANL	A,direct	AND direct byte to Accumulator	2	1
ANL	A,@Ri	AND indirect RAM to Accumulator	1	1
ANL	A,#data	AND immediate data to Accumulator	2	1
ANL	direct,A	AND Accumulator to direct byte	2	1
ANL	direct,#data	AND immediate data to direct byte	3	2
ORL	A,Rn	OR register to Accumulator	1	1
ORL	A,direct	OR direct byte to Accumulator	2	1

LOGICAL OPERATIONS (CONTINUED)

Mnemonic		Destination	Byte	Cyc
ORL	A,@Ri	OR indirect RAM to Accumulator	1	1
ORL	A,#data	OR immediate data to Accumulator	2	1
ORL	direct,A	OR Accumulator to direct byte	2	1
ORL	direct,#data	OR immediate data to direct byte	3	2
XRL	A,Rn	Exclusive-OR register to Accumulator	1	1
XRL	A,direct	Exclusive-OR direct byte to Accumulator	2	1
XRL	A,@Ri	Exclusive-OR indirect RAM to A	1	1
XRL	A,#data	Exclusive-OR immediate data to A	2	1
XRL	direct,A	Exclusive-OR Accumulator to direct byte	2	1
XRL	direct,#data	Exclusive-OR immediate data to direct	3	2
CLR	A	Clear Accumulator	1	1
CPL	A	Complement Accumulator	1	1
RL	A	Rotate Accumulator Left	1	1
RLC	A	Rotate A Left through the Carry flag	1	1
RR	A	Rotate Accumulator Right	1	1
RRC	A	Rotate A Right through Carry flag	1	1
SWAP	A	Swap nibbles within the Accumulator	1	1

DATA TRANSFER

Mnemonic		Description	Byte	Cyc
MOV	A,Rn	Move register to Accumulator	1	1
MOV	A,direct	Move direct byte to Accumulator	2	1
MOV	A,@Ri	Move indirect RAM to Accumulator	1	1
MOV	A,#data	Mov immediate data to Accumulator	2	1
MOV	Rn,A	Move Accumulator to register	1	1
MOV	Rn,direct	Move direct byte to register	2	2
MOV	Rn,#data	Move immediate data to register	2	1
MOV	direct,A	Move Accumulator to direct byte	2	1
MOV	direct,Rn	Move register to direct byte	2	2
MOV	direct,direct	Move direct byte to direct	3	2
MOV	direct,@Ri	Move indirect RAM to direct byte	2	2

Table 1. (Cont.)

DATA TRANSFER (CONTINUED)

Mnemonic		Description	Byte	Cyc
MOV	direct,#data	Move immediate data to direct byte	3	2
MOV	@Ri,A	Move Accumulator to indirect RAM	1	1
MOV	@Ri,direct	Move direct byte to indirect RAM	2	2
MOV	@Ri,#data	Move immediate data to indirect RAM	2	1
MOV	DPTR,#data16	Load Data Pointer with a 16-bit constant	3	2
MOVC	A,@A+DPTR	Move Code byte relative to DPTR to A	1	2
MOVC	A,@A+PC	Move Code byte relative to PC to A	1	2
MOVX	A,@Ri	Move External RAM (8-bit addr) to A	1	2
MOVX	A,@DPTR	Move External RAM (16-bit addr) to A	1	2
MOVX	@Ri,A	Move A to External RAM (8-bit addr)	1	2
MOVX	@DPTR,A	Move A to External RAM (16-bit addr)	1	2
PUSH	direct	Push direct byte onto stack	2	2
POP	direct	Pop direct byte from stack	2	2
XCH	A,Rn	Exchange register with Accumulator	1	1
XCH	A,direct	Exchange direct byte with Accumulator	2	1
XCH	A,@Ri	Exchange indirect RAM with A	1	1
XCHD	A,@Ri	Exchange low-order Digit ind RAM w A	1	1

BOOLEAN VARIABLE MANIPULATION

Mnemonic		Description	Byte	Cyc
CLR	C	Clear Carry flag	1	1
CLR	bit	Clear direct bit	2	1
SETB	C	Set Carry flag	1	1
SETB	bit	Set direct Bit	2	1
CPL	C	Complement Carry flag	1	1
CPL	bit	Complement direct bit	2	1
ANL	C,bit	AND direct bit to Carry flag	2	2
ANL	C,1 bit	AND complement of direct bit to Carry	2	2
ORL	C/bit	OR direct bit to Carry flag	2	2
ORL	C,1 bit	OR complement of direct bit to Carry	2	2
MOV	C/bit	Move direct bit to Carry flag	2	1
MOV	bit,C	Move Carry flag to direct bit	2	2

PROGRAM AND MACHINE CONTROL

Mnemonic		Description	Byte	Cyc
ACALL	addr11	Absolute Subroutine Call	2	2
LCALL	addr16	Long Subroutine Call	3	2
RET		Return from subroutine	1	2
RETI		Return from interrupt	1	2
AJMP	addr11	Absolute Jump	2	2
LJMP	addr16	Long Jump	3	2
SJMP	rel	Short Jump (relative addr)	2	2
JMP	@A+DPTR	Jump indirect relative to the DPTR	1	2
JZ	rel	Jump if Accumulator is Zero	2	2
JNZ	rel	Jump if Accumulator is Not Zero	2	2
JC	rel	Jump if Carry flag is set	2	2
JNC	rel	Jump if No Carry flag	2	2
JB	bit,rel	Jump if direct Bit set	3	2
JNB	bit,rel	Jump if direct Bit Not set	3	2
JBC	bit,rel	Jump if direct Bit is set & Clear bit	3	2
CJNE	A,direct rel	Compare direct to A & Jump if Not Equal	3	2
CJNE	A,#data rel	Comp. immed. to A & Jump if Not Equal	3	2
CJNE	Rn,#data rel	Comp. immed. to reg & Jump if Not Equal	3	2
CJNE	@Ri,#data,rel	Comp. immed. to ind. & Jump if Not Equal	3	2
DJNZ	Rn,rel	Decrement register & Jump if Not Zero	2	2
DJNZ	direct,rel	Decrement direct & Jump if Not Zero	3	2
NOP		No operation	1	1

Notes on data addressing modes:

Rn —Working register R0-R7
direct —128 internal RAM locations, any I/O port, control or status register
@Ri —Indirect internal RAM location addressed by register R0 or R1
#data —8-bit constant included in instruction
#data16 —16-bit constant included as bytes 2 & 3 of instruction
bit —128 software flags, any I/O pin, control or status bit

Notes on program addressing modes:

addr16 —Destination address for LCALL & LJMP may be anywhere within the 64-K program memory address space
Addr11 —Destination address for ACALL & AJMP will be within the same 2-K page of program memory as the first byte of the following instruction
rel —SJMP and all conditional jumps include an 8-bit offset byte. Range is +127-128 bytes relative to first byte of the following instruction

Table 2. Instruction Opcodes in Hexadecimal Order

Hex Code	Number of Bytes	Mnemonic	Operands
00	1	NOP	
01	2	AJMP	code addr
02	3	LJMP	code addr
03	1	RR	A
04	1	INC	A
05	2	INC	data addr
06	1	INC	@R0
07	1	INC	@R1
08	1	INC	R0
09	1	INC	R1
0A	1	INC	R2
0B	1	INC	R3
0C	1	INC	R4
0D	1	INC	R5
0E	1	INC	R6
0F	1	INC	R7
10	3	JBC	bit addr, code addr
11	2	ACALL	code addr
12	3	LCALL	code addr
13	1	RRC	A
14	1	DEC	A
15	2	DEC	data addr
16	1	DEC	@R0
17	1	DEC	@R1
18	1	DEC	R0
19	1	DEC	R1
1A	1	DEC	R2
1B	1	DEC	R3
1C	1	DEC	R4
1D	1	DEC	R5
1E	1	DEC	R6
1F	1	DEC	R7
20	3	JB	bit addr, code addr
21	2	AJMP	code addr
22	1	RET	
23	1	RL	A
24	2	ADD	A,#data
25	2	ADD	A,data addr
26	1	ADD	A,@R0
27	1	ADD	A,@R1
28	1	ADD	A,R0
29	1	ADD	A,R1
2A	1	ADD	A,R2
2B	1	ADD	A,R3
2C	1	ADD	A,R4
2D	1	ADD	A,R5
2E	1	ADD	A,R6
2F	1	ADD	A,R7
30	3	JNB	bit addr, code addr
31	2	ACALL	code addr
32	1	RETI	
33	1	RLC	A
34	2	ADDC	A,#data
35	2	ADDC	A,data addr
36	1	ADDC	A,@R0
37	1	ADDC	A,@R1
38	1	ADDC	A,R0
39	1	ADDC	A,R1
3A	1	ADDC	A,R2
3B	1	ADDC	A,R3
3C	1	ADDC	A,R4
3D	1	ADDC	A,R5
3E	1	ADDC	A,R6
3F	1	ADDC	A,R7
40	2	JC	code addr
41	2	AJMP	code addr
42	2	ORL	data addr,A
43	3	ORL	data addr,#data
44	2	ORL	A,#data
45	2	ORL	A,data addr
46	1	ORL	A,@R0
47	1	ORL	A,@R1
48	1	ORL	A,R0
49	1	ORL	A,R1
4A	1	ORL	A,R2
4B	1	ORL	A,R3
4C	1	ORL	A,R4
4D	1	ORL	A,R5
4E	1	ORL	A,R6
4F	1	ORL	A,R7
50	2	JNC	code addr
51	2	ACALL	code addr
52	2	ANL	data addr,A
53	3	ANL	data addr,#data
54	2	ANL	A,#data
55	2	ANL	A,data addr
56	1	ANL	A,@R0
57	1	ANL	A@R1
58	1	ANL	A,R0
59	1	ANL	A,R1
5A	1	ANL	A,R2
5B	1	ANL	A,R3
5C	1	ANL	A,R4
5D	1	ANL	A,R5
5E	1	ANL	A,R6
5F	1	ANL	A,R7
60	2	JZ	code addr
61	2	AJMP	code addr
62	2	XRL	data addr,A
63	3	XRL	data addr,#data
64	2	XRL	A,#data
65	2	XRL	A,data addr

Table 2. (Cont.)

Hex Code	Number of Bytes	Mnemonic	Operands
66	1	XRL	A,@R0
67	1	XRL	A,@R1
68	1	XRL	A,R0
69	1	XRL	A,R1
6A	1	XRL	A,R2
6B	1	XRL	A,R3
6C	1	XRL	A,R4
6D	1	XRL	A,R5
6E	1	XRL	A,R6
6F	1	XRL	A,R7
70	2	JNZ	code addr
71	2	ACALL	code addr
72	2	ORL	C,bit addr
73	1	JMP	@A+DPTR
74	2	MOV	A,#data
75	3	MOV	data addr,#data
76	2	MOV	@R0,#data
77	2	MOV	@R1,#data
78	2	MOV	R0,#data
79	2	MOV	R1,#data
7A	2	MOV	R2,#data
7B	2	MOV	R3,#data
7C	2	MOV	R4,#data
7D	2	MOV	R5,#data
7E	2	MOV	R6,#data
7F	2	MOV	R7,#data
80	2	SJMP	code addr
81	2	AJMP	code addr
82	2	ANL	C,bit addr
83	1	MOVC	A,@A+PC
84	1	DIV	AB
85	3	MOV	data addr,data addr
86	2	MOV	data addr,@R0
87	2	MOV	data addr,@R1
88	2	MOV	data addr,R0
89	2	MOV	data addr,R1
8A	2	MOV	data addr,R2
8B	2	MOV	data addr,R3
8C	2	MOV	data addr,R4
8D	2	MOV	data addr,R5
8E	2	MOV	data addr,R6
8F	2	MOV	data addr,R7
90	3	MOV	DPTR,#data
91	2	ACALL	code addr
92	2	MOV	bit addr,C
93	1	MOVC	A,@A+DPTR
94	2	SUBB	A,#data
95	2	SUBB	A,data addr
96	1	SUBB	A,@R0
97	1	SUBB	A,@R1
98	1	SUBB	A,R0

Hex Code	Number of Bytes	Mnemonic	Operands
99	1	SUBB	A,R1
9A	1	SUBB	A,R2
9B	1	SUBB	A,R3
9C	1	SUBB	A,R4
9D	1	SUBB	A,R5
9E	1	SUBB	A,R6
9F	1	SUBB	A,R7
A0	2	ORL	C,/bit addr
A1	2	AJMP	code addr
A2	2	MOV	C,bit addr
A3	1	INC	DPTR
A4	1	MUL	AB
A5		reserved	
A6	2	MOV	@R0, data addr
A7	2	MOV	@R1,data addr
A8	2	MOV	R0,data addr
A9	2	MOV	R1,data addr
AA	2	MOV	R2,data addr
AB	2	MOV	R3,data addr
AC	2	MOV	R4,data addr
AD	2	MOV	R5,data addr
AE	2	MOV	R6,data addr
AF	2	MOV	R7,data addr
B0	2	ANL	C,/bit addr
B1	2	ACALL	code addr
B2	2	CPL	bit addr
B3	1	CPL	C
B4	3	CJNE	A,#data,code addr
B5	3	CJNE	A,data addr,code addr
B6	3	CJNE	@R0,#data code addr
B7	3	CJNE	@R1,#data code addr
B8	3	CJNE	R0,#data,code addr
B9	3	CJNE	R1,#data,code addr
BA	3	CJNE	R2,#data,code addr
BB	3	CJNE	R3,#data,code addr
BC	3	CJNE	R4,#data,code addr
BD	3	CJNE	R5,#data,code addr
BE	3	CJNE	R6,#data,code addr
BF	3	CJNE	R7,#data,code addr
C0	2	PUSH	data addr
C1	2	AJMP	code addr
C2	2	CLR	bit addr
C3	1	CLR	C
C4	1	SWAP	A
C5	2	XCH	A,data addr
C6	1	XCH	A,@R0
C7	1	XCH	A,@R1
C8	1	XCH	A,R0
C9	1	XCH	A,R1
CA	1	XCH	A,R2
CB	1	XCH	A,R3