การควบคุมเครื่องเสียงด้วยคอมพิวเตอร์

COMPUTER-CONTROL SOUND SYSTEM

โดย

นาย เกรียงไกร ลีละศุภสกุล เลขประจำตัว 37013187

นาย คเชนทร์ ทิพย์สุคนธร เลขประจำตัว 37013190

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตร์บัณฑิต

สาขาวิชาอิเล็กทรอนิกส์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2539

การควบคุมเครื่องเสียงด้วยคอมพิวเตอร์

COMPUTER-CONTROL SOUND SYSTEM

โดย

นาย เกรียงไกร ลีละศุภสกุล เลขประจำตัว 37013187
นาย คเชนทร์ ทิพย์สุคนธร เลขประจำตัว 37013190

อาจารย์ที่ปรึกษา

ดร. สมศักดิ์ ชุ่มช่วย

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตร์บัณฑิต
สาขาวิชาอิเล็กทรอนิกส์
คณะวิศวกรรมศาสตร์
สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง
ปีการศึกษา 2539

ปริญญานิพนธ์ ปีการศึกษา 2539

ภาควิชา อิเล็กทรอนิกส์

คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

เรื่อง การควบคุมเครื่องเสียงด้วยคอมพิวเตอร์ (Computer Control Sound System)

ผู้จัดทำ นาย เกรียงไกร ลีละศุภสกุล เลขประจำตัว 37013187

นาย คเชนทร์ ทิพย์สุคนธร เลขประจำตัว 37013190

.....อาจารย์ที่ปรึกษา

(ดร.สมศักดิ์ ชุ้มช่วย)

การควบคุมเครื่องเสียงด้วยคอมพิวเตอร์

Computer Control Sound System

ผู้จัดทำ นาย เกรียงไกร ลีละศุภสกุล เลขประจำตัว 37013187

นาย คเชนทร์ ทิพย์สุคนธร เลขประจำตัว 37013190

โครงงานนี้ได้รับการตรวจสอบแล้ว พร้อมที่จะทำการสอบได้

.....อาจารย์ที่ปรึกษา

(ดร.สมศักดิ์ ชุ่มช่วย)

การควบคุมเครื่องเสียงด้วยคอมพิวเตอร์

(Computer Control Sound System)

นายเกรียงไกร ลีละศุภสกุล
นายคเชนทร์ ทิพย์สุคนธร
ดร. สมศักดิ์ ชุมช่วย อาจารย์ที่ปรึกษา
ปีการศึกษา 2539

บทคัดย่อ

ปริญญานิพนธ์ฉบับนี้มีรายละเอียดเกี่ยวกับระบบการควบคุมเครื่องเสียงด้วยคอมพิวเตอร์ ซึ่งระบบนี้ประกอบไปด้วยจูนเนอร์และอีควอไลเซอร์ และใช้คอมพิวเตอร์ส่วนบุคคลทำหน้าที่เป็นศูนย์กลางการติดต่อระหว่างอุปกรณ์เครื่องเสียงต่างๆ โดยผ่านไมโครโพเซสเซอร์ (MCS-51) และระบบนี้โปรแกรมประยุกต์ที่ใช้เป็นเครื่องมือในการพัฒนาระบบ คือ โปรแกรม เดลไฟล์ (DELPHI) ในส่วนของเครื่องเล่นเทปและปรีแอมป์จะอยู่ในปริญญานิพนธ์การควบคุมเครื่องเสียงด้วยคอมพิวเตอร์อีกเล่มหนึ่ง ซึ่งได้ศึกษาร่วมกัน

COMPUTER CONTROL SOUND SYSTEM

Karaingkai Leelasupaskul

Kachan Tipukontorn

Somsak Chomchuay Advisor

1997

ABSTRAT

This thesis describes the development of COMPUTER-CONTROL SOUND SYSTEM which comprises of Sound Equiment namely , tape and pre-tone amplifier . An IBM PC is used as a host computer to control thecommunication the instance between to each sound units microcontroller. Window interface package , DELPHI , is used as a devalopment tool in high level progromming. The discussion of the development of tuner and equalizer parts are involved in another thesis COMPUTER-CONTROL SOUND SYSTEM .

คำนำ

ในสมัยก่อนเราได้ยินเสียงเพลงที่มาจากเครื่องเสียงที่ใช้ระบบกลไกควบคุมการทำงานต่อมาได้มีการพัฒนาการมาเรื่อยๆจาก ระบบที่ควบคุมด้วยกลไก ระบบที่เป็นดิจิตอลพื้นฐานควบคุม ระบบที่ใช้อินฟาเรดควบคุม และในปัจจุบันคอมพิวเตอร์ได้เข้ามามีบทบาทในทุกสาขาวิชาชีพ เนื่องจากมีความสะดวกและง่ายต่อการใช้งาน เราจึงได้นำเสนอระบบที่เป็นคอมพิวเตอร์ควบคุมซึ่งเป็นระบบหนึ่งในอีกหลายๆระบบที่ใช้คอมพิวเตอร์ควบคุม โดยสั่งงานผ่านพอร์ทอนุกรมของคอมพิวเตอร์ ไปยังเครื่องเสียงซึ่งจะต้องมีคอนโทรลเลอร์บอร์ดเป็นตัวส่งผ่านข้อมูลระหว่างเครื่องเสียงกับคอมพิวเตอร์ทำให้สามารถสั่งงานผ่านคอมพิวเตอร์ได้โดยตรง

ในปริญญานิพนธ์ฉบับนี้จะกล่าวถึงเฉพาะการควบคุมเทปและปรีแอมป์ โดยในส่วนของเครื่องเล่นเทปและปรีแอมป์นี้จะต้องเป็นระบบดิจิตอลเท่านั้น หรือถ้าเป็นระบบอานาล็อคจะต้องดัดแปลงให้สามารถใช้ระบบดิจิตอลควบคุมได้เสียก่อนจึงใช้ได้กับหลักการของปริญญานิพนธ์นี้ ในส่วนของจูนเนอร์และอีควอไลเซอร์ ผู้ที่สนใจสามารถศึกษาได้จากปริญญานิพนธ์อีกเล่มหนึ่งซึ่งเป็นปริญญานิพนธ์ที่ศึกษาร่วมกัน

คณะผู้จัดทำ

สารบัญ

บทที่ 1
บทนำ

" ชนใดไม่มีดนตรีกาล ในสันดานเป็นคนชอบกลนัก " น้อยคนนักที่ไม่เคยได้ยินได้อ่านประโยคข้างต้น ซึ่งเป็นพระราชดำรัสขององค์พระบาทสมเด็จพระมงกุฎเกล้าเจ้าอยู่หัว รัชกาลที่ 6 เสียงดนตรีน่าจะเรียกได้ว่าเป็นพื้นฐานความต้องการหนึ่งของมนุษย์ ในดนตรีทุกแนวทุกประเภทมีสิ่งสำคัญพื้นฐานเหมือนกันก็คือจังหวะ มนุษย์เรารู้จักจังหวะ และคุ้นเคยโดยไม่อาจหลีกเลี่ยงได้ตั้งแต่เกิด นั่นคือจังหวะการเต้นของหัวใจ ขณะที่มีอารมณ์ตื่นเต้น ตกใจ ระทึกใจ สนุกสนาน จะสังเกตได้ว่าจังหวะการเต้นของหัวใจจะถี่มากกว่า ขณะที่มีอารมณ์สบาย ๆ จะเห็นว่าจังหวะก็สามารถบ่งบอกอารมณ์ได้อย่างคร่าว ๆ เช่นเดียวกันกับจังหวะของเพลง เพลงที่มีความสนุกสนานระทึกใจ เร้าใจ จะมีจังหวะที่เร็วและถี่กว่าเพลงที่ฟังสบาย ๆ ดังจะสังเกตได้โดยง่ายจากเพลงประกอบภาพยนต์

ในสมัยก่อนที่ยังไม่มีการบันทึกเสียงการฟังดนตรี หรือการรับฟังข่าวสารต่างๆ สามารถฟังได้จากการแสดงสดจริง ๆ เท่านั้น แต่เมื่อเทคโนโลยีถูกพัฒนาจนมีการบันทึกเสียง เช่น แผ่นเสียง, เทป และพัฒนาต่อมา เช่น CD, MD โดยพัฒนาเพื่อให้ความชัดเจนเหมือนจริงของเสียงที่ถูกบันทึกแล้วเมื่อนำกลับมาเล่นใหม่สูงขึ้น(HIGH FIDILITY)เพื่อให้สัญญาณรบกวนในการบันทึกและการนำกลับมาเล่นใหม่ลดลง เพื่อลดขนาดของเครื่องเล่นและขนาดของวัสดุที่ใช้ในการบันทึกเป็นต้น การพัฒนาอีกอย่างของการบันทึกเสียง คือการพัฒนาทางด้านมิติเสียง คือความเหมือนจริงของเสียงและทิศทางของแหล่งกำเนิดให้เหมือนขณะทำการบันทึก เช่นการบันทึกเสียงแบบ 2 ทิศทาง (2 CHANNEL ; STEREO), ระบบเสียงรอบทิศทาง (SURROUND) เป็นต้น การพัฒนามิติของเสียงยังถูกนำไปใช้ในภาพยนต์ เพื่อเพิ่มความเหมือนจริง สมจริงสมจัง และเหมือนอยู่ในเหตุการณ์เดียวกับเนื้อเรื่องภาพยนต์ในขณะนั้น เพื่อเพิ่มอรรถรสในการชมภาพยนต์ เช่น DTS, THX, SDDS เป็นต้น

รูปที่ 1.1 ส่วนประกอบหลักของระบบเสียงในบ้านทั่วไป

ในระบบเสียงต่าง ๆ มีส่วนประกอบหลักที่คล้ายกันดังไดอะแกรมดังรูป SOURCE คือแหล่งกำเนิดเสียงของระบบ เช่น เครื่องเล่นเทป เครื่องเล่น CD (COMPACK DISK) วิทยุ (TUNER) ไมโครโฟน (MICROPHONE) เป็นต้น สัญญาณจากแหล่งกำเนิดจะถูกขยายเพื่อให้มีขนาดใหญ่มากพอที่ภาคปรีแอมป์ เพื่อส่งไปยัง ภาคโทน-คอนโทรล (TONE-CONTROL) เพื่อปรับแต่งสัญญาณ เช่น เพิ่ม (BOOT) หรือลด (CUT) ความถี่ต่ำ (BASS) และ ความถี่สูง

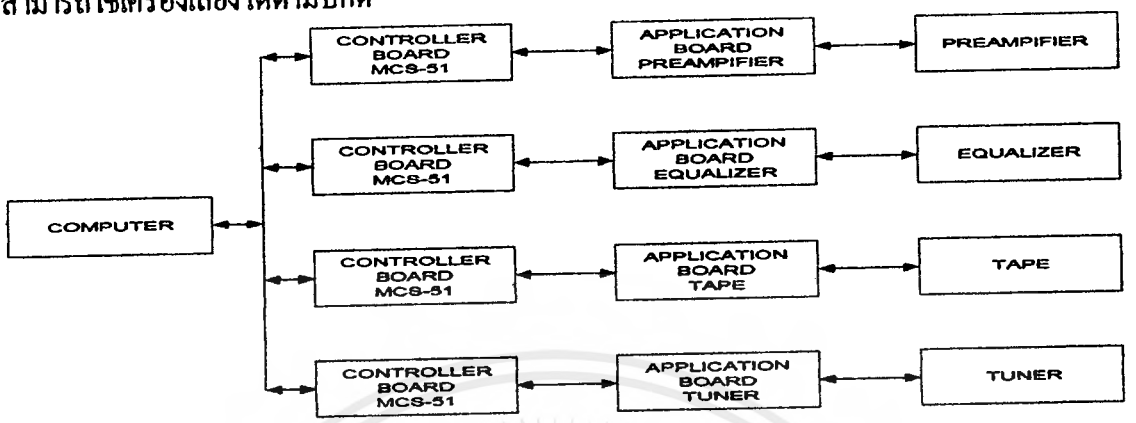
(TREBLE) หรือปรับแต่งความถี่โดยละเอียดขึ้นโดยใช้ อีควอไลเซอร์ (EQUALIZER) รวมทั้งการปรับแต่งลักษณะสัญญาณเช่น REVERBE, DELAY สัญญาณที่ถูกปรับแต่งจะถูกขยายให้มีขนาดใหญ่และมีกำลังสูงพอที่จะขับลำโพง ด้วยภาคขยาย (AMPLIFIER)

ในสมัยก่อนกลไกของเครื่องอ่านการบันทึกเสียงนั้นเริ่มตั้งแต่การใช้มือหมุนโดยตรง, ใช้การไขลาน จนมีการพัฒนามาใช้มอเตอร์ (MOTOR) เพื่อความสะดวกในการใช้งาน ทั้งยังพัฒนาให้ความเร็วรอบ ที่มีความเร็วรอบสูงพอ มีความแม่นยำ และ มีความคงที่ของรอบสูง การพัฒนาให้มีขนาดเล็กที่สุดเท่าที่จะสามารถทำงานได้ปกติ มีรูปร่างลักษณะที่น่าใช้งานและง่ายต่อการใช้งานที่สุด จากเดิมที่การควบคุมการทำงานของระบบด้วยกลไก (MACHANICS) ก็พัฒนาเป็นการควบคุมการทำงานด้วยระบบดิจิตอล (DIGITAL)ซึ่งใช้การกดปุ่มเพื่อควบคุมการทำงานทั้งหมดของระบบ จึงมีความสะดวกและใช้งานง่ายขึ้น

ในปัจจุบัน คอมพิวเตอร์ได้เข้ามามีบทบาทในทุกสาขาวิชาชีพ เนื่องจากมีความสะดวกและง่ายต่อการใช้งาน ตั้งแต่การแสดงผลซึ่งมีความละเอียด สวยงาม ความหลากหลายต่อการออกแบบการสั่งงานที่สามารถรองรับคำสั่งได้ทั้งขนาด และจำนวนที่มหาศาล และยังมีความแม่นยำสูง ทั้งยังปรับเปลี่ยนการควบคุมอุปกรณ์ภายนอกได้อย่างหลากหลายแบบ จึงเป็นที่มาของโครงงานนี้

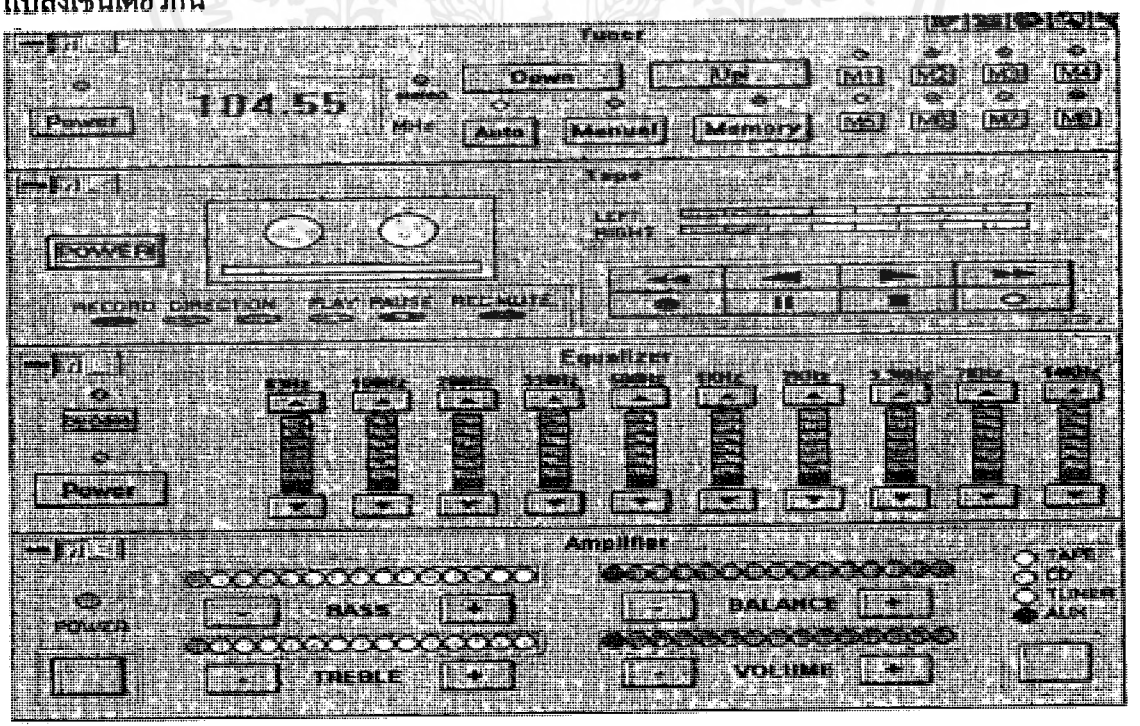
โครงงานการควบคุมเครื่องเสียงด้วยคอมพิวเตอร์ (COMPUTER CONTROL SOUND SYSTEM) นี้เป็นการควบคุมระบบเครื่องเสียงที่ใช้ตามบ้านเรือน (HOME USE) ด้วยคอมพิวเตอร์ โดยใช้เครื่องเสียงที่มีอยู่แล้วเพิ่มภาคการควบคุมการทำงาน และภาคอ่านการแสดงผลซึ่งจะไม่เข้าไปยุ่งเกี่ยวกับสัญญาณเสียงที่ทำงานอยู่เดิม จึงทำให้คุณภาพเสียงเหมือนเดิมเป็นปกติ การสั่งงานจากคอมพิวเตอร์ติดต่อผ่านพอร์ทอนุกรม (SERIES PORT) จำนวน 3 เส้น คือ สายส่ง (Tx) สายรับ (Rx) และกราวด์ (GROUND) ติดต่อกับคอนโทรลเลอร์บอร์ด (CONTROLLER BOARD) ที่มีไอซีไมโครคอนโทรลเลอร์ MCS-51 เป็นหัวใจในการทำงาน และสั่งงานผ่านพอร์ทขนาน (PARALLEL PORT) บนบอร์ดเพื่อควบคุมให้ไอซี 8255 ซึ่งเป็นพอร์ทอินพุท / พอร์ทเอาท์พุท (IN / OUT PORT) สั่งงานไปควบคุมและค่าต่าง ๆ จากอุปกรณ์ที่ต่อเพิ่มเติมในการควบคุมและการอ่านการแสดงผล ไปยัง MCS-51 เพื่อติดต่อกับคอมพิวเตอร์ บนหน้าจอของคอมพิวเตอร์จะแสดงถึงหน้าปัทม์ของอุปกรณ์เครื่องเสียงแต่ละชนิดที่ใช้งาน พร้อมทั้งตำแหน่งการควบคุม, การแสดงผลที่เหมือนกับเครื่องเสียงจริงที่ต้องการควบคุม สามารถควบคุมการทำงานของระบบเครื่องเสียงหน้าที่การทำงานผ่านคอมพิวเตอร์และดูการทำงานจริงของระบบบนจอคอมพิวเตอร์เช่นกัน ทั้งยังสามารถปรับแต่งระบบเสียงที่หน้าปัทม์ของเครื่องเสียงเอง ผลจากการปรับแต่งนี้จะมาแสดงที่จอคอมพิวเตอร์ด้วย เครื่องเสียงชุดนี้ยังสามารถใช้งานได้แม้ขณะที่

คอมพิวเตอร์ไม่ได้ทำงานอยู่ หรือก็คือ แม้จะปิดเครื่องคอมพิวเตอร์ หรือไม่ได้ต่อกับคอมพิวเตอร์ก็สามารถใช้เครื่องเสียงได้ตามปกติ



รูปที่ 1.2 บล๊อกไดอะแกรมการทำงานของระบบ

จากที่กล่าวมาในข้างต้น เมื่อผู้ใช้ระบบนี้เริ่มใช้อุปกรณ์เครื่องเสียงในระบบซึ่งประกอบด้วย เทป, จูนเนอร์, อีควอไลเซอร์ และ ปรี-โทนแอมป์ปรีไฟล์ แล้วเปิดคอมพิวเตอร์และเรียกโปรแกรมของระบบ ที่จอคอมพิวเตอร์และหน้าปัทม์ของเครื่องเสียงจะแสดงผลที่เหมือนกัน จากนั้นผู้ใช้สามารถปรับแต่ง, ควบคุมการทำงานต่าง ๆ ที่หน้าปัทม์ของตัวเครื่องหรือทำการคลิกปุ่มต่าง ๆ บนจอคอมพิวเตอร์ก็ได้ ที่จอคอมพิวเตอร์และหน้าปัทม์เครื่องเสียงก็จะแสดงค่าที่เปลี่ยนแปลงเช่นเดียวกัน



รูปที่ 1.3 รูปแสดงหน้าจอคอมพิวเตอร์ขณะทำงาน

บทที่ 2

สถาปัตยกรรมของไมโครคอนโทรลเลอร์แบบชิพเดี่ยวตระกูล MCS-51

Single Chip Microcontroller system -51 family Architectural

ไมโครโปรเซสเซอร์แบบชิพเดี่ยว (Single Chip Microcontroller) คือ ไมโครคอมพิวเตอร์แบบที่มีขนาดเล็กโดยบรรจุไว้ในแผงวงจรรวม (Integrated Circuit) เพียงชิพเดี่ยวเหมาะสำหรับงานควบคุมอุปกรณ์อื่น ๆ แบบอัตโนมัติ เพราะผู้ใช้สามารถเขียนโปรแกรมควบคุมการทำงานได้ตามต้องการ ไมโครคอนโทรลเลอร์แบบชิพเดี่ยวตระกูล 51 หรือ MCS51 อันได้แก่ เบอร์ 8051 และ 8052 ซึ่งมีโครงสร้างและชุดคำสั่งแตกต่างกันเพียงเล็กน้อย

MCS-51 มีข้อดีดังนี้

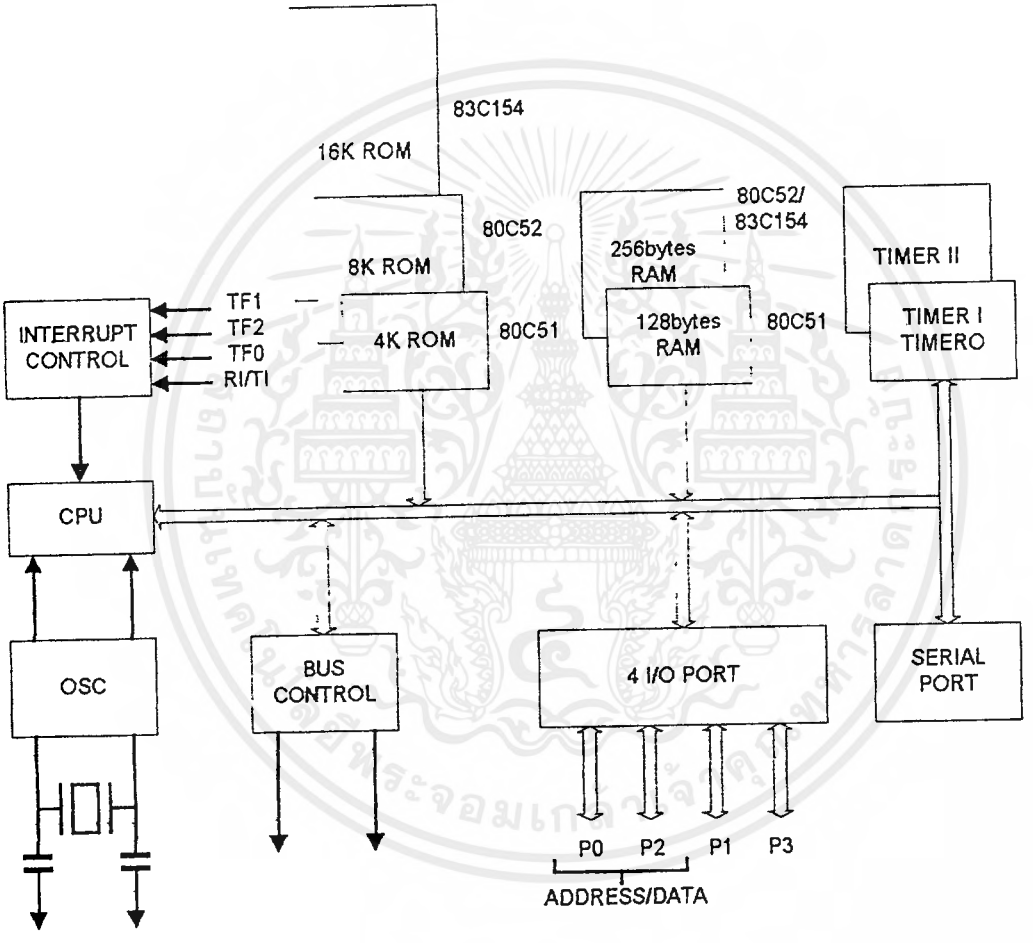
- สามารถนำเอาข้อมูลมา AND,OR หรือทำ Complement ทั้งทีละ 8 บิท และ 1 บิท
- สามารถใช้กับหน่วยความจำสำหรับโปรแกรม (Program Memory) ซึ่งเป็นหน่วยความจำที่สำหรับเก็บชุดคำสั่งที่จะให้ MCS-51 ทำงาน ได้สูงสุด 64 กิโลไบท์ ทำให้เขียนโปรแกรมควบคุมการทำงานได้มาก
- สามารถต่อกับหน่วยความจำสำหรับข้อมูล (Data Memory) ซึ่งเป็นหน่วยความจำสำหรับเก็บข้อมูลในระหว่างการทำงานของโปรแกรมได้สูงสุด 64 กิโลไบท์
- ใน 8051 และ 8751 มีหน่วยความจำสำหรับโปรแกรมจำนวน 4 กิโลไบท์ (ใน 8052 และ 8752 มีหน่วยความจำสำหรับโปรแกรมจำนวน 8 กิโลไบท์) อยู่ภายในวงจรรวมทำให้ไม่ต้องต่อหน่วยความจำสำหรับโปรแกรมอยู่ภายนอก ระบบรวมทั้งหมดจึงมีขนาดเล็ก และสัญญาณรบกวนจากภายนอกจะทำให้ MCS-51 ทำงานผิดพลาดได้ยาก
- มีพอร์ทแบบขนาน (Parallel Port) สำหรับข้อมูลเข้าและออกจำนวน 32 บิท ที่ข้อมูลแต่ละบิทเป็นอิสระต่อกัน
- มีวงจร Timer/Counter ขนาด 16 บิท 2 ชุด (8052 มี 3 ชุด) ที่ทำงานในโหมดต่าง ๆ ได้ถึง 4 โหมด
- มี Universal Asynchonous Receiver Transmitter (UART) สำหรับรับ-ส่งข้อมูลอนุกรม (Serial) แบบ Full duplex ที่สามารถเลือกรูปแบบการรับ-ส่งข้อมูลได้ 4 แบบ
- มีแหล่งกำเนิดสัญญาณขอขัดจังหวะการทำงานของโปรแกรม(Interrupt Request Signal) 6 แหล่ง ซึ่งสามารถทำกระโดดไปทำงานตอบสนองการขัดจังหวะ (Interrupt Service Routine) ได้ต่างๆ กัน 5 ตำแหน่ง

- สามารถเลือกการทำงานให้อยู่ในโหมดของ Idle และ Power Down ซึ่งจะประหยัดการใช้กำลังไฟในการทำงาน

ซึ่งจากข้อดีดังกล่าว จึงทำให้ MCS-51 เป็นที่นิยมนำมาใช้ในการควบคุมระบบอัตโนมัติมาก คุณสมบัติดังกล่าวบรรจุไว้ในวงจรรวมเดียว (Single Chip) ขนาด 40 ขา ดังนั้นจึงสามารถออกแบบให้ระบบทั้งหมดมีขนาดเล็ก และการที่ทั้งหมดบรรจุอยู่ภายในวงจรรวมเดียวจึงทำให้การตรวจสอบหาข้อผิดพลาดในระบบง่ายไม่สลับซับซ้อนรวมทั้งลดปัญหา เรื่องการที่มีสัญญาณรบกวนในระบบจนทำให้การทำงานผิดพลาดไป แต่การที่จะนำเอา MCS-51 มาใช้งานได้จำเป็นที่จะต้องศึกษา และทำความเข้าใจถึงโครงสร้างและองค์ประกอบของ MCS-51 เสียก่อน แล้วถึงจะเขียนโปรแกรมเพื่อควบคุมการทำงานของ MCS-51 ให้เป็นไปตามต้องการ

2.1 โครงสร้างของ 8051

ภายใน 8051 จะประกอบด้วย GATE ต่าง ๆ เช่น AND,OR,NOT ซึ่ง GATE เหล่านี้จะถูกนำเอามาออกแบบให้มีหน้าที่ทำงานต่าง ๆ เช่น วงจรถอดรหัสคำสั่ง (Intruction Decoder), วงจรสร้างสัญญาณนาฬิกา (Clock Signal Generator) โครงสร้างภายในของ 8051 จะประกอบด้วยส่วนย่อย ๆดังไดอะแกรมในรูปที่ 2.1



รูปที่ 2.1 ไดอะแกรมโครงสร้างของ 8051

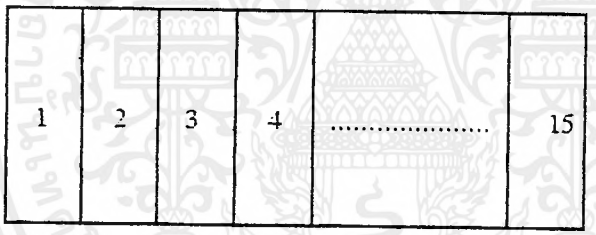
ไดอะแแกรมในรูปที่ 2.1 เป็นโครงสร้างใหญ่ๆ ของ 8051 เนื่องจากลักษณะของ 8051 เป็นคอมพิวเตอร์จึงประกอบด้วย 3 ส่วนหลักๆ คือ

ส่วนที่ 1 คือ CPU (General Processing Unit) หรือตัวประมวลผล ส่วนนี้จะมีวงจรที่ทำหน้าที่สร้างสัญญาณควบคุมในการติดต่อกับส่วนอื่น ๆ เรียกว่าวงจรควบคุม (Control Unit) สัญญาณที่สร้างจากวงจรควบคุมได้แก่สัญญาณสำหรับการติดต่อกับหน่วยความจำ, อุปกรณ์รับ

ข้อมูลเข้าหรือส่งข้อมูลออกจากตัว 8051 ซึ่งส่วนควบคุมการขัดจังหวะ(Interrupt Control) และส่วนควบคุมบัส (Bus Control) ก็เป็นส่วนหนึ่งของวงจรควบคุมด้วยการสร้างสัญญาณควบคุมจากส่วน CPU นี้ จะทำการสร้างสัญญาณโดยการถอดรหัสจากคำสั่ง (Instruction) ตามที่มีการกำหนดไว้ และสัญญาณที่สร้างขึ้นมาจะอ้างอิงกับสัญญาณนาฬิกา ที่สร้างจากวงจรออสซิลเลเตอร์เพื่อให้ทุกๆส่วนในวงจรทำงานประสานกัน (Synchronize) อย่างถูกต้อง

ใน CPU นี้ยังประกอบด้วยส่วนย่อยอีกส่วนที่เรียกว่าส่วนประมวลผล(Arithmetic Logic Unit) ส่วนนี้จะทำหน้าที่ประมวลผลข้อมูลเช่น การบวก,ลบ,คูณ หรือหารข้อมูลแล้วนำผลลัพธ์ไปเก็บไว้ใน รีจิสเตอร์หรือหน่วยความจำที่ต้องการ

ส่วนที่ 2 คือ หน่วยความจำ (Memory) มีไว้สำหรับจดจำข้อมูล ถ้าจะให้เห็นภาพพจน์ของหน่วยความจำได้ดีก็คือ หน่วยความจำเปรียบเสมือนกล่องเก็บเอกสารจำนวนมากที่นำมาต่อเรียงกันไว้ แต่ละกล่องก็มีเอกสาร 1 แผ่น ดังในรูปที่ 2.2 มีกล่องเอกสารทั้งหมด 15 กล่อง



รูปที่ 2.2 ภาพเสมือนของหน่วยความจำ

ถ้าต้องการเอาเอกสารจากกล่องใด หรือเอาเอกสารไปเก็บที่กล่องใด จะต้องรู้หมายเลขของกล่องข้อมูลเสียก่อน ซึ่งถ้าเป็นหน่วยความจำแล้วหมายเลขของกล่องก็คือตำแหน่งของหน่วยความจำหรือแอดเดรส (Address) นั่นเอง การเอาข้อมูลเข้าไปเก็บในหน่วยความจำเรียกว่าการเขียน (write) ข้อมูล และการเอาข้อมูลออกจากหน่วยความจำจะเรียกว่าการอ่าน(Read)ข้อมูล ซึ่งแต่ละตำแหน่งของหน่วยความจำจะเก็บข้อมูลได้แค่ค่าเดียวเท่านั้น ในไมโครโปรเซสเซอร์ทั่วไปรวมทั้ง 8051 นั้นข้อมูลในแต่ละตำแหน่งของหน่วยความจำจะมีค่าได้เพียง 8 หลักของเลขฐาน2 (8 บิทเท่ากับ 1 ไบท์) ดังนั้นแต่ละตำแหน่งของหน่วยความจำจะเก็บข้อมูลมีค่าได้ระหว่าง 0 ถึง 255 (00000000 ถึง 11111111 ในเลขฐาน 2) แต่จำนวนตำแหน่งที่จะเก็บข้อมูลได้ขึ้นกับไมโครโปรเซสเซอร์แต่ละเบอร์ การติดต่อกับหน่วยความจำจะต้องมีสัญญาณ 3 กลุ่มคือ

1. แอสเดรสหรือค่าตำแหน่งที่ต้องการติดต่อกับหน่วยความจำใน 8051 จะติดต่อกับหน่วยความจำประเภท Program Memory หรือ Data Memory ได้สูงสุดชนิดละ 65536 ตำแหน่ง ดังนั้นการอ้างอิงแต่ละตำแหน่งของหน่วยความจำ จะต้องใช้เส้นแสดงตำแหน่งในเลขฐาน 2 ทั้งหมด 16 เส้น (2^{16} เท่ากับ 64*1024 = 65536)

2. ข้อมูลที่จะอ่านหรือเขียนกับหน่วยความจำที่ตำแหน่งในข้อ 1

3. สัญญาณควบคุมที่จะส่งไปยังหน่วยความจำ เพื่อบอกกับหน่วยความจำว่าต้องการอ่านหรือเขียนข้อมูล

สัญญาณเหล่านี้จะถูกวงจรควบคุมภายใน 8051 สร้างมาจากวงจรถอดรหัสของคำสั่งที่ 8051 อ่านจากหน่วยความจำ Program Memory เข้าไปทำงานนั่นเอง ในรูปที่ 2.1 หน่วยความจำได้แก่ 4K ROM และ 128 Byte RAM ซึ่งขนาดของหน่วยความจำนี้มีขนาดต่าง ๆ กันตามเบอร์ของไมโครโปรแกรมเมอร์ และจะอธิบายโดยละเอียดในข้อ 2.1

ส่วนที่ 3 อุปกรณ์อินพุทและเอาท์พุท (Input/Output Device) เป็นส่วนที่จะใช้ส่งข้อมูลเข้าหรือออกจาก 8051 ทำให้ 8051 ติดต่อกับภายนอกได้ ดังในไดอะแกรมรูปที่ 2.1 อุปกรณ์อินพุท และเอาท์พุทได้แก่ 4 I/O Port. Timer0, Timer1, Serial Port การทำงานของแต่ละส่วนมีดังนี้

1. 4 I/O Port คำว่าพอร์ทหมายถึงจุดที่จะติดต่อกับส่วนที่อยู่ภายนอก 4 I/O Port ของ 8051 เป็นที่ใช้สำหรับรับ-ส่งข้อมูล ซึ่งเป็นสัญญาณดิจิตอลเข้าหรือออกจากตัว MCS-51 พอร์ทมีทั้งหมด 4 พอร์ท โดยแต่ละพอร์ทจะรับ-ส่งข้อมูลได้ 8 บิท มีพอร์ท P0,P1,P2 และ P3 บางพอร์ทจะใช้ทำงานมากกว่า 1 อย่างก็ได้ เช่น พอร์ท P0 และP2 จะใช้สำหรับส่งค่าตำแหน่ง (Address) ของหน่วยความจำที่ต้องการติดต่อ และพอร์ท P0 จะใช้รับ-ส่งข้อมูลเมื่อติดต่อกับหน่วยความจำได้ด้วยแต่สิ่งเหล่านี้ไม่ได้เกิดขึ้นที่เวลาเดียวกัน แต่จะใช้วิธีทำงานตามลำดับโดยควบคุมจากสัญญาณควบคุม (Control) ที่ถอดรหัสมาจากแต่ละคำสั่งที่ทำให้คอมพิวเตอร์ทำงานนั่นเอง และสัญญาณทั้งหมดจะอ้างอิงกับจากสัญญาณนาฬิกา

2. Time 0 และ Time 1 เป็นวงจรนับที่สามารถกำหนดให้ทำงานนับจำนวนไซเคิลของสัญญาณที่ต่อจากภายนอก 8051 หรือจำนวนไซเคิลของสัญญาณภายใน 8051 ก็ได้ค่าจากการนับจะถูกอ่านหรือตั้งค่าเริ่มต้นของการนับได้โดย CPU

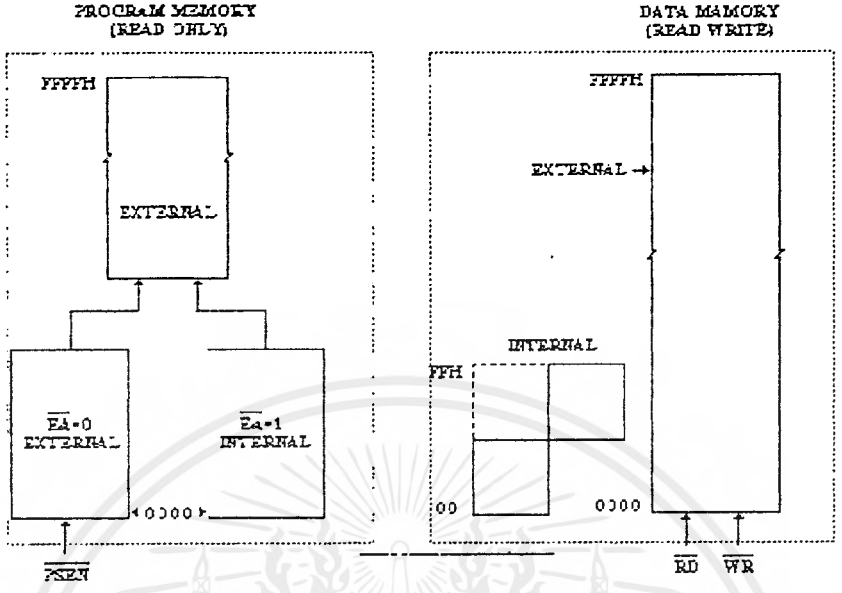
3. Serial Port หรือพอร์ทอนุกรม CPU จะอ่านและเขียนข้อมูลกับ Serial Port เป็นแบบ 8 บิท แต่ข้อมูลจะถูกส่งออกจาก 8051 เรียงไปทีละบิทออกจากขา TXD และ ในการรับข้อมูลเข้าก็จะรับเข้ามาทีละบิททางขา RXD แล้วจัดเรียงใหม่เป็น 8 บิท เพื่อให้ CPU อ่านไปใช้งานต่อไป

8051 มีพอร์ทให้ใช้งานได้หลายแบบ ทำให้สะดวกแก่การนำไปใช้งานต่างๆ มากมาย การจะนำพอร์ทเหล่านี้ไปใช้งานได้จะต้องเขียนโปรแกรมขึ้นมาควบคุมที่จะกล่าวต่อไป

2.2 การจัดการหน่วยความจำของ 8051

หน่วยความจำของ 8051 แบ่งออกไว้เป็น 2 แบบตามลักษณะของการใช้งานคือ

1. **Program Memory** เป็นหน่วยความจำที่ใช้เก็บคำสั่งในรูปรหัสภาษาเครื่อง (Machine Language) ซึ่งต้องการให้ 8051 ทำงาน เมื่อ 8051 ทำงานก็จะอ่านข้อมูลที่เก็บในหน่วยความจำประเภทนี้เข้าไปถอดรหัสแล้วสร้างสัญญาณควบคุมสิ่งอื่น ๆ ตามการทำงานของแต่ละคำสั่งนั้น หน่วยความจำแบบนี้จะต้องเป็นแบบ Read Only Memory (ROM) และผู้ใช้ต้องเขียนข้อมูลในแต่ละตำแหน่งของหน่วยความจำเป็นรหัสภาษาเครื่องของ 8051 ตามลำดับการทำงานที่ต้องการ (หน่วยความจำแบบROM เป็นแบบ Non Volatite ซึ่งเมื่อปิดไฟแล้วข้อมูลก็จะไม่มีการสูญหาย) การเขียนข้อมูลลงไปบน ROM จะต้องใช้ข้อมูลพิเศษ ในระหว่างการทำงานของ 8051 ผู้ใช้จะไม่สามารถใช้คำสั่งทำการเขียนข้อมูลลงในหน่วยความจำแบบนี้ได้ จำนวนตำแหน่งสูงสุดของหน่วยความจำแบบนี้ที่ 8051 จะใช้งานได้คือ 65536 ตำแหน่ง ค่าของตำแหน่ง (Address) จะเขียนเป็นเลขฐาน 16 ได้ตั้งแต่ 0000H ถึง FFFFH หน่วยความจำตำแหน่ง 0000H ถึง 0FFFH จำนวน 4 กิโลไบท์ นั้นผู้ใช้จะเลือกได้ว่าเป็นตำแหน่งของ ROM ที่อยู่ภายในหรือภายนอก 8501 (ไมโครคอนโทรลเลอร์เบอร์อื่น ๆ เช่น 8052 จะมีขนาดของROM ส่วนนี้ได้ถึง 8 กิโลไบท์ ตำแหน่ง 0000H ถึง 1FFFH) ถ้าต้องการให้ 8051 ทำงานตามคำสั่งที่เก็บไว้ใน ROM ภายใน 8051 ก็ให้ป้อนสัญญาณสภาวะลอจิก High (1) เข้าที่ขา EA ของ 8051 แต่ถ้าต้องการให้ทำงานในโปรแกรมที่เก็บไว้ใน ROM ภายนอก 8051 ก็ให้ต่อลอจิก Low(0) เข้าที่ขา EA ของ 8051 ส่วนหน่วยความจำที่ตำแหน่ง 1FFFH ถึง FFFFH จะต้องต่ออยู่ภายนอก 8051 เสมอ ดังแสดงในแผนภูมิหน่วยความจำ (Memory Map) ในรูปที่ 2.3



รูปที่ 2.3 แผนภูมิหน่วยความจำของ 8051

Internal Memory หมายถึงหน่วยความจำนั้นอยู่ภายใน 8051 ส่วน External Memory หมายถึงหน่วยความจำนั้นอยู่ภายนอก 8051

ไมโครคอนโทรลเลอร์เบอร์ 8031.8051 และ 8751 นั้นโดยโครงสร้างและรหัสคำสั่งจะเหมือนกันทุกประการแตกต่างกันที่

- 8031 จะไม่มี ROM ขนาด 4 กิโลไบท์ อยู่ภายใน ผู้ใช้จะต้องเลือกการใช้งาน Program Memory อยู่ภายนอกวงจรรวมทั้งหมด 64 กิโลไบท์

- 8051 จะมี ROM ขนาด 4 กิโลไบท์อยู่ภายใน ถ้าต้องการเก็บคำสั่งควบคุมการทำงานไว้ในหน่วยความจำส่วนนี้ จะต้องส่งโปรแกรมคำสั่งไปให้โรงงานผู้ผลิตทำการเขียนใส่ใน ROM ให้ตั้งแต่ในขั้นตอนของการผลิตวงจรรวม ผู้ใช้ไม่สามารถแก้ไขโปรแกรมได้เอง ถ้าจะนำมาใช้งานโดยเก็บโปรแกรมไว้ในหน่วยความจำช่วง 4 กิโลไบท์แรกอยู่ภายนอกก็สามารถทำได้ โดยการต่อ ROM ไว้ภายนอก แล้วต่อขา EA ของ 8051 ไว้สัญญาณที่มีสภาวะลอจิกเป็น 0

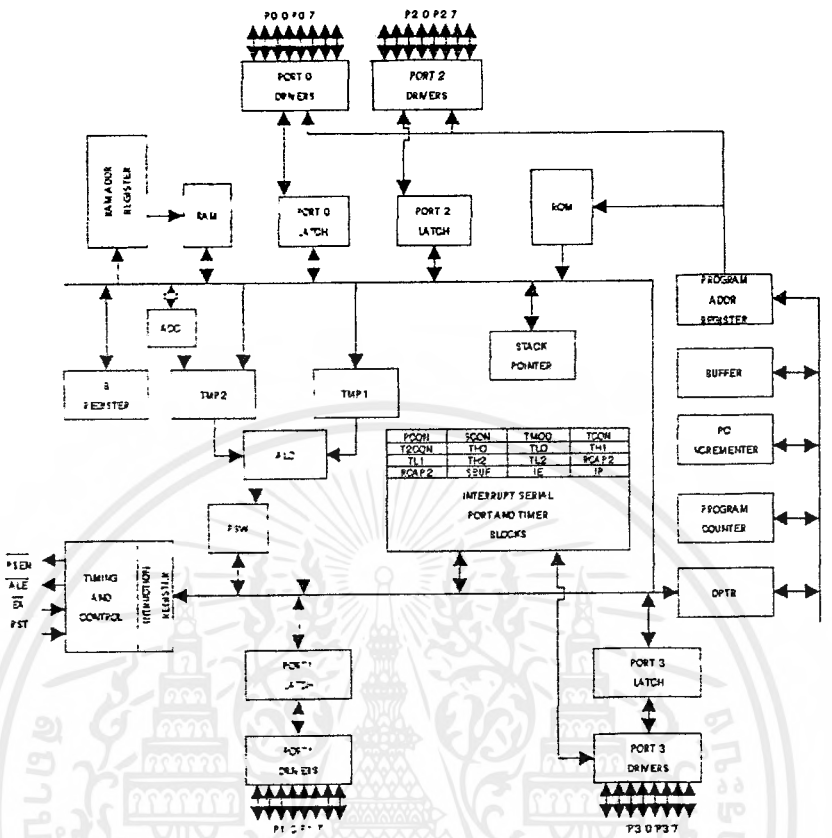
- 8751 จะมีหน่วยความจำขนาด 4 กิโลไบท์เป็นแบบ EPROM (Erasable Program Read Only Memory) อยู่ภายในวงจรรวมเอาไว้ ใช้เก็บโปรแกรมคำสั่งที่จะให้ 8751 ทำงานผู้ใช้สามารถเขียนคำสั่งลงไปใน EPROM ได้เองโดยใช้เครื่องมือที่เรียกว่าเครื่องโปรแกรม EPROM (EPROM Programmer) และผู้ใช้สามารถแก้ไขโปรแกรมที่อยู่ใน EPROM ได้โดยการล้างข้อมูลในทุกตำแหน่งของ EPROM ออกด้วยการฉายแสงอุลตร้าไวโอเล็ต (Ultraviolet) ผ่านกระจกใสบนวงจรรวมเข้าไปยังวงจรภายใน ตามเวลาที่กำหนดในคู่มือเฉพาะ (Data sheet) ของ 8751 จากนั้นก็ใช้

เครื่องโปรแกรม EPROM เขียนโปรแกรมลงไปใหม่ 8751 นี้จะสะดวกมากสำหรับการพัฒนาโปรแกรม

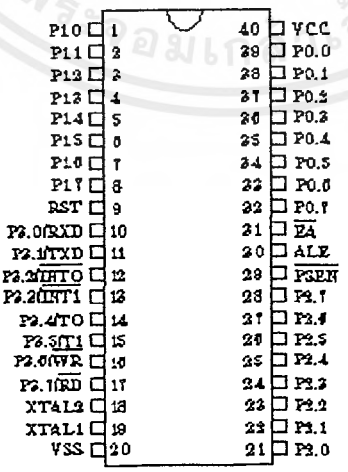
2. **Data Memory** เป็นหน่วยความจำที่ 8051 จะใช้สำหรับพัก ,เก็บข้อมูล แล้วเรียกมาใช้ใหม่ในระหว่างการทำงานของ 8051 การอ่านหรือเขียนข้อมูลจากหน่วยความจำจะกระทำโดยคำสั่งที่เก็บไว้ใน Program Memory หน่วยความจำแบบนี้เป็นประเภท Random Access Memory (RAM) ถ้ามีไฟเลี้ยงอยู่ข้อมูลที่เก็บไว้จะไม่สูญหายแต่ถ้าปิดเครื่องหรือไม่จ่ายไฟให้แก่ RAM แล้วข้อมูลใน RAM ก็จะสูญหายไป การสูญหายของข้อมูลไม่ได้หมายความว่าไม่มีอะไรอยู่เลยแต่เป็นการที่มีข้อใหม่ซึ่งไม่ใช่ข้อมูลที่เก็บไว้เดิมเข้ามาอยู่แทนที่ เช่น เดิมอาจเก็บข้อมูล 18H ไว้ที่ตำแหน่ง 1900H เมื่อปิดไฟแล้วเปิดใหม่ ข้อมูลที่ตำแหน่ง 1900H จะไม่ใช่ 18H อาจเป็นค่าอะไรก็ได้ ซึ่งเรียกการเกิดลักษณะแบบนี้ว่าข้อมูลสูญหายไป หน่วยความจำแบบ Data Memory จะมีอยู่ 2 ชุด ชุดหนึ่งอยู่ภายใน 8051 จำนวน 128 ไบท์ ที่ตำแหน่ง 00H ถึง 7FH (เบอร์ 8052 จะมี 256 ไบท์อยู่ที่ตำแหน่ง 00H ถึง FFH) และอีกชุดหนึ่งจะต้องต่ออยู่ภายนอกของวงจรรวม 8051 มีได้สูงสุด 65536 ไบท์ (64 กิโลไบท์) อยู่ที่ตำแหน่ง 0000H ถึง FFFFH ดังแสดงในรูป 1.4 หน่วยความจำแบบ Data Memory ภายใน 8051 ที่ตำแหน่ง 80H ถึง FFH นั้น ไม่ได้มีอยู่ทุกตำแหน่ง จะมีเฉพาะในบางตำแหน่ง ซึ่งเรียกหน่วยความจำบางตำแหน่งนี้ว่า Special Function Register (SFR) เพราะจะใช้หน่วยความจำเหล่านี้สำหรับงานพิเศษเท่านั้น แต่ละตำแหน่งของหน่วยความจำแบบ SFR นี้อาจเป็น RAM หรือวงจรนับ (Counter) วงจรตั้งเวลา (Timer) ก็ได้ เช่นเป็น Timer0 ,Timer1 ดังนั้นใน 8051 จึงไม่ถือว่า SFR เป็น Data Memory ถ้าเป็น 8052 ซึ่งมี Data Memory ขนาด 256 ไบท์ จะใช้บางตำแหน่งของหน่วยความจำช่วงตำแหน่ง 80H ถึง FFH เป็น SFR ส่วนตำแหน่งอื่นที่เหลือก็เป็น RAM เหมือนกับหน่วยความจำช่วง 00H ถึง 7FH นั่นเอง

2.3 สถาปัตยกรรมของ 8051

ในรูปที่ 2.4 เป็นสถาปัตยกรรมภายในของ 8051 ซึ่งจะอธิบายถึงส่วนย่อยๆ ของภายใน 8051 เพียงชิพเดียว และสัญญาณจากภายในจะต่อออกสู่ภายนอกทางขาของ 8051 ที่มีอยู่ 40 ขา ดังรูปที่ 2.5



รูปที่ 2.4 สถาปัตยกรรมภายในของ8051



รูปที่ 2.5 ไดอะแกรมขาของ 8051 แบบ DIP

8051 ไมโครคอนโทรลเลอร์ที่บรรจุอยู่ในวงจรรวมแบบ Dual Inline Package (DIP) ซึ่งแต่ละข้างของ 8051 มีขาอยู่ข้างละ 20 ขารวมทั้งหมด 40 ขานั้นจะใช้งานต่าง ๆ กันดังนี้ คือ

Vcc

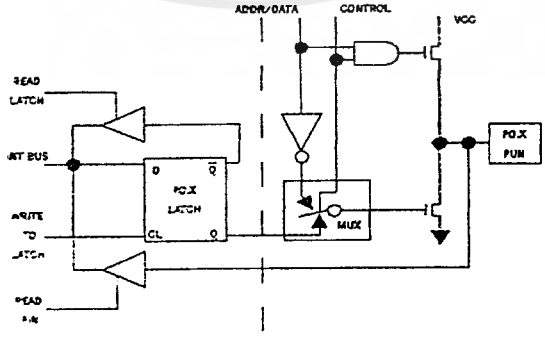
ขา 40 เป็นขาที่ต้องป้อนไฟเลี้ยง +5 โวลท์เข้าไปเพื่อให้วงจรรวมทำงานได้ ระดับโวลเตจของลอจิก 0 และ 1 ของ 8051 จึงต่อเข้ากับอุปกรณ์ลอจิกแบบ TTL ได้โดยตรง

Vss

ขา 20 เป็นขาที่ต้องต่อกับกราวด์ (Ground) ของแหล่งจ่ายไฟ การต่ออุปกรณ์ทั้งหมดจะต้องมีกราวน์ของอุปกรณ์ต่อเข้าด้วยกัน

Port 0

เป็นพอร์ทขนานขนาด 8 บิท อยู่ที่ขา 39 ถึง 32 เริ่มจากบิท 0 ถึงบิท7 ตามลำดับดังในรูปที่ 2.5 แต่ละขาจะเขียนว่า P0.0, P0.1, ... , P0.7 หมายถึงบิท 7 ของพอร์ท จ ซึ่งเป็นบิทที่มีนัยสำคัญสูงสุด (Most Significant) และ P0.0 คือบิท 0 ของพอร์ท 0 เป็นบิทที่มีนัยสำคัญต่ำสุด (Least Significant) พอร์ท 0 นี้ใช้ได้ทั้งการรับ-ส่งข้อมูลก็ได้ ข้อมูลที่ส่งออกทางพอร์ท 0 จะถูก Latch ไว้ที่ขาของพอร์ท โครงสร้างแต่ละบิทของพอร์ท 0 เป็นแบบ Open Drain Bidirectional ดังรูปที่ 2.6



รูปที่ 2.6 โครงสร้างของพอร์ท 0

ในรูปที่ 2.6 เมื่อเปรียบเทียบกับรูปที่ 2.4 ส่วนที่ 1 ของรูป 2.6 ก็คือ Port 0 Latch ในรูปที่ 2.4 และส่วนที่ 2 ของรูป 2.6 ก็คือ Port 0 Driver ของรูปที่ 2.4 นั่นเอง จากโครงสร้างในรูปที่ 2.6 เมื่อมีคำสั่งการเขียนข้อมูลมายังพอร์ท 0 ข้อมูลจาก Internal Data Bus จะถูก Latch ไว้ที่ D-FF โดยสัญญาณ "Write to Latch" ที่ถูกสร้างมาจากส่วน Timing and Control และในการอ่านข้อมูลจากพอร์ท 0 จะอ่านได้ 2 แบบคือการอ่านข้อมูลที่ส่งไปเก็บไว้ที่พอร์ทก็จะมีสัญญาณ Read Latch มาเพื่ออ่านข้อมูลจาก D-FF กลับเข้าไปยัง Internal Data Bus การอ่านข้อมูลอีกแบบก็คือการอ่านสภาวะของสัญญาณที่เข้ามาทางพอร์ท 0 ก็จะมีสัญญาณ Read Pin มาควบคุมการอ่าน พอร์ท 0 จะใช้งานหลายอย่างดังนี้

1. ใช้สำหรับส่งค่าตำแหน่งหน่วยความจำภายนอกที่ต้องการติดต่อด้วย ตำแหน่งหน่วยความจำสูงสุดที่จะติดต่อได้ก็คือ 64 kbyte จึงมีค่าตำแหน่งหน่วยความจำ 16 บิทของเลขฐาน 2 ค่าตำแหน่งหน่วยความจำ 8 บิทล่างจะถูกส่งออกไปทางพอร์ท 0 และ 8 บิทบนจะส่งออกไปทางพอร์ท 2

2. ใช้รับส่งข้อมูลกับ Data Memory หรือใช้รับข้อมูลจาก Program Memory

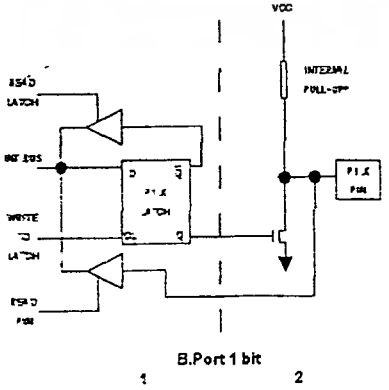
3. ใช้รับส่งข้อมูลผ่านทางพอร์ทโดยตรง ในกรณีที่ไม่มีการใช้หน่วยความจำของ Program Memory หรือ Data Memory ภายนอก

วงจรภายในส่วน Timing and Control จะเป็นตัวสร้างสัญญาณมาควบคุมวงจรในรูปที่ 2.6 เพื่อให้การทำงานแต่ละอย่างข้างต้น เมื่อแต่ละบิทของพอร์ท 0 ทำงานตามข้อ 1 และ 2 ข้างต้น วงจร Timing and Control จะทำให้สภาวะลอจิกของขา Control เป็น 1 ซึ่งทำให้สวิทช์ MUX อยู่ในตำแหน่งข้างบน เมื่อพอร์ท 0 จะส่งข้อมูลซึ่งเป็นค่าตำแหน่งหน่วยความจำ หรือข้อมูลที่จะเขียนออกไปยังหน่วยความจำภายนอกก็จะส่งค่าดังกล่าวมายัง ADDR/DATA ถ้าข้อมูลที่ส่งมาเป็น 1 จะทำให้สัญญาณออกจาก AND GATE เป็น 1 และสัญญาณที่ออกจาก Inverter เป็น 0 ดังนั้น FET ตัวบน ON (สภาวะ ON ของ FET คือความต้านทานระหว่างขา D กับ S มีค่าต่ำมากเสมือนกับเป็นวงจรปิด) ส่วน FET ตัวล่าง OFF (สภาวะ OFF ของ FET คือความต้านทานระหว่างขา D กับ S มีค่าสูงมากเหมือนกับเป็นวงจรเปิด) สภาวะลอจิกที่ขา PO.X Pin จะเป็น 1 แต่ถ้าข้อมูลที่ส่งออกมายัง ADDR/DATA เป็น 0 ก็จะทำให้สัญญาณจาก AND GATE เป็น 0 และสัญญาณที่ออกจาก Inverter เป็น 1 ดังนั้น FET ตัวบนจะ OFF ส่วน FET ตัวล่างจะ ON ทำให้สภาวะลอจิกที่ขา PO.X Pin เป็น 0 เมื่อ 8051 ต้องการใช้พอร์ท 0 สำหรับการอ่านข้อมูลจากหน่วยความจำภายนอก หรือใช้ทำงานในข้อ 3 ข้างบน ก็จะทำได้โดยวงจร Timing and Control ทำให้สภาวะ

ลอจิกของสัญญาณ Control ในรูปเป็น 0 ทำให้เอาท์พุทจาก AND GATE เป็น 0 FET ตัวบนจะ OFF และสวิตช์ MUX จะอยู่ในตำแหน่งข้างล่างดังนั้น FET ตัวล่างจะ ON หรือ OFF ก็แล้วแต่ข้อมูลที่ขา Q ของ D-FF เมื่อมีการเขียนข้อมูลจาก Internal Data Bus มายัง D-FF ก็จะมีสัญญาณ Write to Latch มายัง D-FF ด้วย ถ้าข้อมูลที่เขียนมาเป็น 1 ก็จะทำให้ขา Q มีสภาวะลอจิกเป็น 0 ทำให้ FET ตัวล่าง OFFดังนั้นขา PO.X ก็จะอยู่ในสภาวะอิมพีแดนซ์สูง (High Impedance) เพราะเป็น FETทั้ง 2 ตัว OFF แต่ถ้าข้อมูลที่เขียนมาจาก D-FF เป็น 0 จะทำให้ FET ตัวล่าง ON แต่ตัวบน OFF ทำให้สภาวะลอจิกที่ขา PO.X เป็น 1 ดังนั้นพอร์ท 0 สำหรับข้อมูลเข้าจะต้องเขียน 1 มาเก็บไว้ยัง D-FF เสียก่อนเพื่อให้ขา PO.X อยู่ในสภาวะ High Impedance แล้วจึงใช้คำสั่งอ่านสภาวะลอจิกเข้าไปยัง Internal Data Bus ต่อไป โดยคำสั่งอ่านสภาวะลอจิกทางพอร์ท 0 ก็จะทำให้วงจร Timing and Control สร้างสัญญาณ Read Pin สำหรับการอ่านสภาวะลอจิกข้างต้น ถ้าไม่เขียน 1 มาเก็บไว้ยัง D-FF ก่อนที่จะอ่านข้อมูลแล้วยังมีข้อมูลค้างอยู่ที่ D-FF ทำให้ Q เป็น 0 และ Q เป็น 1 ซึ่งทำให้ FET ตัวล่าง ON สัญญาณที่ต่อเข้ามาที่ขา PO.X ไม่ว่าจะมีสภาวะลอจิกใดจะถูกดึงลงกราวด์ ดังนั้นเมื่ออ่านข้อมูลเข้าไปก็จะพบว่าเป็น 0 เสมอ ในการอ่านข้อมูลจากหน่วยความจำภายนอกนั้นวงจร Timing and Control ก็จะเขียนข้อมูลมายัง D-FF ให้เป็น 1 และสร้างสัญญาณ Control ให้มีลอจิกเป็น 0 ก่อนจะอ่านข้อมูลเข้าไปด้วย

Port 1

เป็นพอร์ทขนานขนาด 8 บิท ในรูปที่ 2.5 คือขา P1.0 ถึง P1.7(ขา 1-8) P1.0 หมายถึงบิท 0 ของพอร์ท 1 ซึ่งเป็นบิท Least Significant Bit และบิท P1.7 หมายถึง บิทที่ 7 ของพอร์ท 1 ซึ่งเป็นบิท Most significant bit โครงสร้างของพอร์ท 1 แต่ละบิทมีดังรูปที่ 2.7

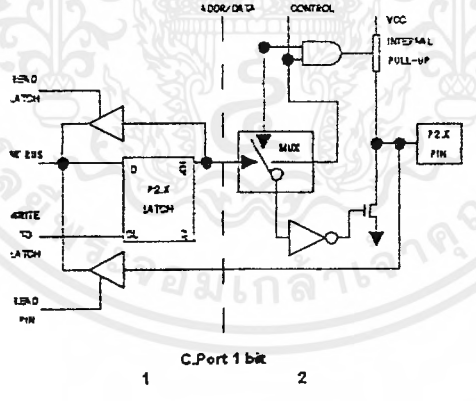


รูปที่ 2.7 โครงสร้างของพอร์ท 1

ส่วนที่ 1 คือ Port 1 Latch ในรูปที่ 2.4 ซึ่งจะมีการทำงานของส่วนที่ 1 ของพอร์ท 0 ในรูป ที่ 2.6 ส่วนที่ 2 คือ Port 1 Driver ในรูปที่ 2.4 Port 1 Driver นี้จะมีตัวต้านทานต่ออยู่เป็น Internal PuLL Up ทาง พอร์ท 1 นี้จะใช้ทำหน้าที่เป็นตัวรับ-ส่งข้อมูลเท่านั้น ข้อมูลที่ถูกส่งออกมาทางพอร์ท 1 จะถูก Latch ไว้แล้วส่งออกไปทางแต่ละขา ก่อนที่จะอ่านข้อมูลเข้าไปทางพอร์ท 1 จะต้องเขียน 1 ไปยังทุกบิทของพอร์ท 1 เสียก่อนเพื่อให้ FET อยู่ในสภาวะ OFF ก่อน มิฉะนั้นแล้วถ้ามีข้อมูล 0 ส่งออกมาค้างอยู่ที่ D-FF จะทำให้ FET อยู่ในสภาวะ ON ดังนั้นถ้าสัญญาณภายนอกส่งเข้ามาที่ขานี้ก็จะถูกลัดวงจรลงกราวด์ โดยไม่สนใจว่าสภาวะลอจิกของสัญญาณที่เข้ามาจะเป็นอะไร ข้อมูลที่อ่านเข้าไปจึงจะเป็น 0 เสมอ

Port 2

พอร์ทขนานขนาด 8 บิท คือขา P2.0 ถึง P2.7 (บิท 0 ถึงบิท 7 ของพอร์ท 2) ในรูปที่ 2.5 โครงสร้างของพอร์ท 2 แต่ละบิทจะมีดังรูปที่ 2.8



รูปที่ 2.8 โครงสร้างของพอร์ท 2

ลักษณะโครงสร้างจะเหมือนกับ Port 0 แตกต่างกันที่ใน Port 2 นั้นภาค Driver จะใช้งานเพียง 2 ลักษณะคือ

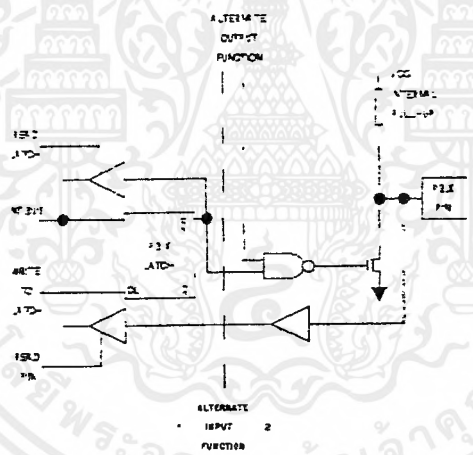
1. ใช้ส่งค่าตำแหน่งหน่วยความจำภายนอกที่ต้องการติดต่อ ค่าตำแหน่งนี้เป็น 8 บิทบนของค่าตำแหน่ง

2. ใช้เป็นพอร์ทรับ-ส่งข้อมูลจากภายนอก

ดังนั้นภาค Driver ของพอร์ท 2 จึงแตกต่างจาก Driver ของพอร์ท 0 โดยที่ในพอร์ท 2 นั้นจะมีเฉพาะ ADDR (ตำแหน่งหน่วยความจำ) เข้ามาที่ MUX (Multiplexer) เท่านั้น นอกนั้นแล้วการทำงานจะเหมือนกัน และเอาท์พุทของพอร์ท 2 จะมี Internal pull-up ซึ่งเป็นตัวต้านทาน และทำให้เอาท์พุทของพอร์ท 2 แสดงสถานะลอจิกเป็น 1 ได้ ถ้า FET อยู่ในสภาวะ OFF บางครั้งเรียกว่า "Quasi-bidirectional" เมื่อใช้เป็นพอร์ทอินพุทก็สามารถทำได้โดย การต่อสัญญาณภายนอกเข้ามาโดยตรง ถ้าสัญญาณภายนอกเป็น 0 ก็จะมีกระแสไหลออกจากพอร์ท (Source Current) ในการที่จะใช้พอร์ทนี้เป็นพอร์ทรับข้อมูลเข้า จะต้องเขียน 1 ไปยังแต่ละบิทของพอร์ทเสียก่อน ดังได้อธิบายในเรื่อง Port 0 และ Port 1

Port 3

คือขา P3.0 ถึง P3.7 หรือขา 10-17 ตามลำดับในรูปที่ 2.5 พอร์ทนี้มีโครงสร้างดังรูปที่ 2.9



รูปที่ 2.9 โครงสร้างของพอร์ท 3

ส่วนที่ 1 ในรูปที่ 2.9 เป็นส่วน Latch ข้อมูลที่เขียนมายังพอร์ท 3 ทาง Internal Bus เหมือนกับพอร์ทอื่น ๆ และพอร์ท 3 จะมี Internal pull-up อยู่ทุกบิท แต่พอร์ท 3 นี้แต่ละบิทจะใช้ในการทำงานอื่นได้โดยใช้คำสั่งลาบคุมการทำงาน ในส่วนที่ 2 จะมีสัญญาณ Alternative Output Function ที่สร้างมาจากส่วน Timing and Control สัญญาณ Alternative Output Funtion เป็นสัญญาณที่ส่งออกในกรณีที่ใช้พอร์ท 3 ทำงานในฟังก์ชั่นอื่น และจุด Alternative Input Function เป็นจุดที่จะเอาสัญญาณไปเข้ากับส่วนอื่นตามการทำงานของบิทนั้น แต่ละบิทของพอร์ท 3 จะมีฟังก์อื่นดังนี้

P3.0/RXD (Serial Input Port) เป็นขาที่ใช้รับข้อมูลแบบอนุกรม

P3.1/TXD (Serial Output Port) เป็นขาที่ใช้ส่งข้อมูลแบบอนุกรม

P3.2/INTO (External Interrupt) ใช้รับสัญญาณขัดจังหวะจากภายนอก

P3.3/INT1 (External Interrupt) ใช้รับสัญญาณขัดจังหวะจากภายนอก

P3.4/T0 (Timer/Counter 0 External Input) ขารับสัญญาณเข้าไปยังวงจร Timer/Counter 0 ที่ทำหน้าที่นับจำนวนไซเคิลของสัญญาณ T0 นี้หรือสัญญาณนาฬิกาก็ได้

P3.5/T1 (Timer/Counter 1 External Input) ขารับสัญญาณเข้าไปยัง Timer/Counter 1 ซึ่งมีการทำงานเหมือน T0

P3.6/WR (External Data Memory Write Strobe) ขาสัญญาณควบคุมการเขียนข้อมูลไปยังหน่วยความจำสำหรับข้อมูลฟภายนอก 8051

P3.7/RD (External Data Memory Read Strobe) ขาสัญญาณควบคุมการอ่านข้อมูลจากหน่วยความจำสำหรับข้อมูลภายนอก

RST

ขารีเซ็ทนี้จะใช้ทำการรีเซ็ทการทำงานของ 8051 ที่ขา RST ภายใน 8051 จะมีตัวต้านทานต่อระหว่างขานี้กับกราวด์ (Ground) ถ้าป้อนสัญญาณที่มีสภาวะลอจิก 1 เข้าไปที่ขานี้จะเป็นการรีเซ็ทการทำงานของ 8051 ดังนั้นจึงสามารถต่อตัวประจุ (Capacitor) ภายนอกระหว่างขา RST กับไฟเลี้ยง +5 โวลท์ เพื่อให้เกิดการรีเซ็ทเมื่อเริ่มป้อนไฟเลี้ยงให้กับ 8051 ซึ่งเรียกว่า Power on reset การรีเซ็ทจะ ทำให้ค่าในรีจิสเตอร์ต่างๆ เปลี่ยนไปเป็นค่าหนึ่งดังในตารางรูปที่ 2.10

REGISTER	CONTENT
PC	0000H
ACC	00H
B	00H
PSW	00H
SP	00H
DPTR	0000H
PO-P3	0FFH
IP	00H
IE	0X000000B
TMOD	00H
TCON	00H
T2CON	00H
TH0	00H
TL0	00H
TH1	00H
TL1	00H
TH2	00H
TL2	00H
RCAP2H	00H
RCAP2L	00H
SCON	00H
SBUF	Indeterminate
IOCON	00H

รูปที่ 2.10 ค่าของรีจิสเตอร์เมื่อเกิดการรีเซ็ท 8051

ในตารางรูปที่ 2.10 ช่องทางขวาเป็นค่าของรีจิสเตอร์ที่อยู่ทางซ้ายเมื่อสิ้นสุดการรีเซ็ท ในรีจิสเตอร์ SBUF เมื่อสิ้นสุดการรีเซ็ทจะมีค่าที่ไม่แน่นอน และพอร์ทจะอยู่ในสภาวะลอจิก 1 ทุกบิทตลอดเวลาที่สัญญาณของขา RST เป็น HIGH อยู่

เมื่อสัญญาณที่ขา RST กลับเป็น 0 ก็จะออกจากการรีเซ็ท 8051 จะเริ่มทำงานจากคำสั่งที่อยู่ใน Program memory ตำแหน่ง 0000H เพราะค่าของรีจิสเตอร์ PC (Program Counter) ซึ่งใช้ชี้

ตำแหน่งโปรแกรมที่จะทำงานถูกเปลี่ยนให้เป็น 0000H ดังนั้นผู้ใช้จะต้องเขียนโปรแกรมมาเก็บไว้ที่ตำแหน่ง 0000H ในเครื่องไมโครกอมพิวเตอร์แบบบอร์ดเดี่ยว (Single Board Microcomputer) จะมีโปรแกรมที่เขียนเก็บไว้เริ่มจากตำแหน่ง 0000H นี้เรียกว่ามอนิเตอร์โปรแกรม (Monitor program) ที่จะคอยรับการกดแป้นพิมพ์ (keyboard) และแสดงผลทางตัวแสดงผล (Display) แบบ 7 Segment

ALE

Address Latch Enable ขานี้จะส่งสัญญาณที่มีความถี่ 1/6 เท่าของสัญญาณนาฬิกาจากออสซิลเลเตอร์ สัญญาณนี้จะส่งออกมาตลอดเวลายกเว้นบางครั้งของการติดต่อกับหน่วยความจำสำหรับข้อมูลภายนอก 8051 สัญญาณนี้จะใช้บอกกับอุปกรณ์ภายนอก 8051 ว่าขณะนี้สัญญาณนี้ Active (เป็นลอจิก 1) จะมีการส่งข้อมูลที่เป็น 8 บิท ล่างของตำแหน่งหน่วยความจำภายนอก 8051 ที่ต้องการติดต่อออกไปทางพอร์ท 0 อุปกรณ์ภายนอกจะใช้สัญญาณนี้ในการ Latch ข้อมูลไว้เฉพาะพอร์ท 0 จะส่งค่าตำแหน่งหน่วยความจำ ออกมาเพียงชั่วขณะเท่านั้น ซึ่งในเวลาต่อมาพอร์ท 0 จะใช้รับ-ส่งข้อมูลกับหน่วยความจำภายนอก สัญญาณ ALE จะสามารถต่อเข้ากับอุปกรณ์ TTL ชนิด LS ได้ถึง 8 อินพุท

PSEN

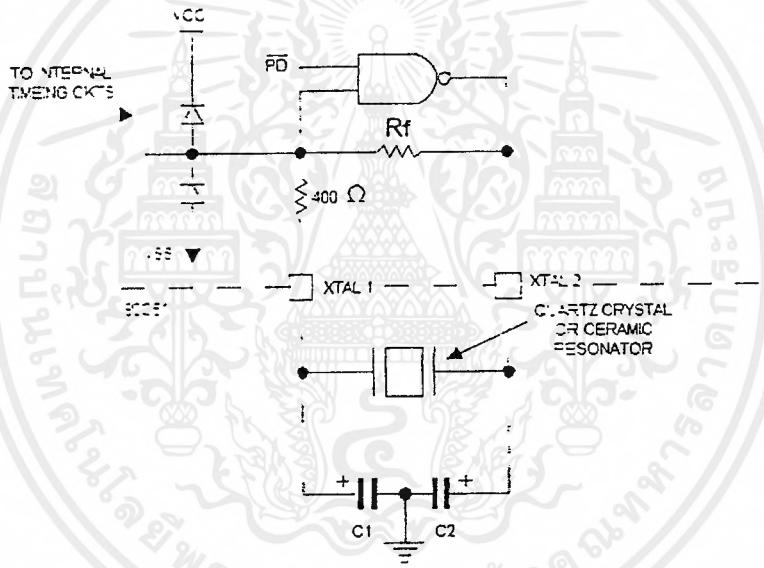
Program Store Enable เป็นขาที่ 29 ในรูปที่ 2.5 ขานี้ปกติจะให้ลอจิก 1 แต่จะส่งลอจิก 0 เมื่อต้องการอ่านคำสั่ง (Fetch Instruction) ที่จะนำไปทำงาน มาจากหน่วยความจำสำหรับโปรแกรมภายนอก 8051 ในกรณีที่อ่านคำสั่งซึ่งเก็บอยู่ในหน่วยความจำสำหรับโปรแกรมภายใน 8051 แล้วสัญญาณนี้จะไม่เปลี่ยนลอจิกเป็น 0 ขา PSEN นี้สามารถต่อไปยังขาอินพุทของ TTL ชนิด LS ได้ถึง 8 อินพุท

EA

External Access ขา 31 ของรูปที่ 2.5 ขานี้เป็นขาอินพุทที่ต่อเข้าไปยังวงจร Timing and Control ในรูปที่ 2.4 เพื่อควบคุมการสร้างสัญญาณ PSEN ถ้าป้อนสัญญาณลอจิก 0 ไปที่ขา EA นี้แสดงว่าโปรแกรมในตำแหน่ง 0000H ถึง 0FFFH ที่ต้องการให้ทำงานถูกเก็บไว้ภายนอก 8051 จะต้องสร้างสัญญาณ PSEN ออกไปยังภายนอก เพื่อทหการ FETCH คำสั่งเข้ามาทำงาน แต่ถ้าสัญญาณที่ป้อนเข้าขา EA เป็น 1 หมายความว่าโปรแกรมในตำแหน่ง 0000H ถึง 0FFFH ถูกเก็บไว้ภายใน 8051 การทำงานในตำแหน่งหน่วยความจำช่วงนี้จะอ่านคำสั่งต้าง ๆ จาก ROM ภายใน 8051

XTAL 1

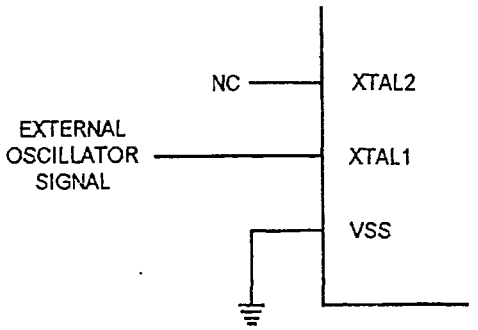
ขาที่ 19 ของรูปที่ 2.5 ขานี้จะต่อเข้ากับขาของ Inverting Amplifier (วงจรขยายแบบป้อนกลับเฟสสัญญาณ) ที่ประกอบเป็นวงจรออสซิลเลเตอร์ ในรูปที่ 2.11 จะเห็นวงจรภายในของออสซิลเลเตอร์ NAND Gate จะทำหน้าที่เป็นวงจรขยายแบบกลับเฟสของสัญญาณที่จะควบคุมให้มีการออสซิลเลตหรือไม่ก็ขึ้นกับสัญญาณ PD ซึ่งต่อมาจากบิท PD ของรีจิสเตอร์ PCON ถ้าต้องการใช้สัญญาณนาฬิกา (Clock Signal) จากภายนอกมาเป็นสัญญาณนาฬิกา ควบคุมการทำงานของ 8051 ก็ให้ป้อนสัญญาณมาที่จุดนี้ แต่ถ้าต้องการใช้สัญญาณออสซิลเลเตอร์ภายในก็ให้ต่อ Crytal หรือเซรามิคเรโซเนเตอร์ดังรูปที่ 2.11 คาปาซิเตอร์ในวงจรควรมีค่าประมาณ 20 PF



รูปที่ 2.11 วงจรออสซิลเลเตอร์ภายใน 8051

XTAL 2

ขาที่ 18 ของรูปที่ 2.5 ขานี้เป็นจุดเอาท์พุทของวงจรขยายแบบกลับเฟสสัญญาณที่ประกอบเป็นวงจรออสซิลเลเตอร์(อินพุทคือขา XTAL 1) ถ้าจะใช้สัญญาณนาฬิกาที่สร้างมาจากภายนอกมาเป็นสัญญาณนาฬิกาของ 8051 แล้ว ให้ปล่อยขานี้ลอยไว้แล้วป้อนสัญญาณนาฬิกามาจากภายนอกเข้ามาที่ขา XTAL 1 ดังรูปที่ 2.12



รูปที่ 2.12 8051 ที่ทำงานโดยสัญญาณที่มาจากภายนอก

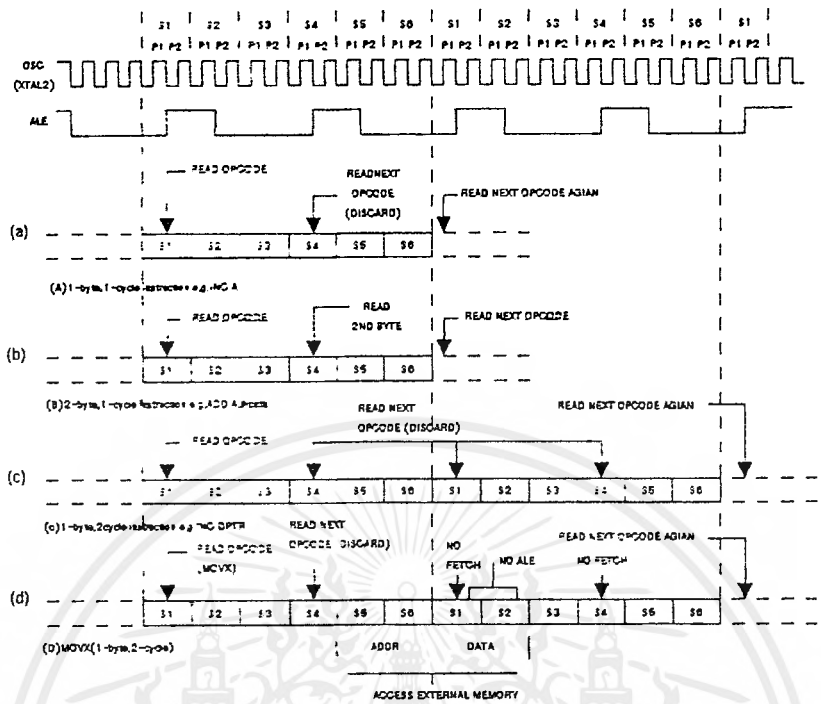
2.4 การทำงานของ 8051

คอมพิวเตอร์จะทำงานด้วยวงจรที่เรียกว่าฮาร์ดแวร์ (Hardware) ประกอบขึ้นมาเพียงอย่างเดียวไม่ได้จะต้องมีโปรแกรมหรือคำสั่งที่จัดเรียงกันไว้ให้คอมพิวเตอรจึทำงานตามลำดับใน 8051 ก็เช่นกัน ผู้ใช้จะต้องเขียนโปรแกรมเป็นภาษาเครื่อง ซึ่งอยู่ในรูปของเลขฐาน 2 เก็บไว้ในหน่วยความจำประเภท Program Memory แต่ละคำสั่งของ 8051 อาจประกอบด้วย 1, 2 หรือ 3 ไบท์แล้วแต่ว่าจะเป็นคำสั่งให้ทำงานอะไร คอมพิวเตอร์ก็จะเหมือนกับคนที่จะต้องทำงานตามคำสั่ง เมื่อรับคำสั่งแล้วก็จะไปทำงานตามคำสั่งนั้น เสร็จสิ้นแล้วก็จะกลับมารับคำสั่งต่อไป

จากรูปที่ 2.4 เมื่อเริ่มป้อนไฟเลี้ยงให้กับ 8051จะเริ่มจากบล็อก Program Counter ซึ่งเป็นวงจรนับ (Counter Circuit) ชนิดหนึ่งส่งค่าตำแหน่งหน่วยความจำสำหรับโปรแกรมลงไปบนบัส (Bus) หมายเลข 1 บัสนี้มีขนาด 16 บิท ค่าตำแหน่งหน่วยความจำนี้จะถูกส่งไปเก็บไว้ที่ Program ADDR Register ที่เป็น วงจร Latch ข้อมูลซึ่งเป็นค่าตำแหน่งหน่วยความจำ จะปรากฏที่บนบัส 16 บิท หมายเลข 2 ถ้าเป็นค่าตำแหน่งหน่วยความจำแรกหลังจากรีเซท ค่าตำแหน่งหน่วยความจำจะเป็น 0000H หน่วยความจำสำหรับโปรแกรมจะเลือกได้ว่าเป็น ROM ภายในหรือ ภายนอก 8051 โดยการป้อนสภาวะลอจิกเข้าไปที่ 8051 ทางขา EA ซึ่งต่ออยู่กับส่วน Timing and Control ทำหน้าที่เป็นวงจรถอดรหัส (Decoder) แล้วสร้างสัญญาณควบคุมต่อไปถ้าป้อนสัญญาณลอจิก 0 เข้าไปที่ขา EA จะเป็นการเลือกใช้ ROM ภายใน 8051 โดยที่วงจร Timing and Control จะสร้างสัญญาณไปยัง ROM ภายในข้อมูล ให้ส่งข้อมูลที่เป็นคำสั่งจากตำแหน่งที่ถูกชี้ด้วยค่าตำแหน่งที่ส่งมาทางบัสหมายเลข 2 ข้อมูลจาก ROM จะถูกส่งลงไปยังบัสหมายเลข 3 ที่เรียกว่า Internal Data Bus แล้วนำไปเก็บไว้ที่ Instruction Register (เป็นวงจร Latch) เพื่อส่งต่อไปให้กับวงจร Timing and Control ทำการถอดรหัสแล้วควบคุมการทำงานส่วนอื่น ๆ ต่อไปแล้วแต่ว่าเป็นคำสั่งให้ทำงานอะไร ในกรณีที่เลือก ROM ภายนอก 8051 โดยป้อนสัญญาณลอจิก 1 เข้าไปที่ขา EA จะทำให้วงจร Timing and Control ส่งสัญญาณไปยังพอร์ท 0 และพอร์ท 2 เพื่อส่งค่าตำแหน่งหน่วย

ความจำบนบัสหมายเลข 2 ออกไปชี้หน่วยความจำภายนอก จากนั้นจะอ่านข้อมูลที่เป็นคำสั่งกลับเข้าทางพอร์ท 0 ไปยัง Internal Data Bus แล้วไปเก็บที่ Instruction Register เพื่อทำงานต่อไป เหมือนกับตอนอ่านคำสั่งจาก ROM ภายในการทำงานในช่วงส่งค่าตำแหน่งหน่วยความจำไปยังหน่วยความจำแล้วอ่านข้อมูลที่เป็นคำสั่งกลับเข้ามาเก็บไว้ใน Instruction Register เรียกว่าเป็นช่วงของการ Fetch (Fetch Cycle) ช่วงต่อไปจะเป็นช่วงของการทำงานตามคำสั่งเรียกว่า Execute Cycle เช่นถ้าเป็นคำสั่งให้บวกข้อมูลในรีจิสเตอร์ Accumulator กับข้อมูลจากหน่วยความจำ Data Memory ภายใน RAM ตำแหน่ง 23H วงจร Timing and Control ก็จะส่งสัญญาณให้ Instruction Registor ส่งค่าตำแหน่งหน่วยความจำ 23H ลงไปยัง Internal Data Bus แล้วนำข้อมูลไปเก็บไว้ที่ RAM ADDR Registor เพื่อใช้ชี้ตำแหน่งหน่วยความจำ RAM จากนั้น Timing and Control ก็จะสั่งให้ RAM ส่งข้อมูลที่เก็บอยู่ในหน่วยความจำตำแหน่ง 23H ลงมายัง Internal Data Bus แล้วนำข้อมูลไปเก็บไว้ที่ TMP1 (วงจร Latch) ขณะเดียวกันวงจร Timing and Control ก็จะส่งสัญญาณไปยัง ACC ให้ส่งข้อมูลมายัง TMP2 (วงจร Latch) วงจรALU ซึ่งโครงสร้างเป็นวงจรทำการคำนวณทางคณิตศาสตร์ (บวก,ลบ,คูณ,หาร) และยังสามารถทำงานทางลอจิก (AND,OR,NOT,XOR) จะทำการบวกเลขจาก TMP1 และTMP2 เข้าด้วยกัน ผลลัพธ์ที่ได้จะส่งผ่าน Internal Data Bus กลับไปเก็บยัง ACC PSW (Program Status Word) ซึ่งจะทำหน้าที่เก็บสถานะผลลัพธ์ของการทำงานใน ALU เช่น ผลลัพธ์การคำนวณมีค่าเกิน 8 บิท ก็จะให้บิทหนึ่งใน PSW ถูก SET เป็น 1

การทำงานที่กล่าวมาข้างต้นจะขึ้นกับสัญญาณควบคุมที่สร้างมาจากวงจร Timing and ConTrol และสัญญาณที่สร้างขึ้นนี้จะอ้างอิงกับสัญญาณนาฬิกาที่สร้างมาจากวงจร Oscillator ทำให้การทำงานต่าง ๆ เป็นไปตามลำดับที่ผู้ผลิตได้ออกแบบไว้ ดังในรูปที่ 2.13



รูปที่ 2.13 ลำดับสถานะการทำงานใน MCS-51

คำสั่งแต่ละคำสั่งของ 8051 จะใช้เวลา 1,2 หรือ 3 ไซเคิลของเครื่อง (Machine Cycle) แล้วแต่ว่าเป็นคำสั่งประเภทใด 1 ไซเคิลของเครื่องจะใช้เวลา 12 ไซเคิลของสัญญาณนาฬิกา ดังนั้น แต่ละคำสั่งของ 8051 จะใช้เวลาการทำงาน 12. 24 หรือ 36 ไซเคิลของสัญญาณนาฬิกานั่นเอง แต่ละไซเคิลของเครื่องจะถูกแบ่งออกเป็น 6 State คือ S1,S2,S3,S4,S5 หรือ S6 แต่ละ State จะประกอบด้วย 2 ไซเคิลของสัญญาณนาฬิกา ในไซเคิลแรกจะเรียกว่าเฟส 1 (P1) และไซเคิลที่ 2 เรียก เฟส 2 (P2) ในแต่ละเฟสจะนับตั้งแต่ขอบขาลงของสัญญาณนาฬิกาถึง ขอบขาลงของสัญญาณนาฬิกาที่อยู่ถัดไปดังรูปที่ 2.13 เมื่อ 8051 ทำงานเสร็จ 1 ไซเคิลของเครื่องก็จะเริ่มทำงาน State 1 Phase 1 (S1P1) ของไซเคิลต่อไป ใน 1 ไซเคิลของเครื่องวงจร Timing and Control จะสร้างสัญญาณ ALE ออกมา 2 ไซเคิลเพื่อ Fetch คำสั่งเข้าไป 2 ครั้งเสมอ ที่บริเวณขอบขาขึ้นของสัญญาณ ALE คำสั่งใดจะมีกี่ไบท์หรือใช้เวลาทำงานกี่ไซเคิลจะดูได้จากตารางชุดคำสั่ง 8051

คำสั่งประเภท 1 ไบท์ 1ไซเคิลของเครื่องได้แก่คำสั่ง INC A จะมีการอ่านคำสั่งจากหน่วยความจำสำหรับโปรแกรม 2 ครั้ง ที่เวลาประมาณขอบขาขึ้นของสัญญาณ ALE เมื่อคำสั่งแรกถูกอ่านเข้าไปที่เวลา ที่เวลาขอบขาขึ้นของสัญญาณ ALE แรก แล้วนำไปเก็บที่ Instruction Register เพื่อให้วงจร Timing and Control ถอดรหัสแล้วเข้าสู่การ Execute ขณะเดียวกันก็จะเริ่มต้นการ Fetch คำสั่งที่อยู่ในหน่วยความจำตำแหน่งถัดไปเข้ามาและคำสั่งที่ 2 จะถูกอ่านเข้ามาที่เวลาขอบขา

ขึ้นของสัญญาณ ALE ถัดไป วงจร Timing and Control เมื่อถอดรหัสคำสั่งแรกก็จะทราบว่าการทำงานคำสั่งนี้ให้สิ้นสุดจะใช้คำสั่ง เพียง 1 ไบท์ ดังนั้นคำสั่งที่ถูกอ่านเข้ามาไบท์ที่ 2 จะไม่ถูกนำมาทำงาน เพียงแต่อ่านเข้ามาแล้วทิ้งไป (Discard)

คำสั่งประเภท 2 ไบท์ และใช้เวลา 1 ไซเคิลของเครื่องได้แก่คำสั่ง ADD A,# data ในหนึ่งไซเคิลของเครื่องนี้จะมีการอ่านคำสั่งเข้ามา 2 ไบท์ เหมือนกับคำสั่งประเภท 1 ไบท์ 1 ไซเคิล ของเครื่อง แตกต่างกันที่ไบท์ที่ 2 จะถูกนำมาใช้งานด้วยไม่ได้ถูกทิ้งไปตัวอย่างของคำสั่ง ADDA, #33H จะเขียนเป็นภาษาเครื่องได้ 2 ไบท์ คือ 24 33 เมื่ออ่านคำสั่งไบท์แรกคือ 24 เข้าไปไว้ที่ Instruction Register แล้ว Timing and Control จะถอดรหัสพบว่าเป็นคำสั่งบวกเลข ก็จะส่งสัญญาณไปยัง Accumulator ให้เอาข้อมูลไปไว้ที่ TMP1 เมื่อคำสั่งที่ 2 ถูกอ่านเข้ามาที่ Instruction Register แล้ว Timing and Control จะสั่งให้เอาข้อมูลไบท์ที่ 2 ส่งลงไปยัง Internal Data Bus ไปเก็บยัง TMP1 จากนั้นวงจร ALU จะนำเอาข้อมูล TMP1 และ TMP2 มาบวกกัน ผลลัพธ์ที่ได้จะส่งออกจาก ALU ไปยัง Internal Data Bus แล้วไปเก็บไว้ที่ Accumulator

คำสั่งประเภท 1,2 หรือ 3 ไบท์ ที่ใช้เวลาทำงาน 2 ไซเคิลของเครื่องเช่นคำสั่ง INC DPTR จะมีการอ่านคำสั่งเข้าไป 4 ครั้งทุก ๆ ขอบขาขึ้นของสัญญาณ ALE ที่มี 2 ครั้งต่อ 1 ไซเคิลของเครื่อง ถ้าเป็นคำสั่งประเภท 1,2 หรือ 3 ไบท์ วงจร Timing and Control จะเอาคำสั่ง 1,2 หรือ 3 ไบท์แรกเท่านั้นไปทำงานส่วนคำสั่งที่เหลือทิ้งไป คำสั่ง 1 ไบท์ที่ใช้เวลาทำงาน 2 ไซเคิลของเครื่องที่กล่าวมาแล้วจะไม่รวมถึงคำสั่ง MOVX ซึ่งใช้ในการอ่านหรือเขียนข้อมูลกับหน่วยความจำ Data Memory ภายนอก การทำงานของคำสั่งนี้จะมีการ Fetch คำสั่งเข้าไป 2 ไบท์ในไซเคิลของเครื่องแรก ในไซเคิลของเครื่องที่ 2 จะไม่มีการ Fetch คำสั่งเข้าไป แต่จะเป็นช่วงเวลาของการอ่านหรือเขียนข้อมูลกับ Data memory ภายนอกสัญญาณ ALE ซึ่งปกติจะเปลี่ยนเป็น 1 ที่ S1P2 ก็จะไม่เปลี่ยนเป็น 1 ในไซเคิลของเครื่องที่ 2 โดยจะเป็น 0 อยู่จนกว่าจะถึงเวลา S4P2 ของไซเคิลของเครื่องที่ 2 สัญญาณ ALE จะเปลี่ยนเป็น 1 เพื่อทำการอ่านหรือเขียนข้อมูลกับ Data memory ภายนอก

บทที่ 3

ชุดคำสั่งของ 8051 (8051 Instruction)

ARTHIMATIC OPERATIONS

Mnemonic		Description
ADD	A.Rn	Add register to Accumulator
ADD	A.direct	Add direct byte to Accumulator
ADD	A,@R	Add indirect RAM to Accumulator
ADD	A,#data	Add immediate data to Accumulator
ADDC	A,Rn	Add register to Accumulator with Carry
ADDC	A.direct	Add direct byte to A with Carry flag
ADDC	A,@R	Add indirect RAM to A with Carry flag
ADDC	A,#data	Add immedate data to A with Carry flag
SUBB	A.Rn	Subtract register from A with Borrow
SUBB	A.direct	Subtract direct byte from A with Borrow
SUBB	A,@R	Subtract indirect RAM fromA with Borrow
SUBB	A,#data	Subtract immed data from A with Borrow
INC	A	Increment Accumulator
INC	Rn	Increment register
INC	direct	Increment direct byte
INC	@R	Increment indirect RAM
INC	DPTR	Increment Data Pointer
DEC	A	Decrement Accumulator
DEC	Rn	Decrement register
DEC	direct	Decrement direct byte
DEC	@R	Decrement indirect RAM
MUL	AB	Multipity A & B
DIV	AB	Divice A by B

รูปที่ 3.1 รหัสคำสั่งช่วยจำของ 8051

LOGICAL OPERATIONS

Mnemonic		Description
ANL	A.Rn	AND registor to Accumulator
ANL	A.direct	AND direct byte to Accumulator
ANL	A.@R	AND inderect RAM to Accumulator
ANL	A.#data	AND immediate data to Accumulator
ANL	direct.A	AND Accumulator to direct byte
ANL	direct.#data	AND immediate data to direct byte
ORL	A.Rn	OR registor to Accumulator
ORL	A.direct	OR direct btye to Accumulator
ORL	A.@R	OR indirect RAM to Accumulator
ORL	A.#data	OR immediate data to Accumulator
ORL	direct,A	OR Accumulator to direct byte
ORL	direct,#data	OR immediate data to direct byte
XRL	A.Rn	Exclusive-OR registor to Accumulator
XRL	A,direct	Exclusive-OR direct byte to Accumulator
XRL	A.@R	Exclusive-OR indirect RAM to A
XRL	A.#data	Exclusive-OR immadiate data to A
XRL	direct,A	Exclusive-OR Accumulator to direct byte
XRL	direct,#data	Exclusive-OR immediate data to direct
CLR	A	Clear Accumulator
CPL	A	Complement Accumulator
RL	A	Rotate Accumulator Left
RLC	A	Rotate A Left through the Carry flag
RR	A	Rotate Accumulator Right
RRC	A	Rotate A Right through Carry flag
SWAP	A	Swap rubbies within the Accumulator

รูปที่ 3.1 รหัสคำสั่งช่วยจำของ 8051 (ต่อ)

DATA TRANSFER

Mnemonic		Description
MOV	A.Rn	Move register to Accumulator
MOV	A.direct	Move direct byte to Accumulator
MOV	A.@R	Move indirect RAM to Accumulator
MOV	A.#data	Move immediate data to Accumulator
MOV	Rn.A	Move Accumulator to register
MOV	Rn.direct	Move direct byte to register
MOV	Rn.#data	Move immediate data to register
MOV	direct.A	Move Accumulator to direct byte
MOV	direct.Rn	Move register to direct byte
MOV	direct,direct	Move direct byte to direct
MOV	direct,@R	Move indirect RAM to direct byte
MOV	direct,#data	Move immediate data to direct byte
MOV	@Ri,A	Move Accumulator to indirect RAM
MOV	@Ri,direct	Move direct byte to indirect RAM
MOV	@Ri,#data	Move immediate data to indirect RAM
MOV	DPTR,#data 16	Load Data Pointer with a 16-bit constant
MOVC	A,@A+DPTR	Move Code byte relative to DPTR to A
MOVC	A,@A+PC	Move Code byte relative to PC to A
MOVX	A,@R	Move External RAM (8-bit addr) to A
MOVX	A,@DPTR	Move External RAM (16-bit addr) to A
MOVX	@Ri,A	Move A to External RAM (8-bit addr)
MOVX	@DPTR,A	Move A to External RAM (16-bit addr)
PUSH	direct	Push direct byte onto stack
POP	direct	Pop direct byte Form stack
XCH	A.Rn	Exchange register with Accumulator

รูปที่ 3.1 รหัสคำสั่งช่วยจำของ 8051 (ต่อ)

BOOLEAN VARIABLE MANIPULATION

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

PROGRAM AND MACHINE CONTROL

Mnemonic		Description
ACALL	addr 11	Absolute Subroutine Call
LCALL	addr 16	Long Subroutine Call
AJMP	addr 11	Absolute Jump
LJMP	addr 16	Long Jump
SJMP	rel	Short Jump (relative addr)
JMP	@A+DPTR	Jump indirect relative to the DPTR
JZ	rel	Jump if Accumulator is Zero
JNZ	rel	Jump if Accumulator is not Zero
JC	rel	Jump if Carry flag is set
JNC	rel	Jump if No Carry flag

รูปที่ 3.1 รหัสคำสั่งช่วยจำของ 8051 (ต่อ)

PROGRAM AND MACHINE CONTROL (cont.)		
Mnemonic		**Description**
JB	bit.rel	Jump if direct Bit Set
JNB	bit.rel	Jump if direct Bit Not Set
JBC	bit.rel	Jump if direct Bit is set & Clear bit
CJNE	A.direct.rel	Compare direct to A & Jump if Not Equal
CJNE	A.#data.rel	Comp. immed. to A & Jump if Not Equal
CJNE	Rn.#data.rel	Comp. immed. to reg. & Jump if Not Equal
CJNE	@Ri,#data.rel	Comp. immed. to ind. & Jump if Not Equal
DJNZ	Rn.rel	Decrement register & Jump if Not Zero
DJNZ	direct.rel	Decrement direct & Jump if Not Zero
NOP		No Operation

รูปที่ 3.1 รหัสคำสั่งช่วยจำของ 8051 (ต่อ)

3.1 ข้อมูลเฉพาะของชุดคำสั่ง

ในรูปที่ 3.1 เป็นการสรุปชุดคำสั่งของ 8051 (MCS-51 Instruction Set Desription)บล็อกที่ 1 ในรูปเป็นรหัสคำสั่งช่วยจำ (Mnemonic Code) ทั้งหมดของ 8051 และในบล็อกที่ 2 จะเป็นคำอธิบายโดยย่อของทำงานในแต่ละคำสั่งที่อยู่ทางซ้ายของคำอธิบาย ในการเขียนโปรแกรมภาษาแอสเซมบลีจะไม่นำเอาส่วนของบล็อกที่ 2 มาเขียน จะมีเฉพาะบล็อกที่ 1 เท่านั้น รหัสคำสั่งช่วยจำเป็นประโยคที่สามารถอ่านแล้วเข้าใจการทำงานได้โดยตรง ในรหัสคำสั่งแต่ละคำสั่งจะประกอบด้วยสองส่วนคือ

1. รหัสคำเนินการ (Operation Coad . OP-CODE) เป็นชุดของตัวอักษรที่บอกถึง การทำงานของไมโครโพรเซสเซอร์ รหัสคำเนินการนี้จะเป็นคำในภาษาอังกฤษความยาว 2 ถึง 4 ตัวอักษรและคำเหล่านี้มีความหมายที่สามารถจดจำได้ง่าย เช่น MOV มีความหมายเป็นคำสั่งสำหรับการเคลื่อนย้ายข้อมูลเหมือนคำภาษาอังกฤษที่ว่า MOVE ในไมโครโพรเซสเซอร์เบอร์อื่น เช่น Z-80 รหัสคำสั่งช่วยจำสำหรับการเคลื่อนย้ายข้อมูลจะใช้คำว่า LD ซึ่งมาจากคำว่า LOAD

2. ตัวถูกคำเนินการ (Operand) เป็นส่วนที่ 2 ของรหัสคำสั่งช่วยจำ ในส่วนนี้จะเป็นชุดของตัวอักษรที่บอกถึงส่วนที่ถูกคำเนินการเช่นรหัสคำสั่งช่วยจำของ 8051 ในรูปที่ 3.1 มีคำสั่งว่า

MOVE A.Rn Move register to Accumulator

จากตัวอย่างข้างบน OP-CODE เป็นสิ่งที่บอกการดำเนินการว่าเป็นการเคลื่อนย้ายข้อมูล และ Operand เป็นตัวบอกสิ่งที่ถูกดำเนินการ ซึ่งก็คือรีจิสเตอร์ A และรีจิสเตอร์ Bn ทั้งสองเป็นรีจิสเตอร์ของ 8051

ลักษณะการจัดวางข้อมูลใน Operand ของไมโครโพเซสเซอร์โดยทั่วไปสำหรับการกระทำระหว่างรีจิสเตอร์กับรีจิสเตอร์ หรือรีจิสเตอร์กับหน่วยความจำจะมีรูปแบบ

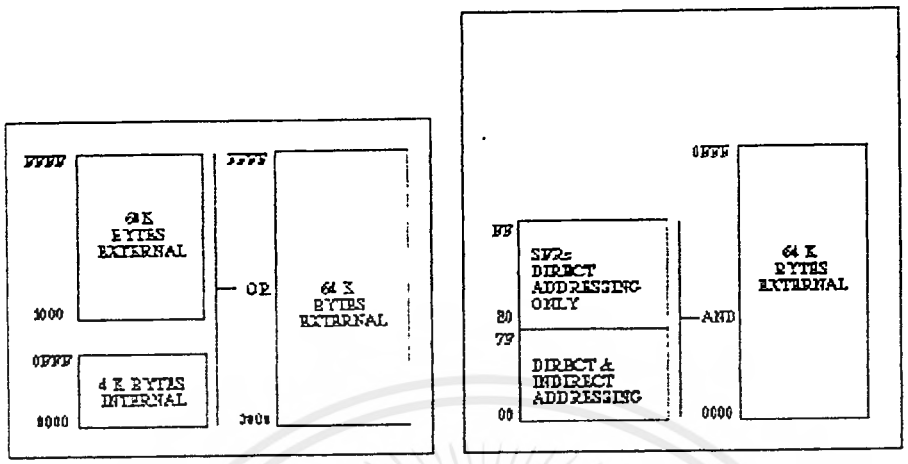
destination , source หรืออาจเป็น source , destination

ใน MCS-51 ก็จะใช้การจัด Operand ในแบบแรก คือข้อมูลที่อยู่ทางขวาของเครื่องหมาย คือ Source จะถูกนำไปทำงานแล้วเก็บไว้ในรีจิสเตอร์ หรือหน่วยความจำของ Destination ดังนั้นในตัวอย่างข้างบนจะเป็นการเคลื่อนย้ายข้อมูลจากรีจิสเตอร์ Rn ไปยังรีจิสเตอร์ A หรืออาจทำความเข้าใจการทำงานของคำสั่งนี้ได้จากคำอธิบายที่อยู่ทางขวาของคำสั่งก็ได้

การเขียนโปรแกรมภาษาแอสเซมบลีก็คือ การนำเอารหัสคำสั่งช่วยจำมาจัดเรียงกันเพื่อให้คอมพิวเตอร์ทำงานตามคำสั่งนั้น แต่ก่อนที่จำนำเอาโปรแกรมดังกล่าวไปให้คอมพิวเตอร์ทำงาน จะต้องเปลี่ยนเป็นรหัสคำสั่งช่วยจำให้เป็นภาษาเครื่องเสียก่อน รหัสคำสั่งช่วยจำแต่ละคำสั่งจะมีภาษาเครื่องต่างกันและไม่ซ้ำกัน

3.2 รีจิสเตอร์ของ 8051

หน่วยความจำของ 8051 แบ่งออกเป็น 2 แบบคือ หน่วยความจำสำหรับโปรแกรม (Program Area) และหน่วยความจำสำหรับเก็บข้อมูล (Data Area) ดังแสดงในรูปที่ 3.2



รูปที่ 3.2 ไดอะแกรมภาพของหน่วยความจำ 8051

หน่วยความจำสำหรับโปรแกรมเป็นหน่วยความจำที่ 8051 ใช้สำหรับเก็บโปรแกรมภาษาเครื่องที่ 8051 จะทำงานเมื่อเริ่มป้อนไฟเลี้ยงให้ 8051 หรือมีการรีเซ็ท (Reset) 8051 จะทำให้เริ่มการทำงานจากคำสั่งในหน่วยความจำที่ตำแหน่ง 0000H เมื่อทำงาน 1 คำสั่งก็จะทำให้รีจิสเตอร์ PC ที่ชี้ตำแหน่งโปรแกรมมีค่าเพิ่มขึ้นเพื่อชี้คำสั่งของตำแหน่งต่อไป ตำแหน่งสุดท้ายของหน่วยความจำคือ FFFFH หน่วยความจำสำหรับโปรแกรมนี้สามารถที่จะเลือกได้ว่าเป็นหน่วยความจำที่อยู่ภายใน 8051 หรือภายนอก 8051 ก็ได้ หน่วยความจำสูงสุดสำหรับโปรแกรมภายนอก 8051 มีได้ถึง 64 Kbyte ทำให้สามารถใช้งานได้อย่างกว้างขวาง หน่วยความจำในช่วงนี้ 8051 สามารถอ้างข้อมูลได้อย่างเดียว ไม่สามารถเขียนข้อมูลเข้าไปได้ระหว่างการทำงาน

หน่วยความจำสำหรับข้อมูลเป็นหน่วยความจำที่ 8051 ใช้สำหรับเก็บข้อมูลหรือพักข้อมูลระหว่างที่ทำงาน หน่วยความจำสำหรับข้อมูลมี 2 แบบ แบบที่หนึ่งมีขนาด 128 ไบท์อยู่ภายใน 8051 หน่วยความจำอีกแบบหนึ่งจะมีขนาด 64 กิโลไบท์ (Kbyte) ต้องต่อเพิ่มเติมเข้าไปภายนอก 8051 หน่วยความจำภายในตำแหน่ง 0 ถึง 7FH นี้สามารถอ้างถึงได้โดยตรงคือการมีให้อ่านหรือเขียนข้อมูลไปยังตำแหน่งนั้นได้โดยตรง แต่หน่วยความจำตำแหน่ง 80H ถึง FFH นั้น เป็นแบบรีจิสเตอร์ฟังก์ชั่นพิเศษ (Special Function Register , SFR) หน่วยความจำภายในช่วยนี้ใช้เป็นรีจิสเตอร์สำหรับงานเฉพาะอย่าง

หน่วยความจำสำหรับข้อมูลภายใน 8051 ช่วง 00H ถึง 07FH สามารถแบ่งออกได้เป็น 3 กลุ่ม

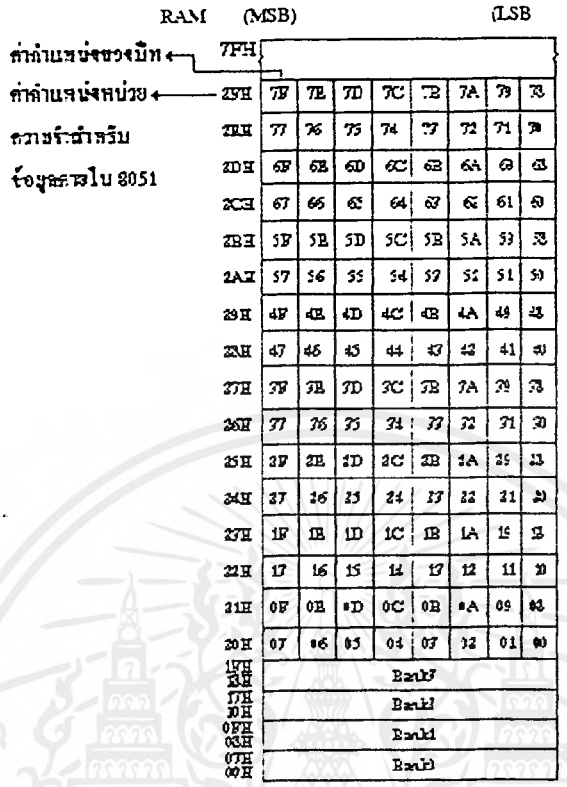
1. Register bank 0.3 อยู่ในหน่วยความจำช่วงตำแหน่งที่ 00H ถึง 1FH หน่วยความจำนี้สามารถแบ่งออกเป็น 4 ชุด ชุดละ 8 ไบท์ แต่ละชุดเราเรียกว่า BANK แต่ละไบท์ใน 1 BANK จะมีชื่อของรีจิสเตอร์ว่า R0 , R1 , R2 , R3 , R4 , R5 , R6 และ R7 รีจิสเตอร์เหล่านี้จะเรียกใช้งานในระหว่างการทำงานของโปรแกรมได้อย่างสะดวก และรีจิสเตอร์เหล่านี้จะเป็นชื่อซ้ำกันในทุก BANK การใช้งานจึงต้องเรียกใช้งานที่ละ BANK เท่านั้น โดยการกำหนดในรีจิสเตอร์ PSW โดยการกำหนดในรีจิสเตอร์ PSW ที่จะกล่าวถึงต่อไป เมื่อมีการ Reset การทำงานของ 8051 จะเริ่มการใช้งานรีจิสเตอร์ R0 ถึง R7 ที่ BANK 0 ซึ่งรีจิสเตอร์ R0 ถึง R7 ในแต่ละ BANK นั้นจะอ้างอิงในหน่วยความจำสำหรับข้อมูลภายใน 8051 ดังในตาราง

รีจิสเตอร์	ตำแหน่งหน่วยความจำ			
	BANK 0	BANK 1	BANK 2	BANK 3
R0	0	8	10	18
R1	1	9	11	19
R2	2	A	12	1A
R3	3	B	13	1B
R4	4	C	14	1C
R5	5	D	15	1D
R6	6	E	16	1E
R7	7	F	17	1F

ตัวอย่าง เมื่อกำลังมีการใช้งานในหน่วยความจำ BANK 1 และมีการอ้างถึงรีจิสเตอร์ R7 เช่นคำสั่ง MOV A,R7 (รหัสภาษาเครื่องคือ EFH)

การทำงานของคำสั่งนี้คือการเอาข้อมูลจากตำแหน่ง FH ของหน่วยความจำภายใน 8051 ไปไว้ยังรีจิสเตอร์ A นั่นเอง

2. Bit Address Area เป็นหน่วยความจำในช่วงตำแหน่ง 20H ถึง 2FH หน่วยความจำแต่ละบิท ในช่วงของหน่วยความจำดังกล่าวจะสามารถตรวจสอบหรือตั้งค่าเป็น 1 หรือ 0 ได้โดยการโปรแกรมภาษาเครื่อง แต่ละบิทของข้อมูลในหน่วยความจำของช่วงนี้จะมีค่าของตำแหน่งดังใน Memory Map รูปที่ 3.3 เช่น บิทที่ 7 ของหน่วยความจำในตำแหน่ง 2FH จะมีค่าตำแหน่งเป็น 7FH นั่นเอง



รูปที่ 3.3 ค่าตำแหน่งของแต่ละบิท

ในรูปที่ 3.3 ตัวเลขทางซ้ายเป็นค่าตำแหน่งของหน่วยความจำสำหรับข้อมูลภายใน 8051 ซึ่งในแต่ละบิทใน ตำแหน่งนั้นจะมีค่าเป็นเลขฐาน 16 ที่จะใช้เป็นค่าอ้างอิงในคำสั่งจัดการกับข้อมูลบิทนั้น

3. Scratched Pod Area เป็นช่วงของหน่วยความจำตั้งแต่ 30H ถึง 7FH หน่วยความจำช่วงนี้จะใช้สำหรับเก็บข้อมูลทั่วไป ถ้ารีจิสเตอร์ Stack pointer ชี้มายังหน่วยความจำ ช่วงนี้จะต้องระวังไม่ให้เกิดการเขียนทับของข้อมูลอันจะทำให้การทำงานของโปรแกรมผิดพลาดได้

จากตารางคำสั่ง จะมีคำอธิบายการใช้งานหรือการทำงานแต่ละคำสั่งใน 8051 ไว้ด้วยคำสั่งของ 8051 เป็นคำสั่งที่มีประสิทธิภาพการทำงานสูงมาก ในขณะที่ 8051 ทำงานจะมีรีจิสเตอร์ตัวหนึ่งที่เก็บภาวะ (Flag) ที่เกิดขึ้นระหว่างการคำนวณเช่น ตัวทด (Carry) หรือจะเลือกใช้ BANK ของรีจิสเตอร์ภายใน 8051 ก็ได้ รีจิสเตอร์นั้นคือ Program Status Word (PSW) มีขนาด 8 บิท แต่ละบิทจะใช้เก็บสถานะการทำงานต่างๆ ไว้ดังรูป 3.4

(MSB) (LSB)

CY	AC	F0	RS1	RS0	0V	-	P

Symbol	Posltion	Name and Significance
CY	PCW.7	Carry flag.
AC	PSW.6	Auxiliary Carry fiag. (For BCD operations)
F0	PSW.5	Flag 0 (Available to the user for general purposes)
RS1	PSW.4	Register bank select
RS0	PSW.3	control bits 1& 0. Set/ cleared by software to determine working register bank (see note)
OV	PSW.2	Overflow fiag.
-	PSW.1	User definable fiag.
P	PSW.0	Parity flag. Set/cleared by hardware each instruction cycle to indicate an odd/even number of one bits in the Accumulator, i.e.. even parity.

Note : The contents of (RS1 , RS0) enable the working register banks as follow :

(0.0)-Bank 0 (00H-07H)

(0.1)-Bank 1 (08H-0FH)

(1.0)-Bank 2 (10H-17H)

(1.1)-Bank 3 (18H-1FH)

รูปที่ 3.4 Program Status Word (PSW)

PSW.0 บิท 0 เรียกว่า บิทพาริตี้ บิทนี้จะบอกว่าในรีจิสเตอร์ Accumulator หรือรีจิสเตอร์ A มี 1 เป็นจำนวนคี่หรือคู่ เช่น ในรีจิสเตอร์ A ขนาด 8 บิทมี 1 อยู่ 3 ตัว และมี 0 อยู่ 5 ตัว ก็จะทำให้บิท PSW.0 นี้มีค่าเป็น 1 ถ้าใน Accumulator มี 1 อยู่เป็นจำนวนคู่ก็จะทำให้บิทนี้มีค่าเป็น 0

PSW.1 บิท 1 บิทนี้ไม่มีการใช้งาน

PSW.2 บิท 2 เรียกว่า Overflow Fiag เป็นบิทที่บอกการคำนวณนั้นทำให้เกิดตัวทดขึ้นในระหว่างการคำนวณ ตัวทดนี้เป็นตัวทดที่เกิดจากบิท 6 ไปยังบิท 7 มีประโยชน์เมื่อคำนวณแบบ Signed Integer

PSW.3, PSW.4 บิท 3 และ 4 2 บิทนี้จะใช้งานร่วนกันเพื่อเป็นตัวบอกว่าขณะนี้ใช้รีจิสเตอร์ R0 ถึง R7 ใน BANK ใดในตาราง

บิท 4 (RB1)	บิท 3 (RB0)	Register bank	address
0	0	0	00H-07H
0	1	1	08H-0FH
1	0	2	10H-17H
1	1	3	18H-1FH

ตัวอย่างเช่น บิท 4 และบิท 3 มีค่าเป็น 10_2 เป็นการเลือกรีจิสเตอร์ Bank 2 หมายความว่าในรหัสคำสั่งช่วยจำที่อ้างอิงถึง R0 ก็จะอ้างอิงหน่วยความจำภายในที่ตำแหน่ง 10H

PSW.5 บิท 5 เรียกว่าบิทอเนกประสงค์เป็นบิทที่ผู้ใช้สามารถใช้คำสั่งกำหนดค่าให้เป็น 0 หรือ 1 ก็ได้ โดยที่การทำงานของคำสั่งอื่นจะไม่ทำกับบิทนี้มีค่าเปลี่ยนแปลง บิทนี้มีประโยชน์สำหรับในการส่งสถานะของโปรแกรมระหว่างเรียกการทำงานของโปรแกรมย่อย (Subroutine)

PSW.6 บิท 6 เรียกว่า Auxiliary Carry Flag เป็นบิทที่ใช้สำหรับเก็บตัวทดที่เกิดขึ้นระหว่างการคำนวณ โดยตัวทดนี้เป็นตัวทดที่เกิดการคำนวณของบิท 3 ข้ามไปยังบิท 4

PSW.7 บิท 7 เรียกว่า Carry Flag เป็นบิทที่บอกสถานะการคำนวณทางคณิตศาสตร์ว่าผลลัพธ์นั้นทำให้เกิดตัวทดขึ้นหรือไม่ เช่น การบวกเลข 2 จำนวนเข้าด้วยกันแล้วผลลัพธ์มีค่ามากกว่า 255 ก็จะทำให้เกิดตัวทดขึ้น เนื่องมาจากว่า Accumulator ที่ทำการบวกนี้สามารถเก็บข้อมูลได้เพียง 8 บิทเท่านั้นและทำให้บิทนี้มีค่าเป็น 1

3.3 แอดเดรสซิ่ง (Addressing)

การติดต่อกับหน่วยความจำในชุดคำสั่งเรียกว่าการกำหนดที่อยู่ (Addressing) สามารถทำได้หลายวิธี

1. Direct Addressing เป็นการกำหนดตำแหน่งที่อยู่ของหน่วยความจำที่จะติดต่อเข้าไปใน Operand ของคำสั่งโดยตรง วิธีการนี้สามารถติดต่อหน่วยความจำสำหรับข้อมูลในตัวของ 8051 จำนวน 256 ตำแหน่งเท่านั้น เช่น DEC 20H

เป็นการลดข้อมูลของหน่วยความจำสำหรับข้อมูลภายใน 8051 ที่ตำแหน่ง 20H ลง 1 ถ้าเดิมข้อมูลในหน่วยความจำสำหรับข้อมูลภายใน 8051 ตำแหน่ง 20H มีค่า 11H เมื่อสิ้นสุดคำสั่งนี้

ค่าในหน่วยความจำจะเป็น 10H แต่ถ้าค่าเดิมเป็น FFH เมื่อทำคำสั่งนี้สิ้นสุดลงจะทำให้ข้อมูลเป็น 00H เพราะแต่ละตำแหน่งเก็บข้อมูลได้ 8 บิทเท่านั้น

2. Indirect Addressing เป็นการกำหนดที่อยู่ของหน่วยความจำสำหรับข้อมูลภายใน 8051 โดยอ้อม วิธีการระบุตำแหน่งของหน่วยความจำที่ต้องการติดต่อวิธีนี้ จะใช้รีจิสเตอร์ตัวหนึ่งเป็นตัวชี้ (Pointer) ไปยังหน่วยความจำที่ต้องการ รีจิสเตอร์ที่ใช้เป็นตัวชี้ได้แก่ R0 , R1 , DPTR เป็นต้น ใน Operand ของชุดคำสั่ง 8051 ที่ติดต่อกับหน่วยความจำโดยอ้อมจะมีสัญลักษณ์ @ นำหน้ารีจิสเตอร์ที่เป็นตัวชี้ เช่น DEC @R1

ตัวอย่างโปรแกรมภาษาแอสเซมบลี

```
MOV  R0,#10H
DEC  @R0
```

คำสั่งแรกเป็นคำสั่งกำหนดค่าให้กับรีจิสเตอร์ R0 โดยตรงให้มีค่าเท่ากับ 10H

คำสั่งที่สองเป็นคำสั่งลดค่า ในหน่วยความจำสำหรับข้อมูล ที่ชี้โดยรีจิสเตอร์ R0 ซึ่งจากคำสั่งแรกนั้น R0 มีค่า 10H ดังนั้นคำสั่งที่ 2 จึงเป็นการลดค่าในหน่วยความจำที่ตำแหน่ง 10H

3. Register instruction เป็นคำสั่งที่ใช้ติดต่อกับรีจิสเตอร์ R0 ถึง R7 ของรีจิสเตอร์ Bank ที่กำลังใช้งานอยู่ใน Operand จะมีชื่อของรีจิสเตอร์ที่ต้องการอยู่ เมื่อแปลเป็นภาษาเครื่องจะพบว่า ในภาษาเครื่องของคำสั่งติดต่อกับรีจิสเตอร์เหล่านี้จะมี 3 บิท ที่เป็นตัวบอกรีจิสเตอร์ R0 ถึง R7 ดังนั้นเมื่อคำสั่งนั้นถูกอ่าน (Fetch) เข้าไปประมวลผล (Execute) ก็จะแยกเอาตัวชี้รีจิสเตอร์ที่ต้องการมาจากคำสั่งภาษาเครื่อง (OP-CODE) ที่อ่านเข้าไปนั้นเอง เช่น MOV A,Rn โดยค่า Rn คือ R0 , R1 , ...,R7 คำสั่งนี้จะเปลี่ยนเป็นภาษาเครื่องได้ 1 ไบท์

4. Immediate Constant เป็นคำสั่งเกี่ยวกับค่าคงที่โดยตรง คำสั่งนี้จะมีการกำหนดส่วนของค่าคงที่ใน Operand เช่น MOV A,#10 คำสั่งนี้เป็นการกำหนดค่า 100 ไปเก็บในรีจิสเตอร์ A เครื่องหมาย # ใช้สำหรับบอกว่าค่าที่ตามหลังมาเป็นค่าคงที่ ตัวชี้ตำแหน่งหน่วยความจำคือรีจิสเตอร์ Program Counter ที่จะชี้ตำแหน่งของ Operand

5. Index Addressing เป็นการกำหนดเลขที่อยู่โดยดรรชนี การอ้างหน่วยความจำวิธีนี้ใช้ได้เฉพาะกับการติดต่อหน่วยความจำสำหรับโปรแกรมเท่านั้น ซึ่งจะใช้รีจิสเตอร์ DPTR หรือ Program Counter ขนาด 16 บิท บวกด้วยรีจิสเตอร์ A ขนาด 8 บิท แล้วนำผลลัพธ์ไปชี้ตำแหน่งหน่วยความจำโปรแกรมเพื่ออ่านข้อมูลออกมา คำสั่งนี้มีประโยชน์ในการอ่านข้อมูลที่เก็บไว้ในรูปแบบของตารางที่อยู่ในหน่วยความจำโปรแกรม เช่น MOVC A,@A+DPTR

3.4 ชุดคำสั่ง 8051

ชุดคำสั่ง 8051 แบ่งออกได้เป็น 5 กลุ่ม ซึ่งจะใช้ตารางเป็นตัวอธิบายโดยประกอบด้วย 4 คอลัมน์

คอลัมน์ 1 คือ Mnemonic เป็นช่องที่บอกถึงรหัสคำสั่ง

คอลัมน์ 2 คือ Operation เป็นการกระทำที่เกิดขึ้นตามรหัสคำสั่ง

คอลัมน์ 3 คือ Addressing Mode แบ่งเป็นส่วนย่อย ๆ คือ

Dir = Direct Addressing

Ind = Indirect Addressing

Reg = Register Addressing

Imm = Immediate Addressing

เครื่องหมาย X ในช่องนี้หมายความว่า รหัสคำสั่งมีเครื่องหมาย <byte> อยู่ในส่วน Operand สามารถอ้างอิงตำแหน่งของหน่วยความจำได้ด้วยวิธีดังกล่าว

Execution Time (μs) เป็นเวลาที่ใช้ในการทำคำสั่งนั้น มีหน่วยเป็นไมโครวินาที โดยจะใช้สัญญาณนาฬิกา 12 เมกกะเฮิรตซ์ ในการทำงานแต่ละคำสั่งจะใช้จำนวนรอบเครื่อง (Machinc- Cycle) เท่ากับ 1 , 2 หรือ 3 รอบเครื่อง 1 รอบจะใช้เวลาเท่ากับ 12 ไซเคิลของสัญญาณนาฬิกา ดังนั้นจะสามารถทราบเวลาที่ใช้ในการทำงาน 1 รอบเครื่องเท่ากับ $12/f_{clk}$ วินาที

3.4.1 คำสั่งคณิตศาสตร์ (Arithmetic Instruction)

8051มีคำสั่งการกระทำทางคณิตศาสตร์ที่สามารถบวก,ลบ,คูณและหารกันได้ซึ่งดีกว่าไมโครโพรเซสเซอร์เบอร์อื่น ที่ไม่มีคำสั่งการคูณและหาร คำสั่งสำหรับการกระทำทางคณิตศาสตร์มีดังรูป 3.5

Mnemonic	Operation	Addressing Modes				Execution
		Dir	Ind	Reg	Imm	Time(us)
ADD A,<byte>	A=A-<byte>	X	X	X	X	1
ADDC A,<byte>	A=A-<byte>-C	X	X	X	X	1
SUBB A,<byte>	A=A - <byte>-C	X	X	X	X	1
INC A	A=A-1	Accumulator only				1
INC <byte>	<byte>=<byte>-1	X	X	X	X	1
INC DPTR	DPTR=DPTR-1	Data Pointer only				2
DEC A	A=A-1	Accumulator only				1
DEC <byte>	<byte>=<byte>-1	X	X	X		1
MUL AB	B:A=BxA	ACC and B only				4
DIV AB	A=Int(A/B) B=Mod (A/B)	ACC and B only				4
DA A	Decimal Adjust	Accumulator only				1

รูปที่ 3.5 คำสั่งคณิตศาสตร์

3.4.2 คำสั่งทางตรรกศาสตร์ (Logical Instruction)

คำสั่งกลุ่มนี้มีการทำงานเหมือน Boolean Operation ซึ่งได้แก่ AND, OR, EXCLUSIVE-OR, NOT คำสั่งเหล่านี้ทำงานแบบบิทต่อบิท คือบิท 0-7 ของข้อมูลชุดที่ 1 ก็จะถูกกระทำกับ บิท 0-7 ของข้อมูลอีกชุดหนึ่งในลักษณะบิท 0 ต่อบิท 0, บิท 1 ต่อบิท 1 จนถึงบิท 7 ต่อบิท 7 รูปแบบของคำสั่งทางตรรกศาสตร์มีดังรูปที่ 3.6

Mnemonic	Operation	Addressing Modees				Execution Time (us)
ANL A,<byte>	A=A.AND.<byte>	X	X	X	X	1
ANL ,byte>,A	<byte>=<byte>.AND.A	X				1
ANL,byte.,#data	<byte>=<byte>.AND.#data	X				2
ORL A,<byte>	A=A.OR<byte>	X	X	X	X	1
ORL <byte>,A	<byte>=<byte>.ORA	X				1
ORL<byte>.#data	<byte>=<byte>.OR#data	X				2
XRL A,<byte>	A=A.XOR<byte>	X	X	X	X	1
XRL<byte>,A	<byte>=<byte>.XOR.A	X				1
XRL<byte>,#data	<byte>=<byte>.XOR#data	X				2
CLR A	A=00H	Accumulator only				1
CPL A	A=.NOT.A	Accumulator only				1
RL A	Rotate ACC Left 1 bit	Accumulator only				1
RLC A	Rotate Left through Carry	Accumulator only				1
RR A	Rotate Acc Right 1 bit	Accumulator only				1
RRC A	Rotate Right through Carry	Accumulator only				1
SWAP A	Swap Nibbles in A	Accumulator only				1

รูปที่ 3.6 คำสั่งทางตรรกศาสตร์

3.4.3 คำสั่งเคลื่อนย้ายข้อมูล (Data Transfer)

เป็นกลุ่มคำสั่งที่ใช้กันมาก ในคำสั่งเหล่านี้จะมีคำสั่งที่ใช้เคลื่อนย้ายข้อมูล จากหน่วยความจำตำแหน่งหนึ่งไปยังอีกหน่วยความจำอีกตำแหน่งหนึ่ง รูปแบบของคำสั่งกลุ่มนี้คือ

OP-CODE , Destination , Source

OP-CODE จะเป็นส่วนที่บอกการทำงานทั้งหมดของคำสั่งนี้ว่า จะเป็นการเคลื่อนย้ายข้อมูลทางเดียวหรือแลกเปลี่ยนข้อมูล

Source เป็นตำแหน่งของข้อมูลที่จะถูกเคลื่อนย้าย

Destination เป็นตำแหน่งของปลายทางที่จะใช้เก็บข้อมูล

ในการเคลื่อนย้ายข้อมูลแบบทางเดียวนั้นเมื่อสิ้นสุดการทำงาน ข้อมูลที่อยู่ในตำแหน่ง Source จะเปลี่ยนแปลง แต่ถ้าเป็นการแลกเปลี่ยนข้อมูลระหว่าง Source กับ Destination คำสั่งในกลุ่มนี้มีดังตารางรูปที่ 3.7

Mnemonic	Operation	Addressing Modees				Execution
		Dir	Ind	Reg	imm	Time (us)
MOV A,<scr>	A=<scr>	X	X	X	X	1
MOV <dest>,A	<dest>=A	X	X	X		1
MOV<dest>,<scr>	<dest>=<scr>	X	X	X	X	2
MOV PTR # data16	DPTR=16-bit immediate constant				X	2
PUSH <scr>	INC SP:MOV "@SP".<SCR>	X				2
POP <dest>	MOV<dest>. "@SP":DEC SP	X				2
XCH A,<byte>	ACC and <byte> exchange data	X	X	X		1
XCHD A,@Ri	ACC and @Ri exchange low nibbles		X			1

รูปที่ 3.7 คำสั่งเคลื่อนย้ายข้อมูล

คำสั่งเคลื่อนย้ายข้อมูลกับหน่วยความจำสำหรับข้อมูลภายนอก 8051

การติดต่อข้อมูลกับหน่วยความจำที่อยู่ภายนอก 8051 จะต้องใช้รีจิสเตอร์ RO,R1 หรือ DPTR เป็นตัวชี้ของตำแหน่งหน่วยความจำเท่านั้นการใช้ RO,R1 เป็นตัวชี้ตำแหน่งจะทำให้อ้างอิงตำแหน่งได้เพียง 256 ตำแหน่ง แต่การใช้ DPTR ชี้ตำแหน่ง ทำให้สามารถชี้ตำแหน่งได้ถึง 64*1024 ตำแหน่ง

คำสั่งสำหรับการติดต่อหน่วยความจำข้อมูลภานนอก8051มีเพียง 4 คำสั่งเท่านั้นดังตารางที่ 3.8

Address Width	Mnemonic	Operation	Execution Time (us)
8 bits	MOVX A, @Ri	Read external RAM @Ri	2
8 bits	MOVX @Ri,A	Write external RAM @Ri	2
16 bits	MOVXA, @DPTR	Read external RAM @DPTR	2
16 bits	MOVX @DPTR,A	Write external RAM @ DPTR	2

รูปที่ 3.8 คำสั่งเคลื่อนย้ายข้อมูลของหน่วยความจำสำหรับข้อมูลที่อยู่ภายนอก 8051

คำสั่งอ่านข้อมูลจากตาราง (Lookup Table)

คำสั่งในกลุ่มนี้จะมีเฉพาะการอ่านข้อมูลจากตารางซึ่งเป็นหน่วยความจำสำหรับโปรแกรมมาเก็บไว้ยัง Accumulator ตารางนี้จะต้องเขียนไว้ในหน่วยความจำตั้งแต่ขั้นตอนของการเขียนโปรแกรมสำหรับ 8051 อยู่แล้ว คำสั่งการอ่านข้อมูลจากตารางมีเพียง 2 คำสั่งดังตารางที่ 3.9

Mnemonic	Operation	Execution Time (us)
MOVC A, @A+DPTR	Read Pgm Memory at (A+DPTR)	2
MOVC A, @A+PC	Read Pgm Memory at (A+PC)	2

รูปที่ 3.9 คำสั่งอ่านข้อมูลจากตารางที่อยู่ในหน่วยความจำสำหรับโปรแกรม

3.4.4 คำสั่งบูลีน

คำสั่งของ 8051 นอกจากจะมีการกระทำทีละ 8 บิทแล้วหน่วยความจำสำหรับข้อมูลภายใน 8051 ยังสามารถติดต่อหรือมีการกระทำต่อข้อมูลที่ละบิท คือแต่ละบิทของรีจิสเตอร์ SRF แต่ในช่วงหน่วยความจำสำหรับข้อมูลตำแหน่ง 20H ถึง 2FH จำนวน 16 ไบท์ (มีจำนวนบิทที่สามารถติดต่อได้ 128 บิทหรือ 128 ตำแหน่ง) คำสั่งที่สามารถติดต่อกับหน่วนความจำดังกล่าวมีดังรูป 3.10

Mnemonic	Operation	Execution Time (us)
ANL C,bit	C=C.AND.bit	2
ANL C,/bit	C=C.AND..NOT.bit	2
ORL C,bit	C=C.OR.bit	2
ORL C,bit	C=C.ORNOT.bit	2
MOV C,bit	C=bit	1
MOV bit,C	bit=C	2
CLR C	C=0	1
CLR bit	bit=0	1
Mnemonic	Operation	Execution Time (us)
SETBC	C=1	1
SETB bit	bit=1	1
CPL C	C=.NOT.C	1
CPL bit	bit=.NOT.bit	1
JC rel	Jump ifC=1	2
JNC rel	Jump if C=0	2
JB bit,rel	Jump if bit=0	2
JNB bit,rel	Jump if bit=0	2
JBC bit,rel	Jump if bit=1:CLR bit	2

รูปที่ 3.10 คำสั่งบูลีน

คำสั่ง ANL,ORL,MOV,CRL จะมีการกระทำเหมือนกับในกลุ่มคำสั่งตรรกศาสตร์ทุกประการ แตกต่างกันที่การอ้างอิงตำแหน่งหน่วยความจำคำสั่ง ในตารางรูปที่ 3.10 จะมีคำว่า bit ปรากฏอยู่หมายความว่าที่ตำแหน่งนั้นของ operand จะต้องป้อนค่าตำแหน่งหน่วยความจำที่เป็นแบบบิท เช่น บิท 0 ของหน่วยความจำตำแหน่ง 20H จจะมีค่าหน่วยความจำแบบบิทเท่ากับ 00H และบิท 7 ของหน่วยความจำตำแหน่ง 20H จะมีตำแหน่งแบบหน่วยความจำเท่ากับ 07H

3.4.5 กลุ่มคำสั่งกระโดดข้าม (Jump Intruction)

กลุ่มคำสั่งกระโดดข้ามคือ กลุ่มของคำสั่งที่ทำให้การทำงานของโปรแกรมข้ามไปยังตำแหน่งที่กำหนดโดยไม่ต้องทำตามลำดับของโปรแกรมเดิม คำสั่งกระโดดข้ามแบบไม่มีเงื่อนไขมีดังตารางที่ 3.11

Memonic	Operation	Execution Time (us)
JMP addr	Jump to addr	2
JMP @A+DPTR	Jump to A+DPTR	2
CALL addr	Call subroution at addr	2
RET	Return from subroution	2
RETI	Return from interrupt	2
NOP	No operation	1

รูปที่ 3.11 คำสั่งกระโดดข้าม

คำสั่งข้ามการทำงานแบบมีเงื่อนไข (Conditioned Jump Instruction)

คำสั่งให้ข้ามการทำงานข้างบนนั้นเป็นการข้ามแบบไม่มีเงื่อนไข คือ ไม่ต้องมีการตรวจสอบใด ๆ ก่อนจะข้ามการทำงาน คำสั่งอีกกลุ่มหนึ่งจะต้องตรวจสอบเงื่อนไขก่อนการข้ามการทำงานมีคำสั่งดังนี้

Mnemonic	Operation	addressing Modes				Execution Tine(μs)
		Dir	Ind	Reg	Im m	
JZ rel	Jump if A=0	Accumulator only				2
JNZ rel	Jump if A=0	Accumulator only				2
DJNZ<byte>,rel	Decrement and jump if not zero	X		X		2
CJNE A,<byte>,rel	Jump if A=<byte>	X			X	2
CJNE<byte>,# data,rel	Jump if<byte>=#data		X	X		2

รูปที่ 3.12 คำสั่งข้ามแบบมีเงื่อนไข

บทที่ 4
8255 พอร์ทข้อมูลแบบขนาน

4.1 ไอซี 8255

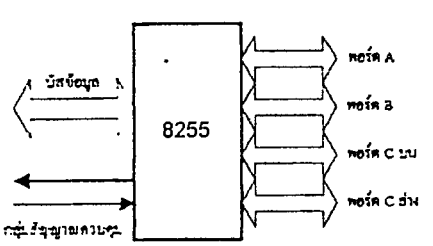
การใช้งานไมโครโพเซสเซอร์ จะต้องเชื่อมต่อกับอุปกรณ์ภายนอก เช่น สวิทช์รีเลย์ หรือตัวตรวจจับอื่นๆ การเชื่อมต่อในลักษณะดังกล่าวจะเชื่อมต่อกับพอร์ทอินพุทเอาต์พุท เพื่อให้ไมโครโพเซสเซอร์ส่งสัญญาณควบคุมไปยังอุปกรณ์ต่างๆ ตามเงื่อนไขที่เกิดขึ้นและสามารถตรวจสอบได้ด้วยไมโครโพเซสเซอร์เอง

การเชื่อมต่อกับพอร์ทอินพุทที่ง่ายที่สุดคือ การเชื่อมต่อโดยตรงโดยใช้ลอจิกเกต 3 สถานะ โดยสัญญาณควบคุมพอร์ทอินพุทจะเป็นตัวไปเปิดเกตให้ข้อมูลเข้าสู่บัสและไมโครโพเซสเซอร์จะอ่านเข้าไป แต่สำหรับพอร์ทเอาท์พุทจะใช้เลตช์ฟลิปฟลอป ทำหน้าที่รับสัญญาณข้อมูลจากไมโครโพเซสเซอร์ที่ส่งเข้าไปในบัสข้อมูลและได้รับการจับไว้ที่พอร์ทในขณะที่มีสัญญาณควบคุมพอร์ททริกมาที่ขาแลตช์

พอร์ทอินพุทเอาท์พุทที่ใช้เกตขนาดเล็กดังกล่าว ยังมีจุดอ่อนในเรื่องของจำนวนไอซีซึ่งอาจจะต้องใช้หลายชิป (ถ้าต้องการหลายพอร์ท) และยากที่กำหนดลักษณะการทำงานให้แตกต่างไปจากจำนวนวงจรเดิมที่ออกแบบไว้ บริษัทที่ออกแบบไมโครโพเซสเซอร์ที่ออกแบบส่วนใหญ่จึงออกแบบ LSI เพื่อทำหน้าที่เป็นพอร์ทอินพุทเอาท์พุทของระบบ ซึ่งมีข้อดีในเรื่องการใช้งานได้ง่าย ในบทนี้จะกล่าวถึงการประยุกต์ใช้ไอซี LSI ที่ทำหน้าที่เป็นพอร์ทอินพุทเอาท์พุท ไอซี LSI ที่รู้จักกันดีมากที่สุด มีราคาถูก และหาได้ง่าย คือ ไอซี 8255 ของบริษัทอินเทล

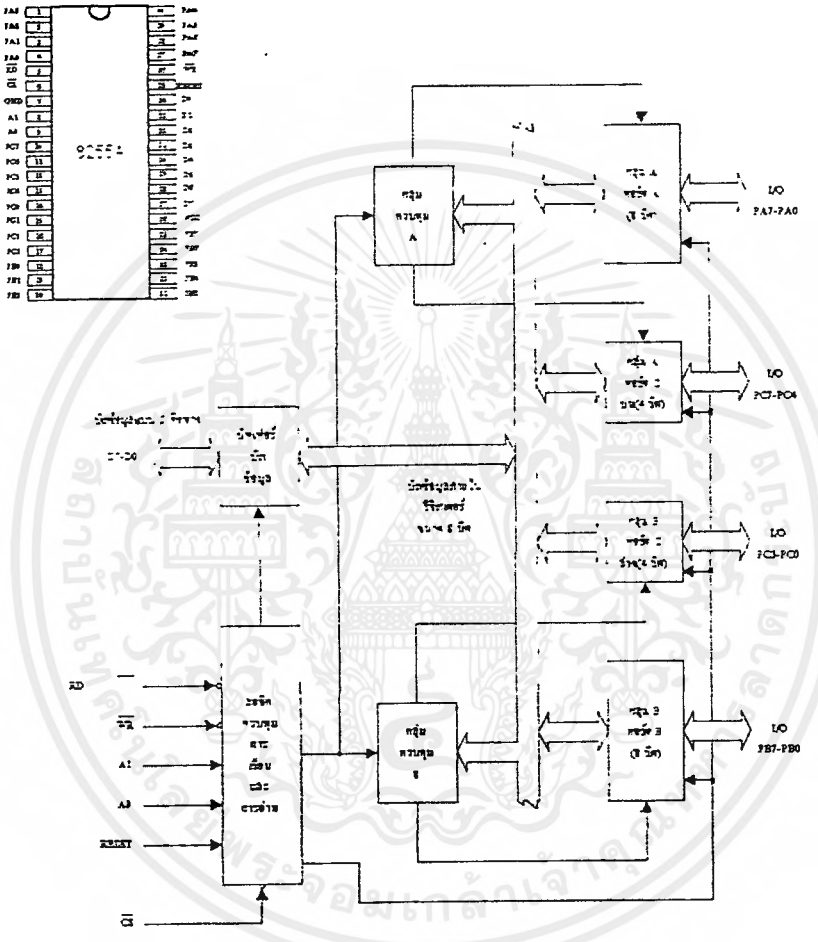
4.2 โครงสร้าง 8255

8255 เป็นไอซีที่มีขา 40 ขา เป็นไอซีที่ต่อเป็นพอร์ทให้ไมโครโพเซสเซอร์ได้ 3 พอร์ท โดยมีโครงสร้างพื้นฐานแสดงได้ดังรูป 4.1



รูป 4.1 แผนผังโครงสร้างของไอซี 8255

การเรียกพอร์ทของ 8255 จะเรียกพอร์ทต่างๆว่า พอร์ท A พอร์ท B และพอร์ท C โดยพอร์ท C แยกเป็น 2 ส่วนคือ พอร์ท C ล่างหรือตั้งแต่ PC_0 - PC_3 มีจำนวน 4 บิท และพอร์ท C บน หรือตั้งแต่ PC_4 - PC_7 ที่พิเศษคือ พอร์ททุกพอร์ทเป็นได้ทั้งพอร์ทอินพุทและเอาท์พุท



รูปที่ 4.2 แผนผังวงจรภายในและการจัดขาของไอซี 8255

รูปที่ 4.2 เป็นแผนผังภายในของไอซีและการจัดวางขาของไอซี 8255 การทำงานของวงจรจะใช้สัญญาณควบคุมจากไมโครโพเซสเซอร์มาควบคุมการทำงาน โดยไมโครโพเซสเซอร์จะส่งคำสั่งมาโปรแกรมการทำงานหรือกำหนดรูปแบบของพอร์ทให้เป็นอินพุทหรือเอาท์พุทได้

4.3 ขาต่าง ๆ ของ 8255

เพื่อให้เข้าใจวิธีการต่อใช้งานระหว่าง ไมโครโพเซสเซอร์ กับ 8255 จึงจำเป็นต้องเข้าใจความหมายและตำแหน่งขาต่างๆ เสียก่อน ขาทั้ง 40 ขาของไอซีประกอบด้วย

D_0-D_7 เป็นขาที่ข้อมูลอินพุทเอาท์พุทจะต้องผ่านเข้าออกจากส่วนนี้ D_0-D_7 จึงต่อเข้ากับระบบบัสของไมโครโพเซสเซอร์ เพื่อให้ไมโครโพเซสเซอร์สามารถอ่านหรือเขียนข้อมูลจากพอร์ทผ่านทางบัสนี้ได้

CS (สัญญาณเลือกชิป) ขานี้เป็นขาอินพุทที่จะได้รับสัญญาณจากภายนอกเพื่อเลือกชิป 8255 โดยเมื่อขานี้เป็น "0" จะทำให้ 8255 ต่อเข้ากับระบบบัสของไมโครโพเซสเซอร์ เพื่อให้ไมโครโพเซสเซอร์เขียนหรืออ่านข้อมูลจากพอร์ทได้

RD (สัญญาณการอ่าน) เป็นสัญญาณอินพุทที่ต้องส่งมาจากซีพียูเมื่อสัญญาณที่ขานี้เป็น "0" และสัญญาณ CS เป็น "0" ด้วย ไอซี 8255 จะทำตัวให้ซีพียูอ่านข้อมูลจากบัสในขณะที่เป็นพอร์ทอินพุท

WR เป็นสัญญาณการเขียน จะแอคทีฟเมื่อมีสัญญาณ WR และสัญญาณ CS เป็น "0" สัญญาณนี้จะมาจากซีพียูเมื่อต้องการเขียนข้อมูลลงบนพอร์ทที่กำหนด

A_0 - A_7 (สัญญาณแอดเดรส) ลอจิกของสัญญาณทั้งสองจะถอดรหัสออกเป็น 4 รหัส เพื่อกำหนดรีจิสเตอร์ภายในที่เชื่อมต่อกับพอร์ทอินพุทเอาท์พุทของ 8255

RESET (สัญญาณรีเซต) เป็นสัญญาณที่ส่งจากภายนอกเข้ามาทำการรีเซต 8255 เพื่อเคลียร์สถานะต่างๆ ของ 8255 เมื่อ 8255 ได้รับการรีเซต ก็จะกลับเข้าสู่โหมดอินพุทหรือทุกพอร์ทที่เป็นพอร์ทอินพุท

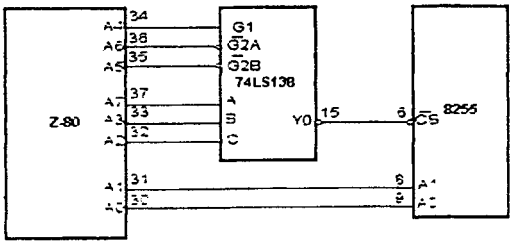
PA_0 - PA_7 เป็นสายสัญญาณที่เป็นพอร์ทของ 8255 ที่ชื่อพอร์ท A การเลือกชิปพอร์ทจะเลือกโดยสัญญาณแอดเดรส A_0 - A_1

PB_0 - PB_7 เป็นสายสัญญาณที่เป็นพอร์ท B ของ 8255 ซึ่งถูกเลือกโดยสัญญาณแอดเดรส A_0 - A_1

PC_0 - PC_7 เป็นสายสัญญาณที่เป็นพอร์ท C ของ 8255 การกำหนดพอร์ทนี้ จะได้รับการกำหนดโดยสัญญาณแอดเดรส A_0 - A_1 พอร์ท C นี้จะแบ่งออกเป็น 2 กลุ่ม คือ กลุ่ม PC_0 - PC_3 และกลุ่ม PC_4 - PC_7

4.4 การเชื่อมต่อ 8255 กับ ไมโครโพเซสเซอร์

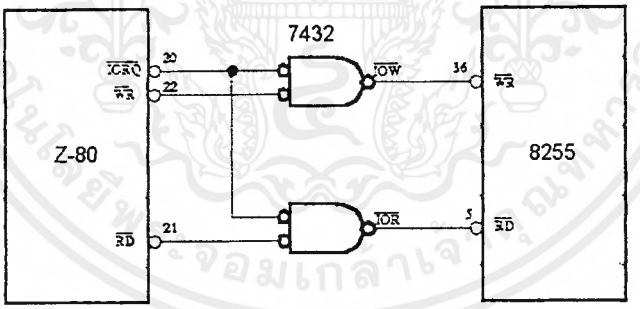
หากพิจารณาจากขาของ 8255 จะเห็นว่า ส่วนขาควบคุมที่จะเชื่อมต่อเข้ากับบัสของไมโครโพเซสเซอร์สามารถเชื่อมต่อกับบัสได้ง่ายในที่นี้จะทดลองต่อ 8255 เป็นพอร์ทให้กับ Z-80 สมมุติว่าต้องการให้ Z-80 มองเห็น 8255 เป็นพอร์ทหมายเลข 10H , 11H , 12H และ 13H การเชื่อมต่อสายสัญญาณการเลือกแอดเดรสของพอร์ทแสดงดังรูป 4.3



รูป 4.3 การกำหนดแอดเดรสให้กับ 8255

สังเกตว่า ขณะสัญญาณ CS แอคตีฟนั้น สัญญาณแอดเดรส A_2 ถึง A_7 จะต้องมีข้อมูล 000100 และเมื่อรวมกับ A_1 , A_0 จะเป็น 000100XX พอร์ทที่เกิดขึ้นเมื่อ A_1 , A_0 เป็น 00 คือพอร์ท 10H และถ้า A_1 , A_0 เป็น 11 พอร์ทจะเป็น 13H การกำหนดพอร์ทของ Z-80 จะใช้ข้อมูลบนบัสแอดเดรส 8 เส้นคือ A_7 - A_0 เท่านั้น

สัญญาณที่จะควบคุม 8255 อีกชุดหนึ่งคือ สัญญาณควบคุมการเขียนและการอ่าน หากสัญญาณ WR แอคตีฟเป็น "0" จะหมายถึง การเขียนพอร์ทหรือการส่งข้อมูลให้พอร์ทเอาท์พุท แต่ถ้าสัญญาณ RD แอคตีฟเป็น "0" จะหมายถึงการอ่านพอร์ทหรือรับข้อมูลอินพุท

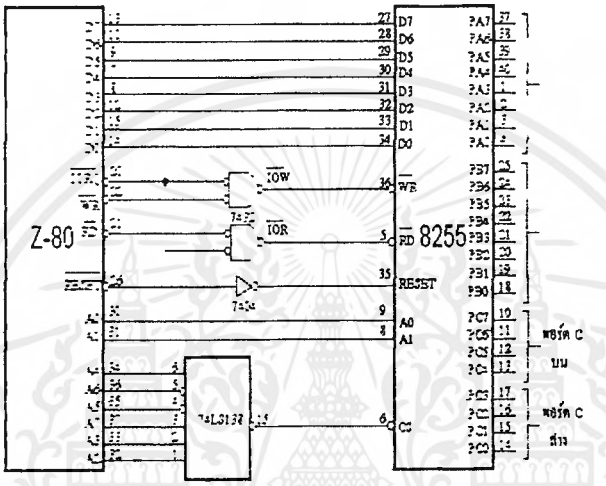


รูปที่ 4.4 วงจรการเชื่อมต่อสายสัญญาณควบคุมการเขียนและการอ่าน 8255

เพื่อให้แยกกันระหว่างการเขียนและการอ่านหน่วยความจำกับการเขียนและการอ่านพอร์ทอินพุทเอาท์พุท จึงต้องใช้สัญญาณ IORQ ร่วมด้วย คือถ้าสัญญาณ WR เกิดขึ้นพร้อมสัญญาณ IORQ จะหมายถึง สัญญาณ IOW หรือสัญญาณเขียนพอร์ทและถ้าให้สัญญาณ IORQ แอคตีฟพร้อมสัญญาณ RD จะหมายถึงสัญญาณ IOR หรือสัญญาณอ่านพอร์ท ซึ่งการเชื่อมต่อสายสัญญาณควบคุมการเขียนและการอ่านพอร์ทแสดงไว้ดังรูป 4.4

เมื่อเชื่อมต่อเป็นระบบ จะต้องมีการเชื่อมสายสัญญาณ RESET ของ Z-80 มายังขา RESET ของ 8255 การรีเซตของ 8255 ใช้ "1" ซึ่งตรงข้ามกับ Z-80 ก็เนื่องจากว่า ขณะที่ Z-80

รีเซต เราจะเริ่มจากให้พอร์ททุกพอร์ทของ 8255 เป็นอินพุท เพื่อว่าอาจมีข้อมูลบางส่วนไปออกที่พอร์ทเอาท์พุทขณะที่เราไม่ต้องการ ซึ่งอาจทำให้ระบบอินเตอร์เฟสภายนอกมีปัญหาได้ เพราะเราไม่รู้สถานะที่แน่นอนของ 8255 ก่อนการโปรแกรมโหมดการทำงาน ระบบการเชื่อมต่อของ 8255 กับ Z-80 หรือไมโครโพเซสเซอร์เบอร์อื่นๆ ทั้งระบบแสดงได้ดังรูป 4.5



รูปที่ 4.5 การเชื่อมต่อ 8255 กับ ไมโครโพเซสเซอร์ทั่วไป

4.5 รีจิสเตอร์ภายในของ 8255

เมื่อต่อ 8255 เข้ากับไมโครโพเซสเซอร์ได้แล้ว สิ่งที่ผู้ใช้จะต้องทำคือ การโปรแกรมให้ 8255 ทำงานตามที่ต้องการ จากการที่ 8255 มีพอร์ที่ไมโครโพเซสเซอร์มองเห็น 4 พอร์ท แต่ละพอร์ทจะเสมือนเป็นรีจิสเตอร์ที่สามารถเขียน และอ่านได้ รีจีสเตอร์แต่ละตัวจึงถูกกำหนดด้วยแอดเดรสตามที่ตั้งไว้ เช่น ในกรณีที่เป็นแอดเดรส 10H , 11H , 12H และ 13H รีจิสเตอร์แต่ละตัวจะได้รับการกำหนดควบคู่กับสัญญาณ RD และ WR เพื่อแสดงความหมายตัวอย่างเช่น พอร์ท10H เป็นพอร์ท A ซึ่งเมื่อเขียนที่พอร์ทนี้ จะเป็นการส่งข้อมูลเอาท์พุท และถ้าอ่านพอร์ทนี้ก็จะเป็นการอินพุทข้อมูลจากพอร์ท ดังนั้นสัญญาณของขาควบคุมที่ประกอบกันจะแสดงความหมายต่างๆ ดังตารางที่ 4.1

RD	WR	A1	A0	ความหมาย
1	0	0	0	เขียนพอร์ท A ซึ่งเป็นข้อมูล
0	1	0	0	อ่านพอร์ท A ซึ่งเป็นข้อมูล
1	0	0	1	เขียนพอร์ท B ซึ่งเป็นข้อมูล
0	1	0	1	อ่านพอร์ท B ซึ่งเป็นข้อมูลP
1	0	1	0	เขียนพอร์ท C ซึ่งเป็นข้อมูล
0	1	1	0	อ่านพอร์ท C ซึ่งเป็นข้อมูล
1	0	1	1	เขียนข้อมูล ซึ่งเป็นรหัสความคุม
0	1	1	1	อ่านเข้ามา ซึ่งไม่มีความหมาย

ตารางที่ 4.1 สัญญาณควบคุมการกระทำของ 8255

การใช้งาน 8255 จะต้องส่งรหัสควบคุม (Control Coad) เข้าไปยังพอร์ทข้อมูลควบคุม เพื่อควบคุมการทำงานของ 8255 โดยใช้สัญญาณควบคุมพอร์ทหมายเลข 13H การควบคุมการทำงานของ 8255 มีหลายโหมด แต่ละโหมดจะแตกต่างกันออกไป การโปรแกรมให้ 8255 ทำงานจะทำได้ 3 โหมดคือ โหมด 0 โหมด 1 โหมด 2

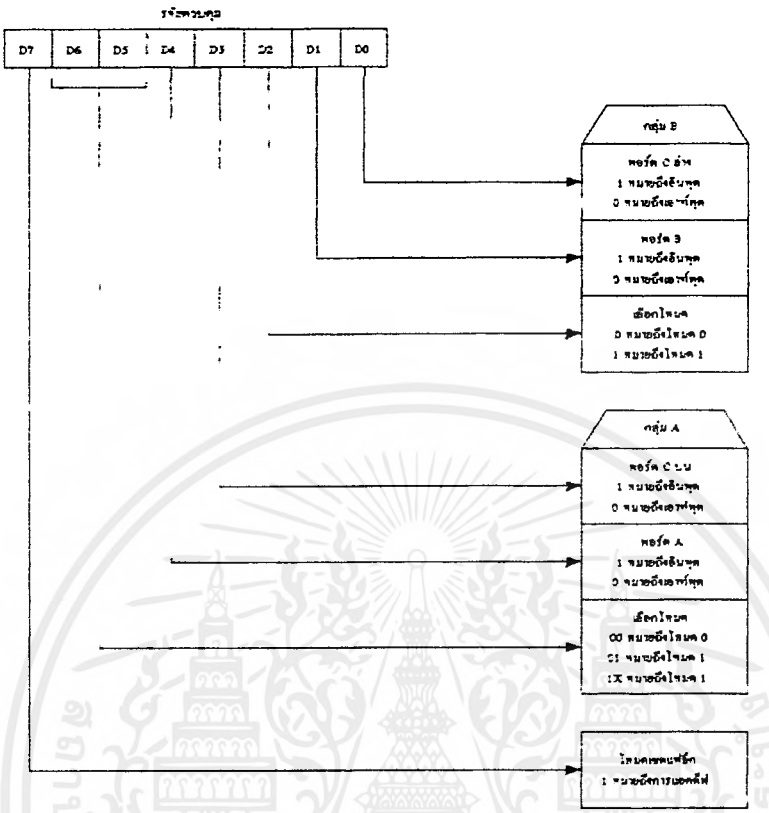
4.6 โหมด 0 หรืออินพุทเอาท์พุทแบบพื้นฐาน

การกำหนดโหมดการทำงาน จะต้องส่งข้อมูลคำสั่งเข้าไปโปรแกรมในพอร์ทควบคุมของ 8255 ซึ่งในที่นี้ใช้พอร์ทหมายเลข 13H (ตามรูปที่ 4.3) แต่ละบิทของข้อมูลที่ส่งไปจะมีความหมายในตัวเอง ลักษณะความหมายของแต่ละบิทในรหัสควบคุมแสดงได้ดังรูปที่ 4.6

การโปรแกรม 8255 คือ การให้ค่ารหัสบิทต่างๆ เข้าไปในรหัสการควบคุมแล้วส่งไปยังรีจิสเตอร์ของพอร์ทควบคุม ความหมายของบิทต่าง ๆ มีดังนี้

บิท D_7 เป็นบิทที่แสดงรหัสคำสั่งควบคุม ถ้าบิทนี้เป็น “1” หมายถึงรหัสควบคุมนี้จะมีผลต่อการเปลี่ยนแปลงการเซตโหมดต่าง ๆ ของ 8255

บิท D_6 และ D_5 เป็นการเลือกโหมดของพอร์ท A ซึ่งมี 3 โหมดคือ โหมด 0 โหมด 1 และโหมด 2 ดังแสดงในรูปที่ 4.6



รูปที่ 4.6 ความหมายของบิทต่างๆ ในรหัสควบคุม

บิท D_4 ถ้ามีค่าเป็น "0" หมายถึงการกำหนดพอร์ท A เป็นเอาท์พุท ถ้ามีค่าเป็น "1" จะหมายถึงการกำหนดให้พอร์ท A เป็นอินพุท

บิท D_3 เป็นบิทที่บอกถึงการเซตของพอร์ท C บน ถ้าเป็น "0" จะทำให้พอร์ท C บนเป็นเอาท์พุท

บิท D_2 เป็นบิทที่บอกถึงภาวะการเซตโหมดของพอร์ท B ถ้าเป็น "0" หมายถึงการเลือกพอร์ท B เป็นโหมด 0 และถ้าเป็น "1" หมายถึงการเลือกโหมด 1

บิท D_1 เป็นการกำหนดอินพุทเอาท์พุทของพอร์ท B ถ้าเป็น "0" หมายถึงเอาท์พุท ถ้าเป็น "1" หมายถึงอินพุท

บิท D_0 เป็นการกำหนดอินพุทเอาท์พุทของพอร์ท C ล่าง ถ้าเป็น "0" หมายถึงเอาท์พุท ถ้าเป็น "1" หมายถึงอินพุท

4.7 การทำงานในโหมด 0

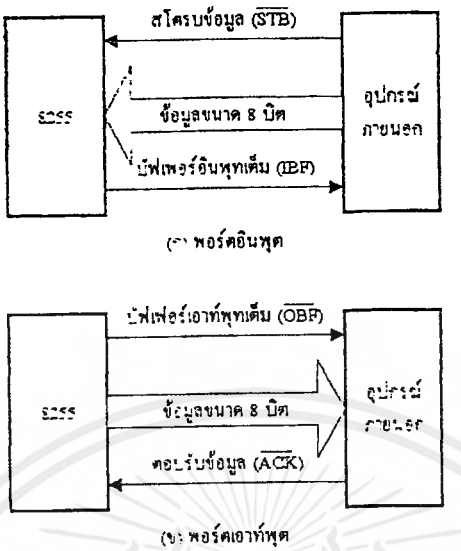
โหมด 0 เป็นโหมดที่กำหนดให้พอร์ททุกพอร์ทบนตัว 8255 เป็นพอร์ทอินพุทเอาท์พุทแบบพื้นฐาน รูปแบบความเป็นไปได้จึงมีทั้งสิ้น 16 รูปแบบตามลักษณะของพอร์ท A พอร์ท B พอร์ท C บนและพอร์ท C ล่าง ลักษณะของรหัสควรคุมแต่ละแบบจึงเป็นดังตาราง 4.2

รหัสควบคุม	D_7.....D_0	PORT A	PORT B	PORT C (PC_7-PC_4)	PORT C (PC_3-PC_0)
0	10000000	OUT	OUT	OUT	OUT
1	10000001	OUT	OUT	OUT	IN
2	10000010	OUT	IN	OUT	OUT
3	10000011	OUT	IN	OUT	IN
4	10001000	OUT	OUT	IN	OUT
5	10001001	OUT	OUT	IN	IN
6	10001010	OUT	IN	IN	OUT
7	10001011	OUT	IN	IN	IN
8	10010000	IN	OUT	OUT	OUT
9	10010001	IN	OUT	OUT	IN
10	10010010	IN	IN	OUT	OUT
11	10010011	IN	IN	OUT	IN
12	10011000	IN	OUT	IN	OUT
13	10011001	IN	OUT	IN	IN
14	10011010	IN	IN	IN	OUT
15	10011011	IN	IN	IN	IN

ตารางที่ 4.2 ลักษณะของรหัสควบคุมแบบต่างๆในโหมด 0

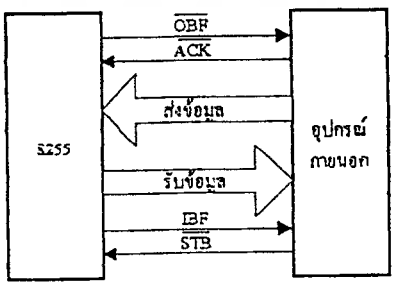
4.8 การทำงานของ 8255 ในโหมด 1

การทำงานของ 8255 ในโหมด 1 เป็นโหมดที่ทำให้อินพุทเอาท์พุทมีการตรวจสอบสัญญาณ (handshaking) โดยใช้อินพุทเอาท์พุทของพอร์ท A และพอร์ท B เป็นหลัก และใช้พอร์ท C บนเป็นตัวตรวจสอบสัญญาณ (handshake) ของพอร์ท A ส่วนพอร์ท C ล่าง เป็นตัวตรวจสอบสัญญาณของพอร์ท B การจัดสัญญาณต่าง ๆ เหล่านี้แสดงได้ดังรูป 4.7



รูปที่ 4.7 โครงสร้างของตัวตรวจสอบสัญญาณของพอร์ทอินพุทและพอร์ทเอาท์พุท

แนวความคิดการใช้พอร์ทอินพุทเอาท์พุทโดยมีตัวตรวจสอบสัญญาณก็เพื่อให้มีการซิงโครไนซ์ระหว่างอุปกรณ์ภายนอกที่ทำงานได้ช้ากว่าการทำงานของคอมพิวเตอร์ที่ทำงานได้เร็ว เช่น เครื่องพิมพ์ทำงานได้ช้า เมื่อคอมพิวเตอร์ส่งตัวอักษรตัวแรกมาพิมพ์เครื่องพิมพ์รับตัวอักษรและกำลังจะพิมพ์ คอมพิวเตอร์ก็จะส่งตัวอักษรตัวที่ 2 ตัวที่ 3 ตามมา ทำให้การประมวลผลของอุปกรณ์เครื่องพิมพ์ทำงานไม่ทัน ซึ่งอาจทำให้ข้อมูลสูญหาย ดังนั้นเครื่องพิมพ์จึงส่งสัญญาณบอกคอมพิวเตอร์ได้ "อย่าเพิ่งส่งมาเพราะยังไม่พร้อมที่จะรับ" ลักษณะของการรับส่งข้อมูลอินพุทเอาท์พุทแบบมีตัวตรวจสอบสัญญาณดังรูปที่ 4.7 นั้นจะใช้ PA_0-PA_7 เป็นเอาท์พุท และ PB_0-PB_7 เป็นอินพุทโดยพอร์ท C เป็นตัวตรวจสอบสัญญาณดังแผนผังในรูปที่ 4.8



รูปที่ 4.8 วงจรการต่อ 8255 ในโหมด 1

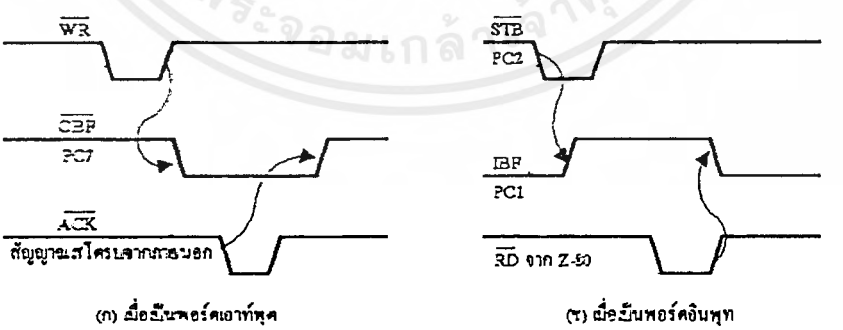
เมื่อโปรแกรม 8255 เป็นโหมด 1 แล้ว ตัว 8255 จะให้พอร์ท C เป็นสัญญาณควบคุม โดยแต่ละบิทของพอร์ท C เป็นไปตามที่กำหนดไว้ดังตารางที่ 4.3

ขา	กรณีอินพุท	กรณีเอาท์พุท
PC_0	$INTR_B$	$INTR_B$
PC_1	IBF_B	OBF_B
PC_2	STB_B	ACK_B
PC_3	$INTR_A$	$INTR_A$
PC_4	STB_A	I/O
PC_5	IBF_A	I/O
PC_6	I/O	ACK_A
PC_7	I/O	OBF_A

ตารางที่ 4.3 หน้าที่ของสัญญาณต่าง ๆ ของพอร์ท C ในการทำงานเป็นตัวตรวจสอบสัญญาณเมื่อ 8255 ทำงานในโหมด 1

โดยปกติ 8255 จะให้สัญญาณอินเตอร์รัพต์ไปบอกซีพียูด้วย สัญญาณอินเตอร์รัพต์ของ 8255 จะเกิดขึ้นที่ PC_3 และ PC_0 โดยที่เมื่อที่บัฟเฟอร์พร้อมแล้วและต้องการให้ซีพียูส่งอินพุทหรือเอาท์พุทมาที่บัฟเฟอร์ สัญญาณอินเตอร์รัพต์ก็จะเกิดขึ้น

โครงสร้างการตรวจสอบสัญญาณของ 8255 แสดงด้วยสัญญาณไฟฟ้าได้ดังรูปที่ 4.9



รูปที่ 4.9 แผนผังเวลาการรับและส่งข้อมูลโดยใช้ตัวตรวจสอบสัญญาณ

สังเกตว่า การทำงานของ 8255 จะเกี่ยวข้องกับสัญญาณ RD และ WR ซึ่งจะทำให้สัญญาณควบคุมเปลี่ยนแปลงไป การตรวจสอบสัญญาณซึ่งกันและกันนี้ เป็นวิธีการรับส่งที่มี

ประสิทธิภาพ เช่น ในกรณีอินพุท เมื่ออุปกรณ์ภายนอกต้องการส่งข้อมูลให้ซีพียู ก็จะส่งข้อมูลแบบขนานเข้ามาพร้อมทั้งสโตบ (STB) บอก 8255 ซึ่ง 8255 จะนำข้อมูลนั้นไปเก็บไว้ในรีจิสเตอร์ภายนอกก่อนแล้วส่งสัญญาณตอบบอกว่า "บัฟเฟอร์ยังเต็มอยู่ (IBF) อย่าเพิ่งส่งมาอีก" ครั้นเมื่อซีพียูอ่านข้อมูลจากรีจิสเตอร์ไปแล้วส่วนของสัญญาณบัฟเฟอร์อินพุท (IBF) ก็จะบอกว่า "ว่างแล้วส่งมาได้" อุปกรณ์ภายนอกก็จะส่งข้อมูลมาให้อีก

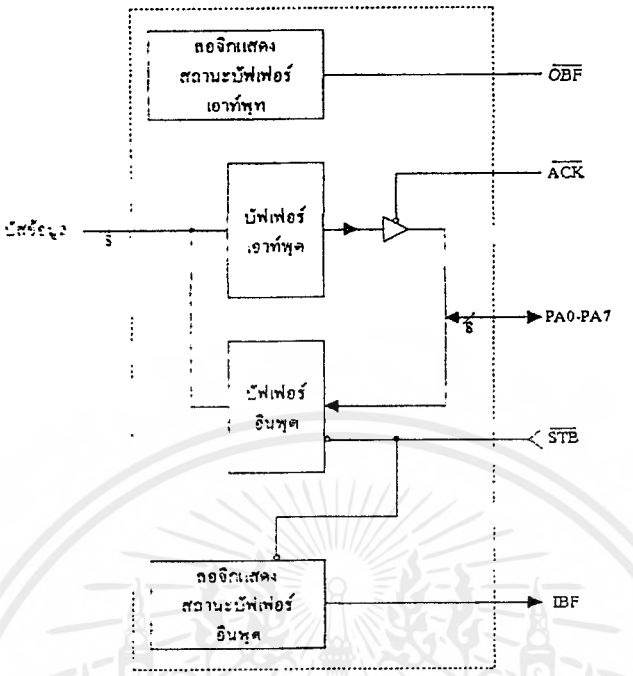
ทำนองเดียวกัน สำหรับพอร์ทเอาท์พุท เมื่อซีพียูส่งข้อมูลออกทางพอร์ทเอาท์พุทให้กับ 8255 ตัว 8255 ก็จะรับไว้ในรีจิสเตอร์ภายใน พร้อมทั้งส่งสัญญาณออกไปบอกอุปกรณ์ภายนอกว่า "เอาท์พุทบัฟเฟอร์ของฉันมีข้อมูลอยู่ (OBF) ให้มาอ่านไป" อุปกรณ์ภายนอกเมื่อทราบและพร้อมอ่านก็จะส่งสัญญาณตอบรับ (ACK) พร้อมกับอ่านข้อมูลไป โดยสัญญาณ (ACK) จะมีความหมายว่า "ฉันอ่านข้อมูลไปแล้ว" ด้าน 8255 ก็จะตอบกลับมาว่า "บัฟเฟอร์ฉันว่างแล้วให้อุปกรณ์ภายนอกรอก่อน จะมีข้อมูลใหม่ส่งมาให้อีก"

4.9 การทำงานของ 8255 ในโหมด 2

8255 ยังมีโหมดการทำงานอีกโหมดหนึ่งคือ โหมด 2 ซึ่งทำได้เฉพาะพอร์ท A ในโหมดนี้ 8255 จะใช้พอร์ท A ทำหน้าที่เป็นพอร์ทแบบ 2 ทิศทางคือ สามารถเป็นได้ทั้งพอร์ทอินพุทและพอร์ทเอาท์พุท โดยโครงสร้างของพอร์ท A ทั้งอินพุทเอาท์พุทมีตัวตรวจสอบสัญญาณทั้งคู่ ส่วนพอร์ท C จะทำหน้าที่เป็นสัญญาณตรวจสอบ โดยมีสัญญาณแต่ละขาดังตารางที่ 4.4

พอร์ท C	ความหมาย
PC_0	I/O
PC_1	I/O
PC_2	I/O
PC_3	$INTR_A$
PC_4	STB_A
PC_5	IBF_A
PC_6	ACK_A
PC_7	OBF_A

ตารางที่ 4.4 หน้าที่ของพอร์ท C ในโหมดที่ 2



รูปที่ 4.10 โครงสร้างของพอร์ท C ที่ทำงานแบบพอร์ท 2 ทิศทาง

สังเกตว่าเมื่อโปรแกรมพอร์ท A เป็นโหมด 2 แล้ว พอร์ท B จะต้องโปรแกรมเป็นโหมด 0 หรือ โหมด 1 ก็ได้ ซึ่งก็ทำงานแบบแยกอิสระอีก ในการใช้งานแบบ 2 ทิศทางนี้ใช้ได้กับงานบางประเภท เช่น ใช้ในการรับส่งข้อมูลของพอร์ทมาตราฐานบางประเภท เช่น IEEE 488 หรือ ใช้เชื่อมโยงระหว่างคอมพิวเตอร์กับคอมพิวเตอร์ในการรับส่งข้อมูลสลับกัน ไปและกลับ

บทที่ 5
แนะนำโปรแกรมเดลไฟ

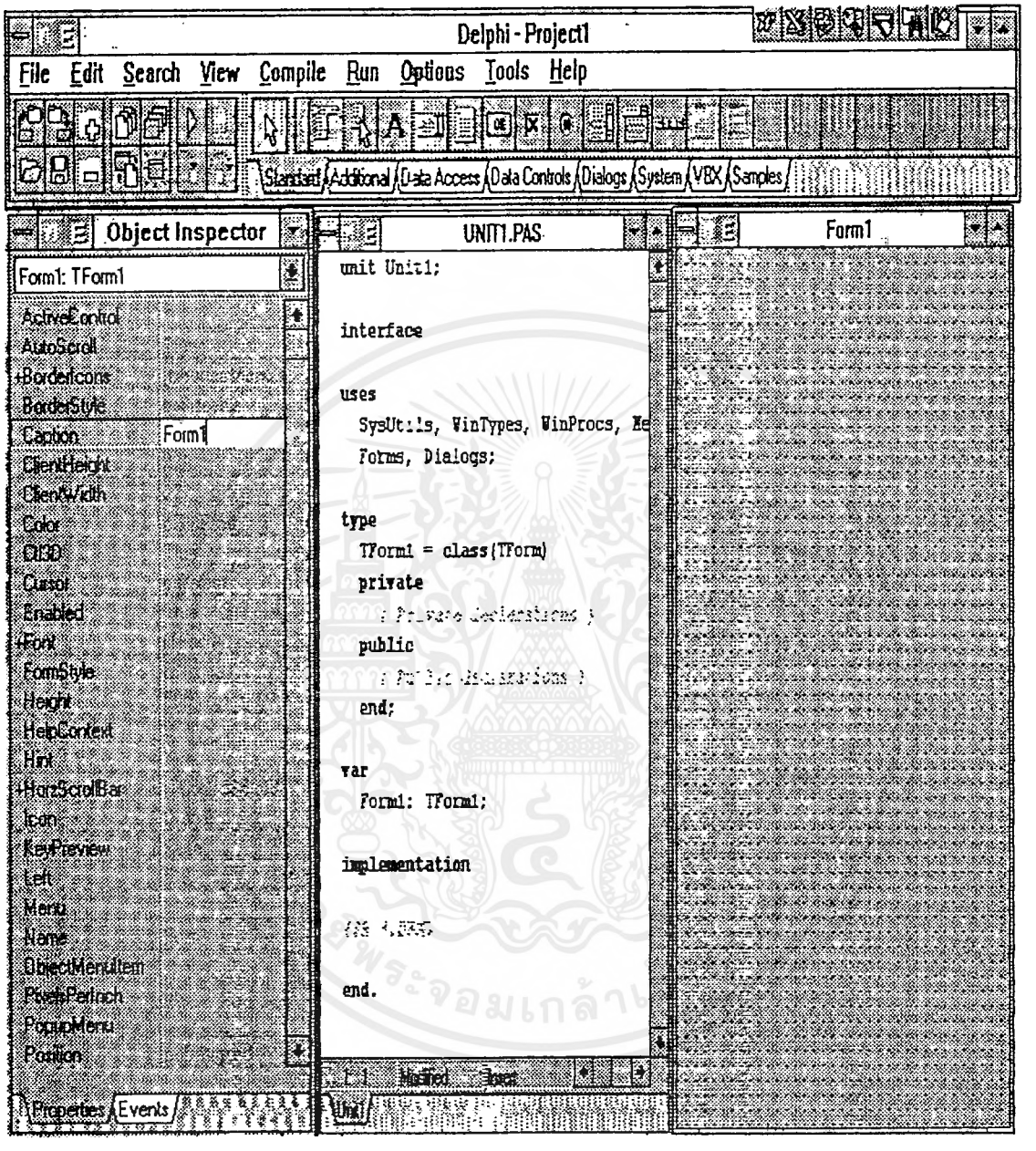
โปรแกรมเดลไฟคือ Borland Delphi for Windows เป็นได้ทั้งคอมพายเลอร์ภาษาปาสคาลเอดิเตอร์ และ จะประกอบด้วยยูทิตี้ต่างๆเพื่อเป็นการโปรแกรมกับวินโดวส์ด้วยวิชวลคือการกำหนดคอมโปเนนต์แล้วเสริมด้วยโปรแกรมปาสคาล ในความต้องการของฮาร์ดแวร์ในโปรแกรมเดลไฟนั้นมีดังนี้

- ซีพียู ควรเป็น 486 DX -66 ขึ้นไป
- หน่วยความจำหลัก ควรเป็น 8 เมกะไบต์ขึ้นไป
- เนื้อที่ว่างในฮาร์ดดิสก์ อย่างน้อย 80 เมกะไบต์ขึ้นไป

หากใช้ ซีพียู ที่มีความเร็วต่ำกว่า 486 DX -66 เมื่อดำเนินการใดแล้วจะต้องคอย เพราะไม่ได้แสดงเคอร์เซอร์เป็นรูปนาฬิกาทรายไว้

เมื่อเปิดใช้เดลไฟจะปรากฏภาพบนจอซึ่งจะประกอบด้วยหน้า 4 บานคือ

1. หน้าต่างหลักอยู่บนสุดจะประกอบด้วย
   - ไตเติ้ลบาร์และเมนูบาร์
   - สปีดบาร์ (Speed bar) ได้เมนูบาร์ตั้งแต่ File ถึง View คือสปีดบาร์ มีปุ่มต่างๆแทนรายการในเมนู
   - กล่องอุปกรณ์ (Component palette) อยู่ใต้เมนูบาร์ทางด้านขวาตั้งแต่รายการ Compile เป็นต้นไปคือกล่องอุปกรณ์ แบ่งออกเป็นหลายชั้นหรือหลายหน้าตามที่ได้ทำที่ชั้นหน้าไว้เป็น Standard ถึง Sample ซึ่งเมื่อเลือกที่ชั้นหน้าใดจะแสดงอุปกรณ์คือคอมโปเนนต์ของหน้านั้นให้เลือกต่อ
2. หน้าต่าง Object Inspector อยู่ทางด้านซ้าย แบ่งออกเป็น 2 หน้าคือ Properties เพื่อให้ค่าพรอพเพอร์ตี้ และ Events เพื่อเลือกเหตุในการป้อนโปรแกรม
3. แบบฟอร์มหรือฟอร์ม คือหน้าต่าง Form1 ในรูปที่ 2 เพื่อการกำหนดคอมโปเนนต์
4. หน้าต่างเอดิเตอร์ คือหน้าต่าง Unit1.PAS เพื่อการป้อนโปรแกรม



รูปที่ 5.1 หน้าต่างของโปรแกรม Delphi

การโปรแกรมจะแบ่งได้ 2 ขั้นคอน คือ 1.การเตรียมโปรแกรม และ2.การรันทดสอบ

1.การเตรียมโปรแกรมจะแบ่งออกเป็น 3 ขั้นตอนคือ

1.1 กำหนดคอมโปเนนต์

- โดยเลือกแล้วกำหนดตำแหน่งและขนาดในฟอร์ม
- จะได้เป็นออปเจ็กต์ มีชื่อตามชื่อคอมโปเนนต์ต่อท้ายด้วยเลข

1.2 ให้ค่าพรอพเพอร์ตี้ส์ในหน้า Properties โดยคลิ๊กเม้าส์ ณ หัวข้อ คือค่าพรอพเพอร์ตี้ส์ที่ต้องการแล้วให้ค่า

1.3 ป้อนโปรแกรม โดยดับเบิ้ลคลิ๊กเมาส์ ณ อีเว้นต์ที่ต้องการในหน้า Events จะแสดงโครงโปรแกรมให้ป้อนโปรแกรม

2. การรันทดสอบ สามารถที่จะรันโปรแกรมเพื่อการทดสอบ หรืออาจถือว่าเป็นการทดลองรันได้ทุกขณะที่ต้องการ โดยการเลือกรายการ Run/Run หรือเลือกปุ่มรันที่สปีดบาร์

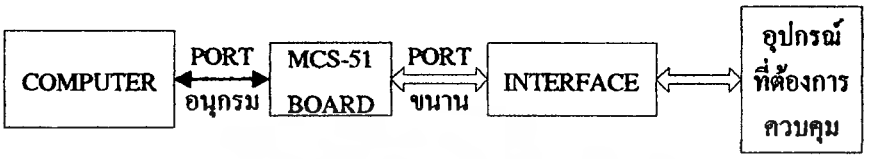
ที่สปีดบาร์หรือที่กล่องอุปกรณ์ เมื่อเลื่อนเคอร์เซอร์ของเมาท์ไปทาบ จะแสดงชื่อของสิ่งนั้นๆ ยุติการรันได้โดยปิดหน้าต่างตามปกติซึ่งเมื่อจะเตรียมโปรแกรมต่อให้ยุติการรัน

โปรแกรมเคลไฟยังมีเนื้อหาอีกมากมายที่เกี่ยวข้องกับการใช้โปรแกรม แต่ในบทนี้ผู้จัดทำต้องการเพียงจะบอกว่าโปรแกรมเคลไฟคืออะไร หน้าตาเป็นอย่างไร ส่วนเรื่องของการเขียนโปรแกรมนั้น คงจะต้องศึกษาค้นคว้า หาตำราอ่านเพิ่มเติมเอาเอง

บทที่ 6

การทำงานของในส่วนต่างๆของระบบ

6.1 การทำงานของระบบ



รูป 6.1 การทำงานของระบบ

ในการควบคุมอุปกรณ์ Hard ware เมื่อเรา Click mouse บนหน้าจอคอมพิวเตอร์ คอมพิวเตอร์จะส่งข้อมูลที่ เป็นระหัสที่ตั้งไว้ออกทางพอร์ทอนุกรม ไปยังคอนโทรลเลอร์ MCS-51 เพื่อให้คอนโทรลเลอร์ MCS-51 ทำการตรวจสอบระหัสที่ตั้งไว้บน MCS-51 ว่าตรงกับรหัสของตัวเองหรือไม่ ถ้าตรงก็ให้ไปทำงานตามคำสั่งที่ตั้งไว้ เช่น ไปเพิ่มเสียง BASS ถ้ารหัสไม่ตรงกับรหัสที่ตั้งไว้ก็ให้กลับไปเก็บค่าจาก Display ใหม่

ในการอ่านค่าจาก Display ไปยังคอมพิวเตอร์ ตัว MSC-51 จะทำการเก็บค่าจากพอร์ทของ 8255 ไว้ที่ memory เสมอ (เมื่อไม่มีการส่งค่าจากคอมพิวเตอร์ไปควบคุม Hard ware) เมื่อคอมพิวเตอร์ต้องการอ่านค่ากลับมายังตัวมันจะต้องส่งระหัสในหารรับค่ามายัง MCS-51 เพื่อบอกให้ส่งค่าที่เตรียมไว้ใน memory ออกมาทางสายอนุกรมไปยังคอมพิวเตอร์ จากนั้นโปรแกรมบนคอมพิวเตอร์จะตรวจสอบรหัสและข้อมูลที่ส่งมาว่าเป็นรหัสและข้อมูลของอะไรและไปทำงานแสดงผลที่ต้องการได้ การทำงานของระบบจะเป็นเช่นนี้ไปเรื่อยๆจนกว่าจะหยุดใช้โปรแกรมควบคุมการทำงานบนคอมพิวเตอร์

เมื่อไม่มีการใช้คอมพิวเตอร์ควบคุมการทำงาน (มีแต่ Hard ware) การใช้จะเหมือนกับอุปกรณ์เครื่องเสียงธรรมดา

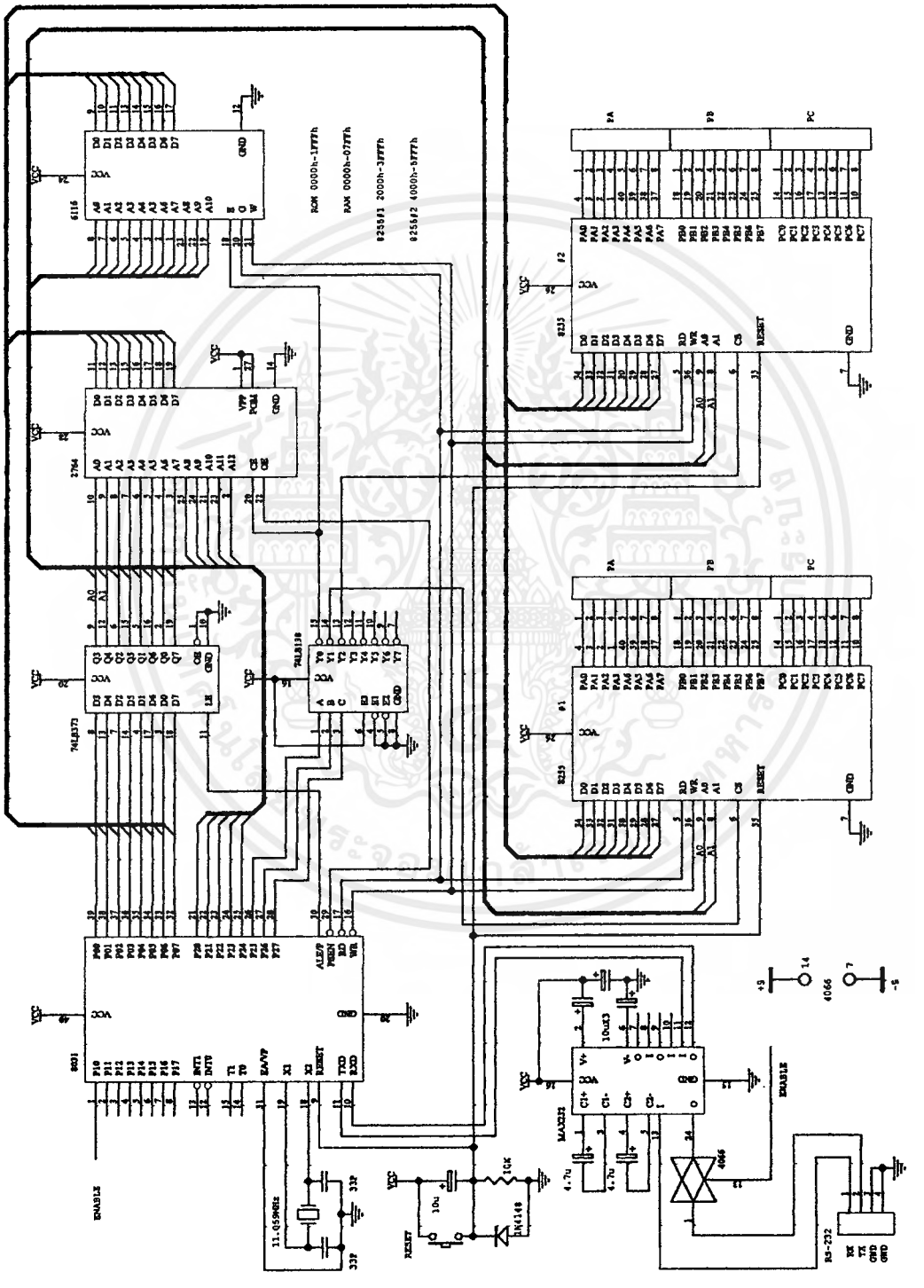
6.2 การทำงานของ HARD WARE

6.2.1 การทำงานของคอนโทรลเลอร์

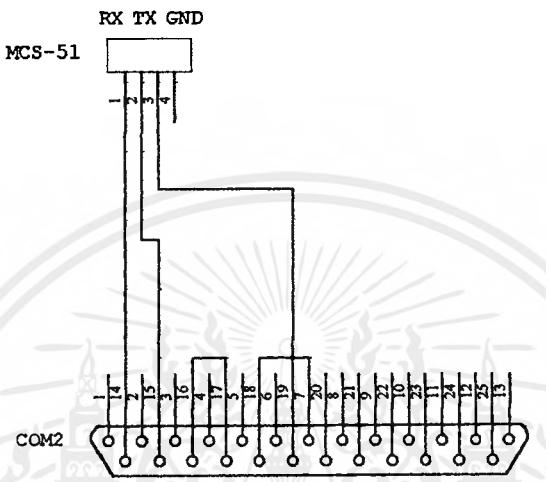
ตัวการใหญ่ที่ทำให้เกิด คอนโทรลเลอร์ คือ ตัวไมโครคอนโทรลเลอร์เอง ในที่นี้จะใช้ไอซีตระกูล MCS-51 โดยใช้เบอร์ 8031 เป็นไมโครคอนโทรลเลอร์ แบบที่ต้องการหน่วยความจำโปรแกรมภายนอกจากรอม(ROM) ซึ่งจะมีอุปกรณ์ต่อช่วยภายนอก ที่จะสามารถเรียกโปรแกรมภายนอกมา วิเคราะห์ทำงานตามความต้องการของผู้โปรแกรม โดยอุปกรณ์ภายนอก ที่ใช้ต่อร่วมต้องใช้ตัว แลตช์ แอดเดรส 8 บิท (bit) คือ 74LS373 ตัวแปลงรหัสตามแอสเดรส คือ 74LS138 และตัวเก็บข้อมูลโปรแกรมถาวร คือตัว EPROM 2764 ขนาดความจุ 8 กิโลไบท์ และมีตัวเก็บข้อมูลชั่วคราว คือ RAM 6116 ขนาด 2 กิโลไบท์ ซึ่งถ้าไม่จำเป็นก็ไม่ต้องใช้ก็ได้ ส่วนการติดต่อกับอุปกรณ์อื่นๆ นั้น จะมีไอซี 8255 เป็นตัวช่วยขยายการติดต่อ ใน 8255 1 ตัวจะสามารถ ขยายได้ 3 พอร์ตถ้าต้องการใส่เพิ่มมากขึ้นก็สามารถต่อเพิ่มได้ โดยใช้ 74LS138 เป็นตัวกำหนดช่วงการทำงาน

ในการที่ ไมโครคอนโทรลเลอร์จะติดต่อกับคอมพิวเตอร์ได้ ต้องใช้อุปกรณ์ช่วยคือ IC MAX-232 จะใช้เป็นบัฟเฟอร์รับส่งข้อมูลผ่านจากคอมพิวเตอร์สู่ MCS-51 และจาก MCS-51 สู่คอมพิวเตอร์ เมื่อมีการติดต่อระหว่างคอมพิวเตอร์กับคอมพิวเตอร์ ก็จะไม่เกิดปัญหาอะไร แต่ถ้า เป็นการติดต่อระหว่าง MCS-51 หลายๆ ตัว กับ คอมพิวเตอร์ตัวเดียว ในเวลาเดียวกัน ขาส่งจาก MAX-232 จะเกิดปัญหา จะต้องมีวงจรตัดต่อการทำงานให้เข้าจังหวะ ซึ่งกันและกัน ของ MCS-51 ทั้งหลาย ถ้าไม่ทำเช่นนี้ข้อมูลจะผิดพลาดได้

เริ่มจากการจ่ายไฟเข้าเครื่อง MCS-51 วงจรรีเซตอัตโนมัติจะทำงาน โดยเคลียค่าต่างๆ เพื่อเริ่มต้น วงจรจะทำงานเมื่อวงจรหยุดการรีเซต MCS-51 ก็เริ่มทำงาน MCS-51 จะทำการเรียกข้อมูลโปรแกรมภายนอกจาก EPROM ตั้งแต่แอดเดรส 0000H มาทำการวิเคราะห์ และทำงานตามโปรแกรมที่มีอยู่ใน EPROM ต่อไป วงจรคอนโทรเลลอร์แสดงดังรูปที่ 6.2



รูปที่ 6.2 คอนโทรลเลอร์บอร์ด



รูปที่ 6.3 แสดงตำแหน่งการเชื่อมต่อขาจาก RS-232 ผ่านพอร์ตคอม 2

ขา 2 ส่งข้อมูลจากคอมพิวเตอร์

ขา 3 รับข้อมูลเข้ามาให้คอมพิวเตอร์

ขา 4 Request To Send สายสำหรับตรวจสอบแฮนด์เช็ค

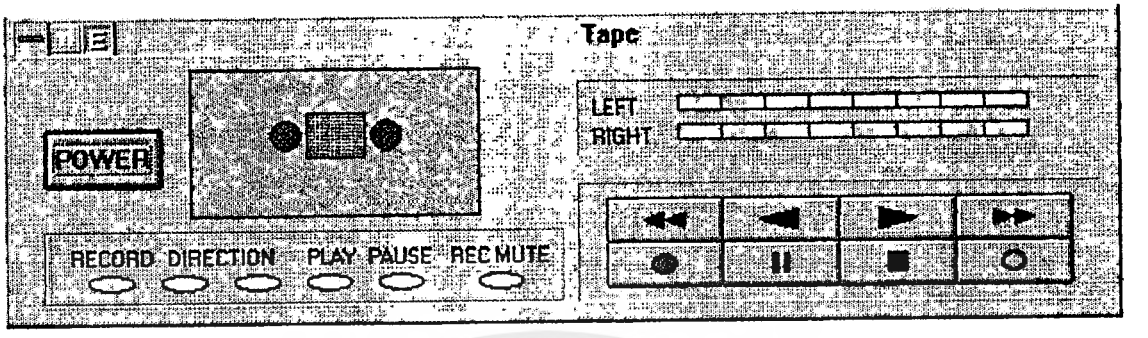
ขา 5 Clear To Send สายสัญญาณสำหรับการทำแฮนด์เช็ค

ขา 6 Data Set Ready สายสัญญาณสำหรับการทำแฮนด์เช็ค

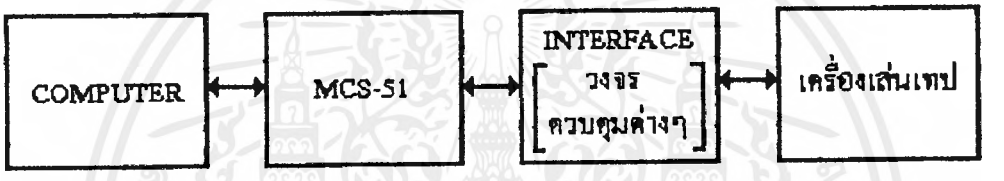
ขา 7 ระดับอ้างอิงของสัญญาณข้อมูล

ขา 20 Data Terminal Ready สายสัญญาณสำหรับการทำแฮนด์เช็ค

6.2.2 การทำงานของ TAPE



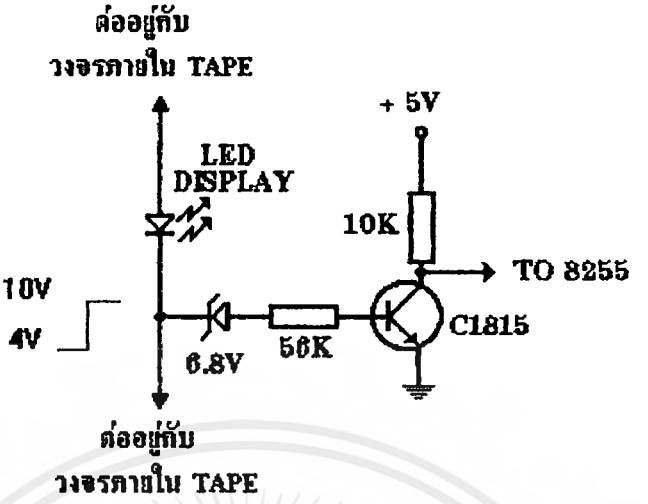
รูปที่ 6.4 แสดงหน้าปัทม์เครื่องเล่นเทปบนจอคอมพิวเตอร์



รูปที่ 6.5 แสดงบล็อคไดอะแกรมการควบคุมเทป

- การทำงานของภาคแสดงสภาวะการทำงานของเครื่องเล่นเทป

ขณะที่เครื่องเล่นเทปทำงานจะมีไฟบอกสภาวะการทำงานในหน้าที่ต่างๆ คือ PLAY, PAUSE, REC, DIRECTION OF PLAY (reverse & forward) , REC MUTE เนื่องจากเครื่องเล่นเทปที่นำมาใช้ร่วมกับโครงงานนี้มีการแสดงสภาวะการทำงานในหน้าที่ต่างๆด้วยแอลอีดี (LED) โดยขาบวก (anode) ของแอลอีดี (LED) ภาคแสดงผลนี้จะต่อร่วมกันหมด (COMMON ANODE) และลบ (cathode) จะต่อไปยังตำแหน่งต่าง ๆ กันดังรูปที่ 6.6 ส่วนวงจรที่นำมาต่อร่วมเพื่อส่งสัญญาณไปแสดงยังคอมพิวเตอร์ ผ่านไอซี 8255 ซึ่งเป็น อินพุท/เอาท์พุท พอร์ท โดยนำทรานซิสเตอร์ชนิด พีเอ็นพี ต่อขาเบสร่วมกับขาลบของแอลอีดี (LED) ผ่านตัวต้านทาน (R1-R6) และซีเนอร์ไดโอดขนาด 6.8 โวลท์ และ ขาคอลเล็คเตอร์ของทรานซิสเตอร์ต่อกับตัวต้านทาน (R1-R6) ค่า 10 กิโลโอห์ม แล้วต่อไปยังแหล่งจ่าย (Vcc 5V) ขาบวกของแอลอีดี (LED) แสดงผลถูกต่อไปยังแหล่งจ่ายด้วยเช่นกัน ขาคอลเลคเตอร์ของทรานซิสเตอร์ซึ่งเป็นเอาท์พุทต่อไปยังพอร์ทของไอซี 8255

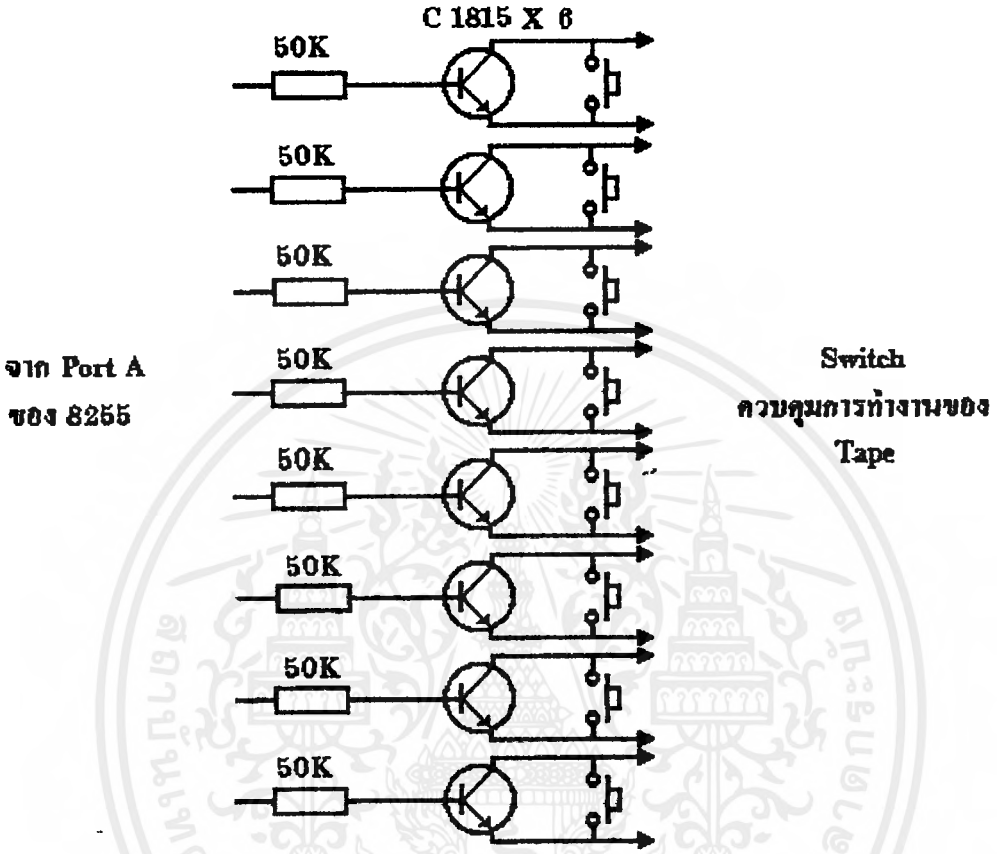


รูปที่ 6.6 วงจรแสดงสภาวะการทำงานของเครื่องเล่นเทป

พิจารณาที่ทรานซิสเตอร์ตัวที่ 1 (TR 1) ในขณะที่แอลอีดี (LED) ไม่ติดจะปรากฏแรงดันที่ขาแคโถดของแอลอีดี ค่า 9.18 โวลท์ จึงมีกระแสเบสของทรานซิสเตอร์ไหล เนื่องจากแรงดันที่ขาแคโถดของแอลอีดี มีค่ามากกว่าแรงดันซีเนอร์ ของซีเนอร์ไดโอด ซึ่งมีค่า 6.8 โวลท์อยู่ประมาณ 2.38 โวลท์ ทำให้ทรานซิสเตอร์นำกระแส (SATURATION) ขณะนี้มีแรงดันที่ขาคอลเลคเตอร์ (Vc,Vce) มีค่าเท่ากับ 0.1 โวลท์ หรือสถานะโลว์ (LOGIC 0) ไปยังพอร์ทของไอซี 8255 และเมื่อแอลอีดี ติดจะมีแรงดันค่า 4.38 โวลท์ ที่ขาแคโถดของแอลอีดี ซึ่งมีค่าน้อยกว่าแรงดันซีเนอร์ของซีเนอร์ไดโอดจึงไม่มีกระแสเบสไหลทำให้ทรานซิสเตอร์ไม่นำกระแส (CUT OFF) จึงมีแรงดัน 5 โวลท์ ที่ขาคอลเลคเตอร์ (Vc,Vce) หรือสถานะไฮ (LOGIC 1) ส่งไปยังพอร์ทของไอซี 8255

ดังนั้นในขณะที่เครื่องเล่นเทปกำลังทำงานในสถานะใดแอลอีดี ของการบอกสถานะการทำงานนั้น ๆ จะติดและจะส่งสัญญาณลอจิก "1" ไปยังพอร์ทของไอซี 8255 เพื่อส่งต่อไปยังคอมพิวเตอร์เพื่อแสดงการทำงานของเครื่องเล่นเทปที่คอมพิวเตอร์ และส่งสัญญาณลอจิก "0" เมื่อไม่ได้ทำงานในสถานนั้น ๆ พอร์ทของ 8255 ที่ใช้ในการบอกสภาวะการทำงานของเครื่องเล่นเทป ได้แก่ พอร์ท PB1-PB6 สำหรับ PB1 บอกสภาวะของ RECORD และ PB2 ถึง PB6 สำหรับ PLAY REVERSE , PLAY FORWARD , PLAY , PAUSE และ REC MUTE ตามลำดับ

-วงจรควบคุมการทำงานของเครื่องเล่นเทป



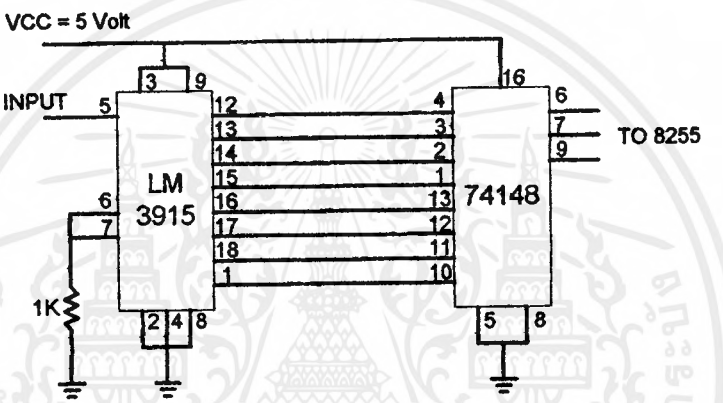
รูปที่ 6.7 วงจรควบคุมการทำงานของเครื่องเล่นเทป

สำหรับการสั่งงานให้เครื่องเล่นเทปทำงานในหน้าที่ต่างๆ คือ PLAY, REVERSE, FOWARD, PAUSE ,DIRECTION OFF PLAY (reverse & forward) , REC MUTE , STOP, RECORD ถูกสั่งงานโดยผ่านสวิทช์ชนิดกดติดปล่อยดับ เมื่อสั่งงานจากตัวเครื่อง และสั่งงานจากคอมพิวเตอร์โดยเราทำการสั่งงานผ่านเอาท์พุทพอร์ทของ 8255 จากรูปที่ 6.7 เป็นวงจรที่ใช้ในการสั่งงานจากคอมพิวเตอร์เพื่อให้เครื่องเล่นเทปทำงานในหน้าที่ต่าง ๆ โดยมีตัวต้านทานต่ออนุกรมอยู่ระหว่างพอร์ทของ IC 8255 และขาเบสของทรานซิสเตอร์ ขาคอลเลคเตอร์และขาอีมิตเตอร์ของทรานซิสเตอร์ถูกต่อคร่อมกับสวิทช์ชนิดกดติดปล่อยดับของเครื่องเล่นเทป เมื่อต้องการให้เครื่องเล่นเทปทำงานในหน้าที่ใด ๆ ก็จะให้พอร์ทที่ต่อกับปุ่มที่ทำหน้าที่นั้น ๆ เป็นสภาวะไฮ (LOGIC 1) จะทำให้ทรานซิสเตอร์อิ่มตัว (Saturate) จึงเกิดการนำกระแส ทำให้เทปทำงานในหน้าที่นั้น ๆ

สัญญาณที่ส่งไปควบคุมจะเป็นลอจิก "1" เมื่อต้องการให้ทำงานในหน้าที่ที่ต้องการและเป็นลอจิก "0" ในสภาวะปกติและสัญญาณลอจิก "1" จะส่งเป็นช่วงสั้นๆช่วงเดียว (Monostatable)

ตำแหน่งพอร์ทเอาท์พุท ที่จะสั่งงานในหน้าที่ต่างๆสำหรับ 8255 คือ PA0-PA7 โดย PA0 สำหรับ REC MUTE และ PA1 ถึง PA7 สำหรับ PAUSE , FORWORD , PLAY FORWORD , PLAY REVERSE , REVERSE , RECORD และ STOP ตามลำดับ

-การทำงานวงจรระดับสัญญาณเสียงของภาคเครื่องเล่นเทป

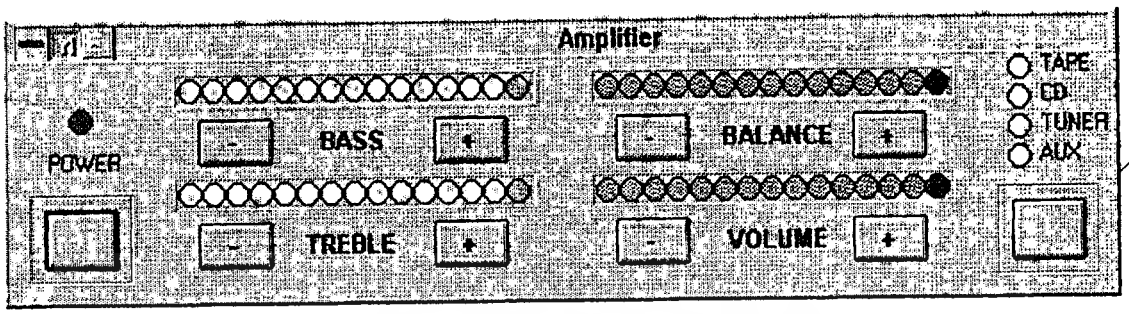


รูปที่ 6.8 วงจรแสดงการทำงานระดับสัญญาณของเครื่องเล่นเทป

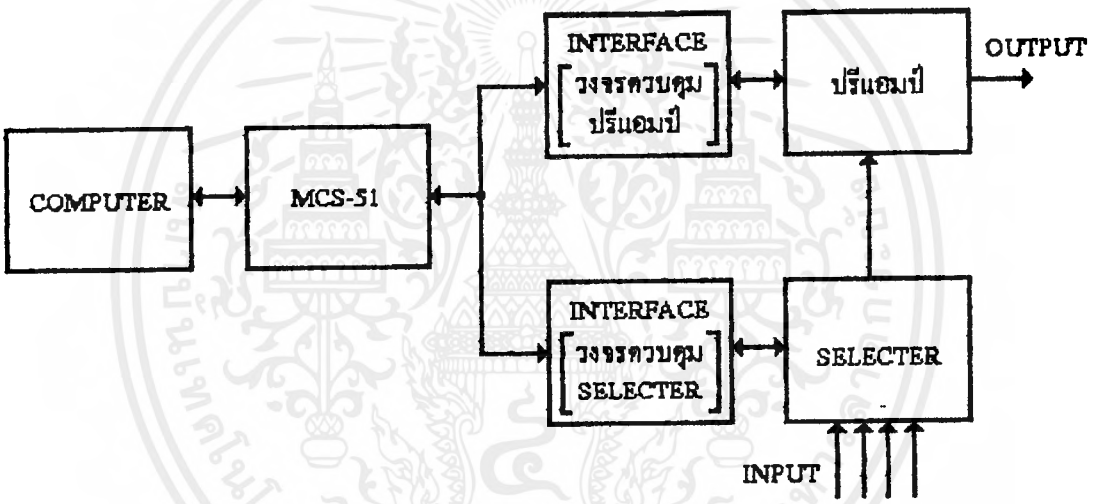
การส่งการแสดงระดับสัญญาณเสียงของเครื่องเล่นเทป ทำโดยใช้ไอซี LM3915N ซึ่งเป็นไอซีสำเร็จรูปสามารถขับแอลอีดี (LED) 10 ดวงให้สว่างเป็นแถบหรือทีละดวงได้ตามขนาดแรงดันอินพุท (input) ภายในของวงจรจะมีการต่อตัวต้านทานเพื่อแบ่งแรงดันเป็นช่วง ๆ ไว้แล้ว

การส่งสัญญาณไปยังพอร์ท 8255 จะส่งไปโดยการเข้ารหัสเป็นฐาน 2 โดยผ่านไอซี 74148 ซึ่งเป็น 8 to 3 line encoder เข้าไปยังตำแหน่งพอร์ท PC0-2 สำหรับแสดงลำดับสัญญาณข้างซ้าย และพอร์ท PC4-6 สำหรับแสดงลำดับสัญญาณข้างขวา

6.2.3 การทำงานของวงจรที่เกี่ยวกับ PRE - TONE AMPLIFIER

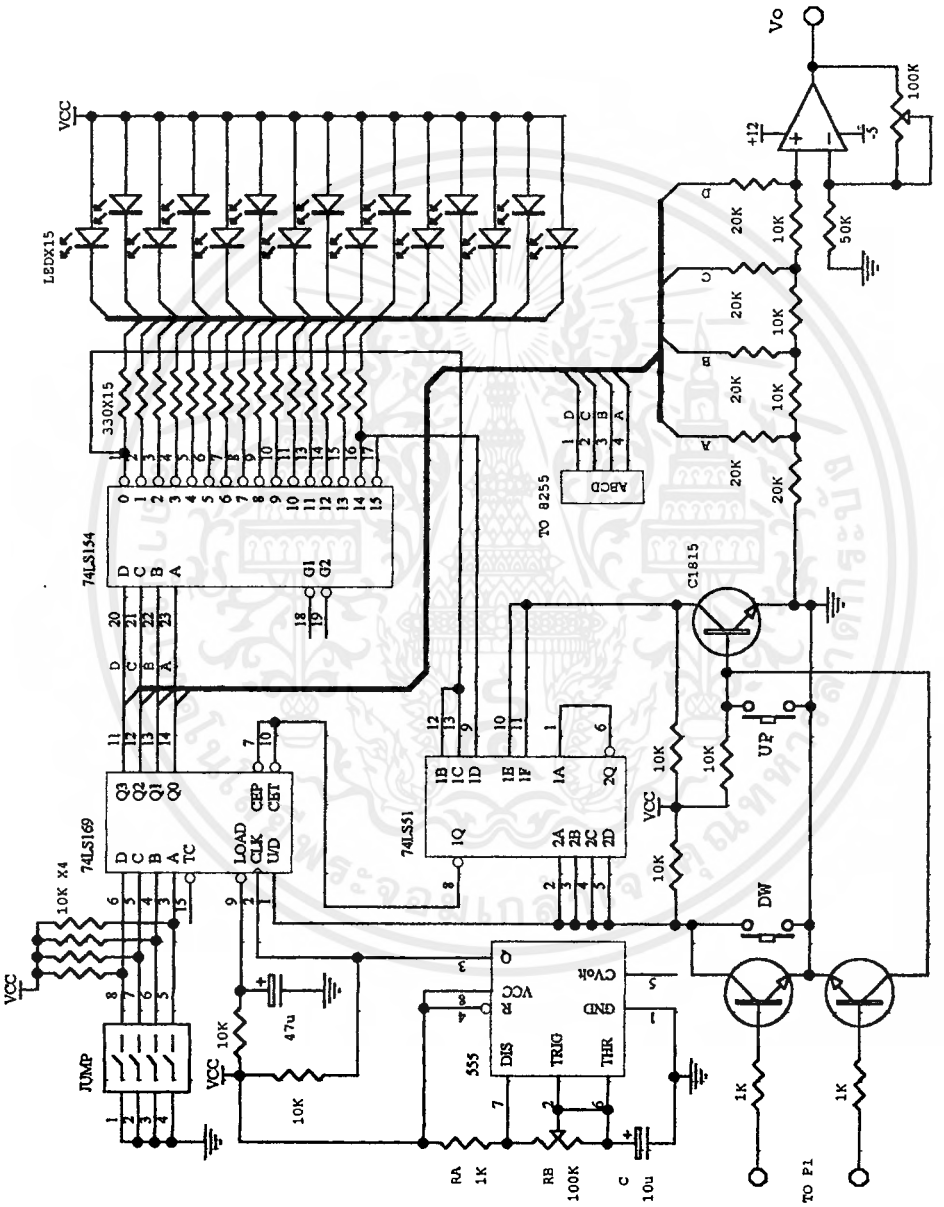


รูปที่ 6.9 แสดงหน้าปัมท์ของปรีแอมป์บนจอคอมพิวเตอร์

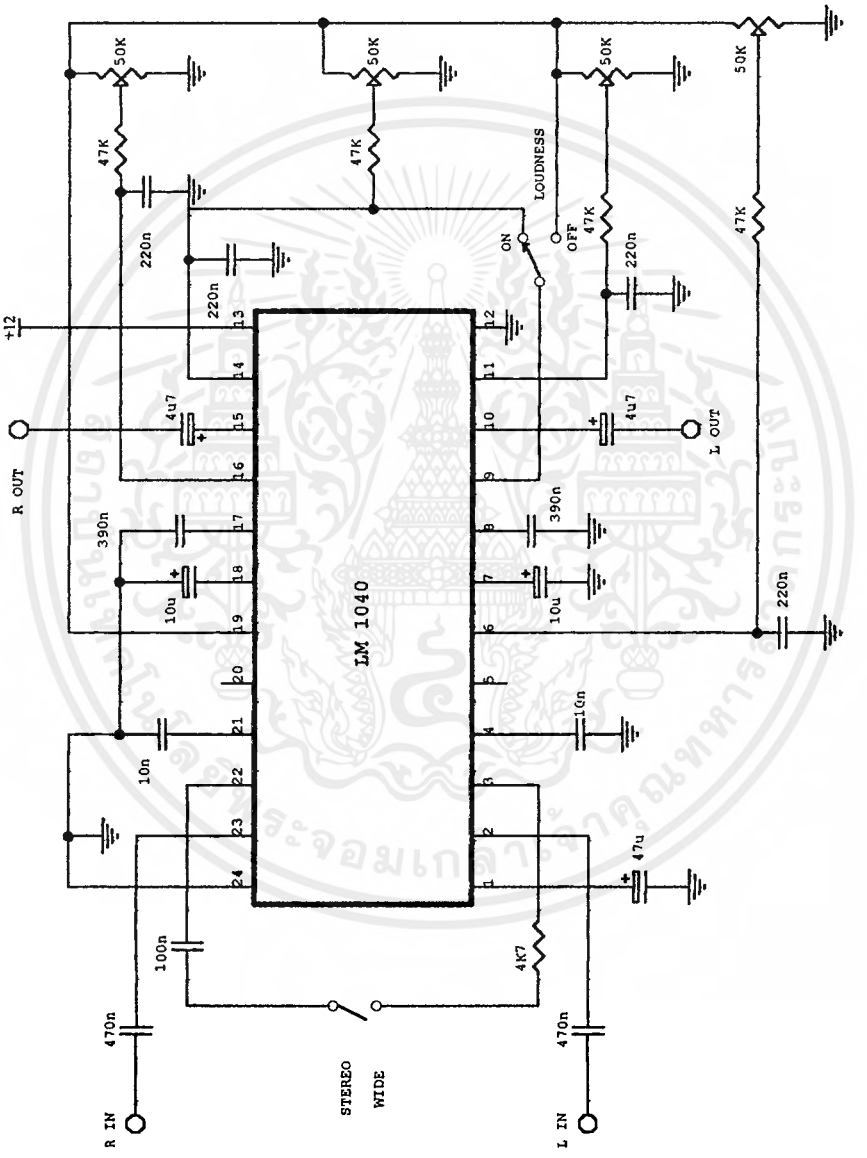


รูปที่ 6.10 แสดงบล็อคไดอะแกรมการควบคุมปรีแอมป์

เนื่องจากวงจรของปรีแอมป์ (PRE-AMPLIFIER) และโทนคอนโทรล (TONECONTROL) เป็นแบบดีซีคอนโทรล (DC CONTROL) คือใช้ระดับแรงดันตรงควบคุมการทำงานต่าง ๆ คือ ควบคุมระดับความแรงของสัญญาณ(VOLUME), ระดับการเพิ่ม, ลด (BOOT , CUT) ของความถี่สูง(TREBLE) และความถี่ต่ำ (BASS) , อัตราส่วนของขนาดของสัญญาณซ้ายและสัญญาณขวา (BALANCE) ในวงจรเดิมใช้ตัวต้านทานปรับค่าได้เพื่อแบ่งแรงดัน (DIVIDER) จากแรงดันอ้างอิงคือ 5 โวลท์ เพื่อให้ค่าแรงดันเปลี่ยนแปลงตั้งแต่ 0 โวท์ถึง 5 โวลท์ ที่โวลุ่ม (VOLUME) หากแรงดันที่ขาควบคุมมีค่า 0 โวลท์ จะมีขนาดสัญญาณเสียงที่มีขนาดเล็กที่สุด และจะมีขนาดของสัญญาณใหญ่ที่สุดเมื่อขาควบคุมมีแรงดัน 5 โวลท์ ที่ เสียงทุ้ม (BASS) เสียงแหลม (TREBLE) และบัลลานซ์ (BALANCE) ซึ่งขณะปกติที่ยังไม่มีการปรับแต่งจะมีระดับอยู่ตรงกึ่ง



รูปที่ 6.11 วงจรการควบคุม PRE-TONE AMPIFIER



รูปที่ 6.12 การควบคุมโดยใช้แรงดันไฟตรง

กลาง ก็คือมีระดับแรงดันประมาณ 2.5 โวลท์ เมื่อต้องการปรับแต่งก็จะปรับเปลี่ยนระดับแรงดันให้มากขึ้นหรือน้อยลงนั่นเอง

วงจรการทำงานจะเริ่มจากไอซี 74LS169 เป็นไอซีนับขึ้นนับลงฐานสอง 4 ตำแหน่ง ในตัวสามารถโหลดอินพุทควบคุมสถานะเอาท์พุทได้ โดยขาโหลดจะแอคตีฟที่ลอจิก "0" ที่ขาโหลดเราต่อวงจรรีเทอัตโนมัติเมื่อเปิดเครื่อง ขาโหลดจะเป็นลอจิก "0" ชั่วขณะ แล้วกลับเป็น "1" ในช่วงที่เป็นลอจิก "0" ไอซี จะโหลดอินพุทที่เราตั้งไว้ออกมาที่เอาท์พุท โดยเอาท์พุทที่ได้นำไปใช้ในการดีโค๊ดเดอร์แปลงเป็นอนาล็อก นำไปให้ ไอซี เบอร์ 8255 เอาท์พุทที่นำไปดีโค๊ดเดอร์ส่งไปให้ ไอซี เบอร์ 74LS154 ดีโค๊ดเดอร์ เข้า 4 ออก 16 ใช้งาน Y1-Y14 ซึ่งเป็นเอาท์พุทนำไปขับ LED แสดงผลหน้าปัทม์ AMP ในการนำไปทำอนาล็อกเพื่อจะนำอนาล็อกไปควบคุมฟังก์ชั่นของ AMP โดยใช้หลักการของวงจร R-2R Ladder ได้แรงดันเอาท์พุทเป็นสัญญาณอนาล็อก แต่เนื่องจากแรงดันเอาท์พุทที่ออกค่านี้ไม่เป็น 0 ถึง 5 โวลท์ จริง จึงนำออปแอมป์มาต่อร่วม โดยจัดวงจรแบบ Non-Inverting Amplifier และปรับแต่งแต่ละวงจรด้วยตัวต้านทานปรับค่าเพื่อให้ได้แรงดันเป็น 0 ถึง 5 โวลท์จริง

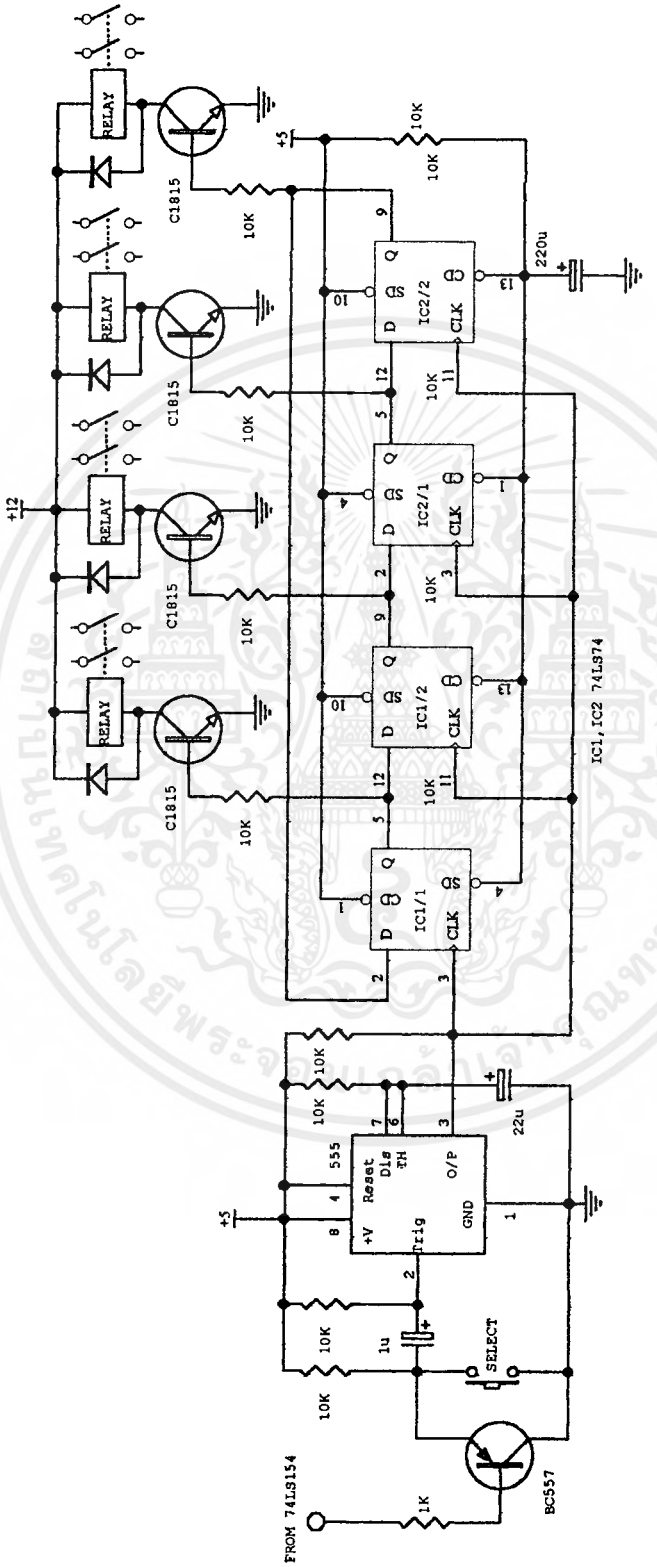
จากหลักการของวงจร R-2R LADDER ที่ใช้จำนวนบิท 4 บิท จะสามารถแบ่งระดับแรงดันเป็น 16 ระดับ แต่เนื่องจากใน BASE , THEBLE , BALANCE เราต้องการให้มีจุดกึ่งกลาง จึงใช้เพียง 15 ระดับ คือจุดกลางอยู่ที่ระดับที่ 7 (0111) สมารถเพิ่มได้ 7 ระดับ (ถึง 1110) และลดลงได้ 7 ระดับเช่นเดียวกัน (ถึง 0000) ในแต่ละระดับที่เพิ่มหรือลดขนาดของระดับแรงดันตรงที่เปลี่ยนแปลงเท่ากับ 0.33 โวลท์ ที่โวลุ่ม (VOLUME) จะตั้งค่าเริ่มต้นในตอนเปิดเครื่องไว้ที่ 0000 (ขาเซตตั้งที่ 0000) คือตั้งไว้ที่ระดับแรงดัน 0 โวลท์ หรือระดับขนาดสัญญาณที่เล็กสุด และ เบส , เสียงแหลม , บัลลานซ์ ตั้งค่าเริ่มต้นที่ 0111 (ขาเซตตั้งค่าที่ 0111) เพื่อตั้งระดับแรงดันเริ่มต้นที่ครึ่งหนึ่งของแรงดันอ้างอิง

ไอซีเบอร์ 555 เป็นไอซีไทม์เมอร์ ต่อเป็นวงจรอะสเตเบิ้ล มัลติเวเบเตอร์ สามารถปรับความถี่ได้ โดยความถี่ f = 1.44/(Ra+2Rb)C มีหน่วยเป็นเฮิร์ท เมื่อ Rb เปลี่ยนแปลง f ก็จะเปลี่ยนแปลง นำสัญญาณเอาท์พุทขา 3 ไปเป็นสัญญาณกระตุ้นให้ ไอซี เบอร์ 74LS169 โดยต่อตรง ๆ แต่ไอซีไม่สามารถจะนับขึ้นนับลงได้ เพราะ CEP,CET เป็นลอจิก "1" อยู่ขา CEP,CET จะแอคตีฟที่ลอจิก "0" จะเป็นลอจิก "0" ได้ก็ต่อเมื่อมีการกดสวิทช์ขึ้นหรือกดสวิทช์ลง แต่การกดสวิทช์จะมีเงื่อนไขเมื่อ Y0 เป็นลอจิก "0" จะนับลงไม่ได้แต่เป็นลอจิก "1" จะนับลงได้ และ Y14 เป็นลอจิก "0" จะนับขึ้นไม่ได้ จะนับขึ้นได้ก็ต่อเมื่อเป็นลอจิก "1" การใช้ไอซีนับขึ้นหรือนับลง จะกำหนดจากขา U/D เมื่อขา U/D เป็นลอจิก "1" วงจรจะนับขึ้นถ้าเป็น ลอจิก "0 "จะนับลง

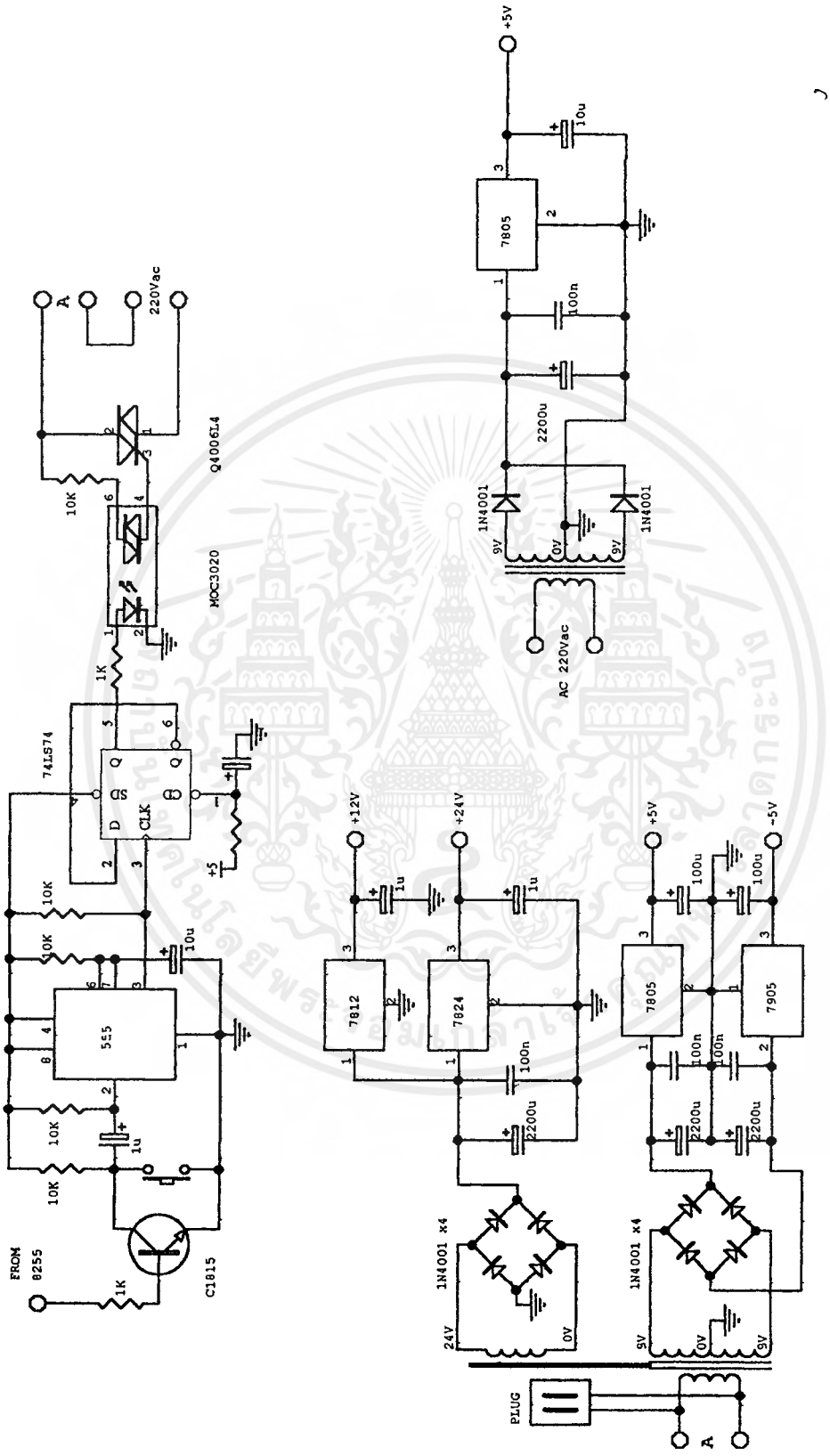
การสั่งงานจากคอมพิวเตอร์ สั่งผ่านพอร์ทเอาท์พุทของ MCS-51 ที่ พอร์ท P10 - P17 โดยการต่อตัวต้านทานค่า 1 กิโลโอห์ม อนุกรมระหว่างพอร์ทและขาเบสของทรานซิสเตอร์ ส่วนขาคอลเลคเตอร์และขาอีมิตเตอร์ของทรานซิสเตอร์ต่อคร่อมกับสวิทช์ชนิดกดติดปล่อยดับของปุ่ม UP และ DOWN เมื่อต้องการให้เครื่องทำงานในหน้าที่ใดก็ให้พอร์ททที่ต่อกับสวิทช์ที่ทำหน้าที่นั้น ๆ มีสถานะไฮ (LOGIC "1") จะทำให้ทรานซิสเตอร์นำกระแส เครื่องก็จะทำงานตามหน้าที่นั้น ๆ

-การทำงานภาค INPUT SELECTER

ไอซี 555 ต่อเป็นวงจรโมโนสเตเบิ้ล MONOSTABLE สร้างสัญญาณกระตุ้นโดยทริกที่ขา 2 ของ ไอซี 555 โดยทริกจากการกดสวิทช์ และ ทริกจากทรานซิสเตอร์ เบอร์ BC 557 เมื่อนำกระแส เอาท์พุทจะออกที่ขา 3 เป็นสัญญาณกระตุ้นต่อให้ ดีฟลิปฟลอป (D-FLIPFLOP) ทั้ง 4 ตัว คือ ไอซี เบอร์ 74LS74 จะมีดีฟลิปฟลอปอยู่ 2 ตัวในไอซี 1 ตัว นำดีฟลิปฟลอปทั้งหมดมาต่ออนุกรมกัน ก็คือ นำ Q ของดีฟลิปฟลอปตัวแรก (Q1) มาต่อกับขา D ของดีฟลิปฟลอปตัวที่ 2 (D2) และ Q2 ต่อกับ D3 และ Q3 ต่อกับ D4 แต่ Q4 นำมาต่อกับ D1 เอาท์พุท Q ของดีฟลิปฟลอปทุกตัวเป็น เอาท์พุทในการนำไปเลือกรับสัญญาณจากแหล่งกำเนิดสัญญาณต่าง ๆ คือ เทป (TAPE) , วิทยุ (TUNER) , ซีดี (CD) , AUX แต่ละแหล่งกำเนิดสัญญาณจะมี 2 ช่องสัญญาณ คือ สัญญาณขวาและสัญญาณซ้าย การนำสัญญาณไปยังภาคปรีแอมป์โดยใช้รีเลย์ (RELAY) โดยสัญญาณ " 1 " จากขา Q จะมีแรงดันประมาณ 5 โวลท์ เป็นตัวทำให้ทรานซิสเตอร์นำกระแส จึงมีกระแสไหลผ่านขดลวดรีเลย์ รีเลย์จึงทำงาน ที่ขดลวดของรีเลย์ต่ออยู่กับไดโอด โดยต่อแบบไบอัสกลับ (REVERSE BIAS) เพื่อป้องกันความเสียหายเนื่องจากแรงดันย้อนกลับของขดลวดขณะขดลวดถูกทำให้กระแสหยุดไหล ขณะที่เปิดเครื่อง เอาท์พุท Q1 จะเป็นลอจิก " 1 " เนื่องจากได้ต่อขาปรีเซ็ทกับ ตัวต้านทานและตัวเก็บประจุดังรูปเพื่อให้เป็นลอจิก " 0 " ชั่วขณะที่เปิดเครื่อง และ Q ที่เหลือขณะเปิดเครื่องจะเป็นลอจิก " 0 " เนื่องจากต่อขาเคลีย (CLEAR) ไว้กับตัวต้านทานและตัวเก็บประจุเช่นกัน เมื่อมีการกระตุ้น 1 ครั้ง หลังจากเปิดเครื่อง ข้อมูลจะเลื่อนไปที่ Q ถัดไป ก็คือ Q2 เป็น " 1 " และ Q ที่เหลือเป็นลอจิก " 0 " เมื่อมีสัญญาณมากระตุ้นอีกก็จะเลื่อนไปเรื่อย ๆ ที่ Q3 และ Q4 และวกกลับมาที่ Q1 อีกครั้ง



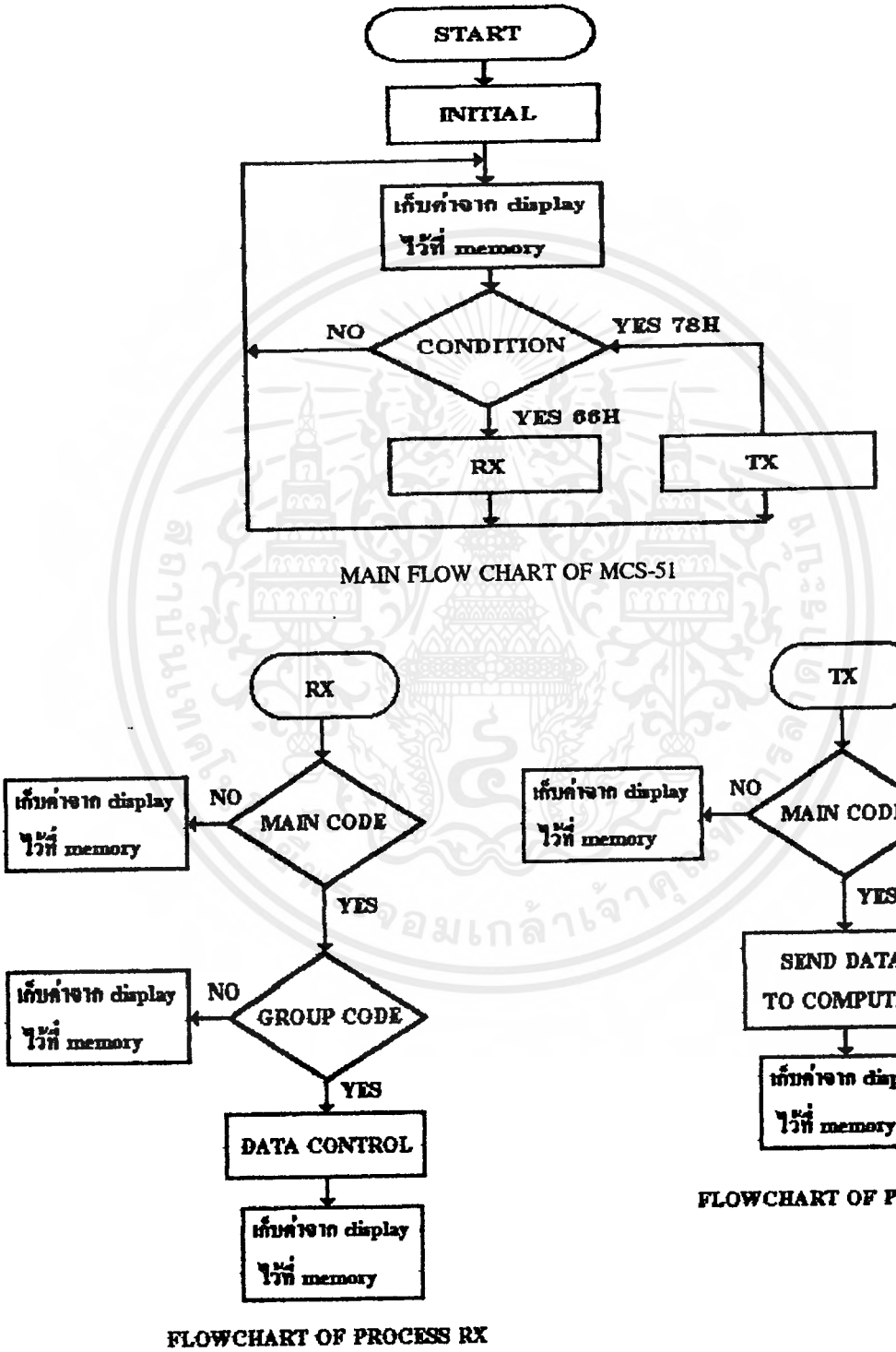
รูปที่ 6.13 วงจรการเลือกใช้ช่องสัญญาณของ PRE-TONE AMPIFIER



รูปที่ 6.14 SUPPLY AMP CONTROL

6.3 วิธีการรับส่งข้อมูลระหว่างคอมพิวเตอร์กับ MCS-51

6.3.1 โฟล์วชาร์ต การทำงานของ MCS-51



รูปที่ 6.15 แสดงการทำงานของ MCS-51

6.3.2 รหัสที่ใช้ในการรับ/ส่งข้อมูล

Delphi	MCS-51	การทำงาน	รหัสเครื่อง	TUNER		EQ			AMP		TAPE
F1	06		TUNER	Group 1		Group 1			Group 1		
F2	18		EQ	Group 2		Group 2			Group 2		
F3	1E		AMP			Group 3					
F4	60		TAPE								
F5	66	รับ									
F6	78	ส่ง									
F7	7E			M1	memory	63 ↑	500 ↑	7k ↑	Bass ↑	Select	Play-L
F8	80			M2	Auto	63 ↓	500 ↓	7k ↓	Bass ↓	Power	Play-R
F9	86			M3	manual	100 ↑	1k ↑	14k ↑	Tre ↑		F/F
FA	98			M4	Up	100 ↓	1k ↓	14k ↓	Tre ↓		REW
FB	9E			M5	Down	200 ↑	2k ↑	By-pass	Vol ↑		REC - MUTE
FC	E0			M6	Power	200 ↓	2k ↓	Power	Vol ↓		REC
FD	E6			M7		330 ↑	3k3 ↑		Bal-L		PAUSE
FE	F8			M8		330 ↓	3k3 ↓		Bal-R		STOP
FF	FE										POWER

ตารางที่ 6.1 แสดงรหัสที่ใช้ในการรับ-ส่งข้อมูล

จากตารางที่ 6.1 จะแสดงรหัสที่ใช้ในการรับ-ส่งข้อมูลจากการทดลองในบทที่ 7 รหัสที่โปรแกรมเดลไฟส่งมาจะไม่ตรงกับรหัสที่ปรากฏบน MCS-51 เช่น โปรแกรมเดลไฟส่ง F1 H ที่ MCS-51 จะได้รับรหัส 06 H และเช่นเดียวกันขณะที่ MCS-51 ส่ง 06 H โปรแกรมเดลไฟก็จะรับรู้เป็น F1 H

จากการลักษณะการรับ-ส่งข้อมูลข้างต้น จะมีการกำหนดรหัสเพื่อใช้ในการรับ-ส่งข้อมูลให้รหัสในการรับเป็น F8 H (66 H), รหัสในการส่งเป็น F6 H (78 H) สัญญาณข้อมูลต่อมาจะเป็นการส่งรหัสเครื่อง โดยใช้รหัส F1 H (06 H) เป็นจูนเนอร์, รหัส F2 H (18 H) เป็นอีควอไลเซอร์, รหัส F3 H (1E H) เป็นปรีแอมป์, รหัส F4 H (60 H) เป็นเครื่องเล่นเทปและสัญญาณข้อมูลต่อไปจะบอกถึงรหัสกลุ่ม (Group Code) และรหัสคำสั่งที่ต้องการควบคุม

6.3.3 การรับข้อมูลของ MCS-51 เมื่อมีการควบคุมจากคอมพิวเตอร์

เมื่อมีการสั่งงานที่คอมพิวเตอร์เพื่อการควบคุมการทำงานใด ๆ คอมพิวเตอร์ก็จะส่งรหัสข้อมูลของการรับ (F5H) เข้าไปยัง MCS-51 ก่อน เมื่อ MCS-51 ได้รับรหัสข้อมูลการรับ (F5H) ก็จะเตรียมตัวรับสัญญาณต่อไป คอมพิวเตอร์จะส่งรหัสข้อมูลของเครื่องที่ต้องการควบคุม {F1H เป็นรหัสเครื่องของ TUNER, F2H เป็นรหัสเครื่องของ EQUALIZER, F3H เป็นรหัสเครื่องของ PRE-TONE AMPLIFIER, F4H เป็นรหัสเครื่องของ TAPE} เป็นสัญญาณต่อไป ส่วนที่ MCS-51 เมื่อรับรหัสข้อมูล {06H เป็นรหัสเครื่องของ TUNER, 18 เป็นรหัสเครื่องของ EQUALIZER, 1E เป็นรหัสเครื่องของ PRE-TONE AMPLIFIER, 60 เป็นรหัสเครื่องของ TAPE ว่าเป็นเครื่องใด ถ้าเป็นรหัสของตัวมันเองก็จะไปรอทำงานต่อไป หากไม่ใช่รหัสเครื่องของตัวมัน ก็จะกลับไปทำการเก็บข้อมูล (MEMORY)

จากนั้นคอมพิวเตอร์ก็จะส่งสัญญาณที่เป็นข้อมูลของการสั่งงานของเครื่องนั้น ๆ โดยที่ PRE-TONE AMPLIFIER มีการทำงานทั้งหมด 10 การทำงาน การทำงานทั้งหมดของ PRE-TONE APMLIFIER จะถูกแบ่งเป็น 2 กลุ่มคำสั่ง คือ GROUP1, GROUP2 แต่ที่ TAPE มีการทำงานทั้งหมด 9 การทำงาน การทำงานทั้งหมดไม่ต้องแบ่งกลุ่มคำสั่ง รหัสข้อมูลต่อมาก็จะเป็นรหัสข้อมูลของการควบคุมการทำงานใด ๆ และเมื่อทำงานตามที่ต้องการ คอมพิวเตอร์ก็จะรอการทำงานต่อไป MCS-51 ก็จะไปเก็บข้อมูล (MEMORY) ต่อไป

ตำแหน่งที่ CLICK ที่ คอมพิวเตอร์	รหัสที่เขียนส่งจาก คอมพิวเตอร์ (H)	รหัสที่ปรากฏที่ MCS-51 (H)	พอร์ทที่ทำงาน ของ 8255 (H)	การทำงานของอุปกรณ์
BASS UP	F5,F3,F1,F7	66,1E,06,7E	P1.3	เพิ่มระดับสัญญาณความถี่ต่ำ
BASS DOWN	F5,F3,F1,F8	66,1E,06,80	P1.2	ลดระดับสัญญาณความถี่ต่ำ
THEBLE UP	F5,F3,F1,F9	66,1E,06,86	P1.1	เพิ่มระดับสัญญาณความถี่สูง
THEBLE DOWN	F5,F3,F1,FA	66,1E,06,98	P1.0	ลดระดับสัญญาณความถี่สูง
VOLUME UP	F5,F3,F1,FB	66,1E,06,9E	P1.5	เพิ่มระดับขนาดสัญญาณ
VOLUME DOWN	F5,F3,F1,FC	66,1E,06E0	P1.4	ลดระดับขนาดสัญญาณ
BALANCE-L	F5,F3,F1,FD	66,1E,06,E6	P1.6	เพิ่มอัตราส่วนระดับเสียงไปทางซ้าย
BALANCE-R	F5,F3,F1,FE	66,1E,06,F8	P1.7	เพิ่มอัตราส่วนระดับเสียงไปทางขวา
SELECT	F5,F3,F2,F7	66,1E,18,7E	PC.7	เลือกสัญญาณเสียงจากแหล่งกำเนิด
POWER	F5,F3,F2,F8	66,1E,18,80	PC.4	ON / OFF

ตารางที่ 6.2 แสดงผลจากการ CLICK ปุ่มต่าง ๆ ที่คอมพิวเตอร์และการส่งรหัสสัญญาณ ณ. จุดต่าง ๆ ของ PRE-TONE AMPLIFIER

ตำแหน่งที่ CLICK ที่ คอมพิวเตอร์	รหัสที่เขียนส่ง จากคอม-พิวเตอร์ (H)	รหัสที่ปรากฏที่ MCS-51 (H)	พอร์ท ที่ ทำงาน ของ 8255 (H)	การทำงานของอุปกรณ์
PLAY-L	F5,F4,F7	66,60,7E	PA.4	เทป PLAY ทางซ้าย (PLAY REVERSE)
PLAY-R	F5,F4,F8	66,60,80	PA.3	เทป PLAY ทางขวา (PLAY FORWARD)
FORWARD	F5,F4,F9	66,60,86	PA.2	FORWARD เทป
REVERSE	F5,F4,FA	66,60,98	PA.5	REVERSE เทป
REC-MUTE	F5,F4,FB	66,60,9E	PA.0	MUTE การ RECORD
RECORD	F5,F4,FC	66,60,E0	PA.6	ทำการ RECORD
PAUSE	F5,F4,FD	66,60,E6	PA.1	ทำการ PAUSE
STOP	F5,F4,FE	66,60,F8	PA.7	หยุดการทำงานของเทปในสถานะนั้น ๆ
POWER	F5,F4,FF	66,60,FE	P1.0	ON / OFF

ตารางที่ 6.3 แสดงผลจากการ CLICK ปุ่มต่าง ๆ ที่คอมพิวเตอร์และการส่งรหัสสัญญาณ ณ. จุดต่างๆ ของ เครื่องเล่น TAPE

6.3.4 การส่งข้อมูลจาก MCS-51 เพื่อแสดงผลบนจอคอมพิวเตอร์

รหัสที่ Computer	ทิศทางการส่ง	รหัสที่ mcs-51	การทำงาน Computer	MCS-51
F6	→	78	ส่งรหัสการส่ง	ถ้าเป็นรหัสการรับเตรียมรับค่าต่อ
Fc	→	Code	ส่งรหัสเครื่อง	ถ้าเป็นรหัสเครื่องทำงานต่อ, ถ้าไม่ใช่กลับไปเก็บข้อมูล
mem	←	70H(xx)	รับรหัสข้อมูลในหน่วยความจำและแสดงผล	ส่งข้อมูลที่เก็บไว้ในหน่วยความจำแรก(70H)
F0	→	00	ส่งรหัสบอกว่ารับข้อมูลเสร็จแล้ว	รอรับรหัสเพื่อส่งข้อมูลต่อไป
mem	←	71H(xx)	รับรหัสข้อมูลในหน่วยความจำและแสดงผล	ส่งข้อมูลที่เก็บไว้ต่อไป(71H)
F0	→	00	ส่งรหัสบอกว่ารับข้อมูลเสร็จแล้ว	รอรับรหัสเพื่อส่งข้อมูลต่อไป
mem	←	7nH(xx)	รับรหัสข้อมูลในหน่วยความจำและแสดงผล	ส่งข้อมูลสุดท้ายที่เก็บไว้(7nH)
F0	→	00	ส่งรหัสบอกว่ารับข้อมูลเสร็จแล้ว และรอการทำงานรอบต่อไป	กลับไปเก็บข้อมูล

ตารางที่ 6.4 แสดงขั้นตอนการส่งข้อมูลจาก MCS-51 เพื่อแสดงผลที่คอมพิวเตอร์

ในการส่งข้อมูลจาก MCS-51 เพื่อแสดงผลบนจอคอมพิวเตอร์ทำโดยส่งรหัสการส่ง (F6H) เมื่อ MCS-51 รับรหัสการส่ง (78H) ได้จะเตรียมรับสัญญาณรหัสเครื่องว่าคอมพิวเตอร์ต้องการติดต่อกับอุปกรณ์ใด ที่ MCS-51 จะได้รับรหัสเครื่อง (Code) ถ้าเป็นรหัสเครื่องของตัวมันเองก็จะส่งข้อมูลที่เก็บไว้ในหน่วยความจำแรกที่อยู่ในแอดเดรส 70H ไปยังคอมพิวเตอร์และจะมีรหัสจากคอมพิวเตอร์ว่ารับเสร็จเรียบร้อยก่อนจึงจะส่งข้อมูลที่เก็บไว้ต่อไป ที่คอมพิวเตอร์เมื่อรับข้อมูลมาเก็บไว้ในหน่วยความจำ (Memory) และนำค่าไปเปรียบเทียบกับค่าที่ตั้งเก็บไว้เพื่อแสดงผลและจะส่งรหัส (F0H) เพื่อบอกว่ารับข้อมูลเรียบร้อยแล้วและให้ส่งข้อมูลต่อไปมา จะทำเช่นนี้จนถึงข้อมูลสุดท้าย เมื่อส่งรหัส (F0H) เพื่อบอกว่ารับเสร็จเรียบร้อยแล้วก็จะรอเวลาเพื่อทำงานในรอบต่อไป ที่ MCS-51 ก็จะกลับไปทำงานในการเก็บค่าที่จะส่งมาแสดงผลที่คอมพิวเตอร์ใหม่อีกครั้ง รอบเวลาการทำงานถูกตั้งค่าโดยโปรแกรมเคลไฟล์คือ ปรี-โทนแอมป์ปลิไฟเออร์ทำการแสดงค่าทุก 4/10 วินาที, อีควอไลเซอร์แสดงค่าทุก 3วินาที, เทปแสดงค่าทุก 4/10 วินาที และจูนเนอร์แสดงค่าทุก 7/10 วินาที

รหัสเครื่องที่ส่งจากคอมพิวเตอร์เพื่อเลือกติดต่อกับอุปกรณ์เครื่องใดและรหัสที่ MCS-51 รับได้ (F6H/Code) สำหรับปรี-โทนแอมป์ปลิไฟเออร์คือ F3H/1E, อีควอไลเซอร์คือ F2H/18, เทป คือ F4H/60 และจูนเนอร์คือ F1H/06

ข้อมูลที่ MCS-51 ส่งไปแสดงผลที่คอมพิวเตอร์จะส่งจากแอดเดรส 70H จนถึง 7nH คือ แอดเดรสสุดท้ายของเครื่องนั้น ๆ และที่โปรแกรมเคลไฟล์จะเก็บข้อมูลไว้ที่หน่วยความจำต่าง ๆ (memory) ที่จูนเนอร์จะเก็บที่ AA, AB, AC, AD,.....AK ซึ่งหมายความว่าจูนเนอร์จะมีข้อมูลที่ส่งมาเพื่อแสดง 11 ค่าโดย 11 ค่านี้จะมาจาก MCS-51 ที่แอดเดรส 70H ถึก 7AH ที่ปรี-โทนแอมป์ปลิไฟเออร์ จะเก็บที่ DA ถึง DF มี 6 ค่าจาก MCS-51 แอดเดรสที่ 70H ถึง 75H อีควอไลเซอร์เก็บไว้ที่ CA ถึง CL มี 12 ค่าจาก MCS-51 แอดเดรส 70H ถึง 7BH และเทปจะเก็บไว้ที่ BA ถึง BH มี 8 ค่าจาก MCS-51 แอดเดรส 70H ถึง 77H รหัสข้อมูลที่ส่งไปยังคอมพิวเตอร์เพื่อแสดงผลในตำแหน่งต่าง ๆ (70H(xx)ถึง 7nH(xx)) จาก MCS-51 แสดงดังต่อไปนี้

PRE-TONE AMPLIFIER		ข้อมูลที่ MCS-51 ส่ง (xx H) LED ดวงที่ติด														
แอดเดรส	แสดงผล	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
70H	TREBLE	60	18	1E	60	66	78	7E	80	86	98	9E	E0	E6	F8	FE
71H	BASS	60	18	1E	60	66	78	7E	80	86	98	9E	E0	E6	F8	FE
72H	VOLUME	60	18	1E	60	66	78	7E	80	86	98	9E	E0	E6	F8	FE
73H	BALANCE	60	18	1E	60	66	78	7E	80	86	98	9E	E0	E6	F8	FE
74H	POWER	ON-06 , OFF-18														
75H	I/P SELECT	AUX-06 , TUNER-18 , TAPE - 60														

ตารางที่ 6.5 แสดงข้อมูลที่ MCS-51 ส่งเพื่อแสดงผลบนจอคอมพิวเตอร์ในหน้าที่ต่าง ๆ จาก Pre-Tone Amplifer

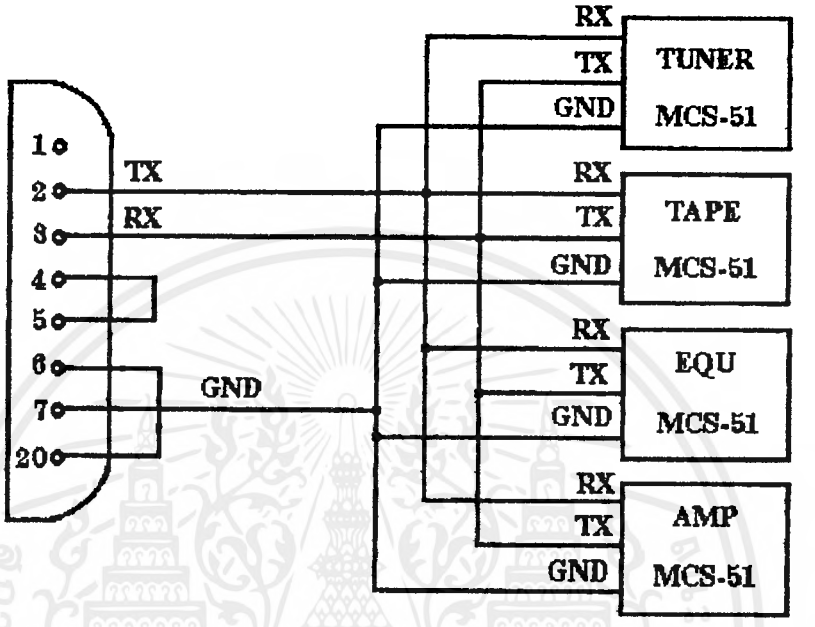
TAPE		ข้อมูลที่ MCS-51 ส่ง (xx H)	
แอดเดรส	แสดงผล	ติด	ไม่ติด
70H	RECORD	60	18
71H	PLAY-L	60	18
72H	PLAY-R	60	18
73H	PLAY	60	18
74H	PAUSE	60	18
75H	REC-MUTE	60	18
76H	LEVEL	LEVEL 1-60 , 2-18 , 3-1E , 4-60 , 5-66 , 6-78 , 7-7E , 8-80	

ตารางที่ 6.6 แสดงข้อมูลที่ MCS-51 ส่งเพื่อแสดงผลบนจอคอมพิวเตอร์ในหน้าที่ต่าง ๆ จากเครื่องเล่นเทป

บทที่ 7 บทสรุป

7.1 ผลการทดลอง

7.1.1 การต่อพอร์ทอนุกรมที่ 2 (COM2) ใช้งานกับเครื่องทั้ง 4 เครื่อง



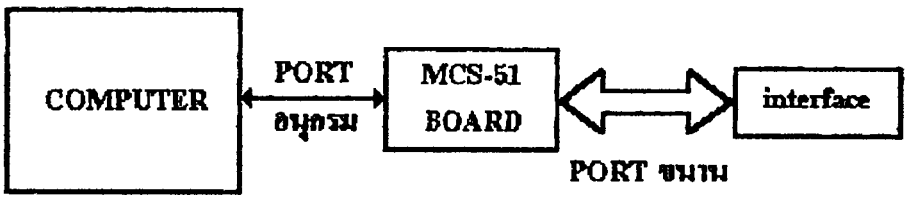
รูปที่ 7.1 วงจรการต่อใช้งานจริง

การต่อใช้งานจะต่อตามรูป โดยออกมาจาก COM2 เพียง 3 เส้นต่อขา 2 (TX) ขา 3 (RX) ขา 7 (GND) นำขา 2 (RX) COM2 ไปต่อเข้าขา RX ของ MCS-51 ของทุกเครื่อง และนำขา 3 (RX) ไปต่อเข้าขา TX ของ MCS-51 ของทุกเครื่อง และขา 7 (GND) นำไปต่อขากราวด์ของ MCS-51 ของทุกเครื่องเช่นกัน

ขา 5 และขา 4 ของ COM2 ต่อถึงกัน ขา 4 เป็นขา O/P ขา 5 เป็นขา I/P ใช้ตรวจสอบแฮนด์เช็คของสายสัญญาณของ COM2 ขา 6 และขา20 ของ COM2 ต่อถึงกัน ขา 6 เป็นขา I/P ขา 20 เป็นขา O/P ใช้ในการทำแฮนด์เช็คของ COM2

7.1.2 การทดสอบระหัสที่คอมพิวเตอร์ส่งมาให้ MCS-51 แล้ว MCS-51 มองเห็นเป็นอะไร

ในการตรวจสอบระหัสจากคอมพิวเตอร์นั้นสำคัญมาก ต้องหาระหัสที่ถูกต้องนำมาใช้งาน โดยการตรวจสอบจากการต่อ COM2 มาเข้า พอร์ทอนุกรมของ MCS-51 โดยต่อเพียงขา RX กับ GND ตามรูป และเขียนโปรแกรมบน DELPHI จากโปรแกรมที่ 7.1.4 และเขียนโปรแกรมให้ MCS-51 รับระหัสจาก COM2 ขา 7 พอร์ทอนุกรมและนำมาแสดงทางพอร์ท A , พอร์ท B ของ 8255 ตัวที่ 1 ตามโปรแกรมที่ 7.1.3



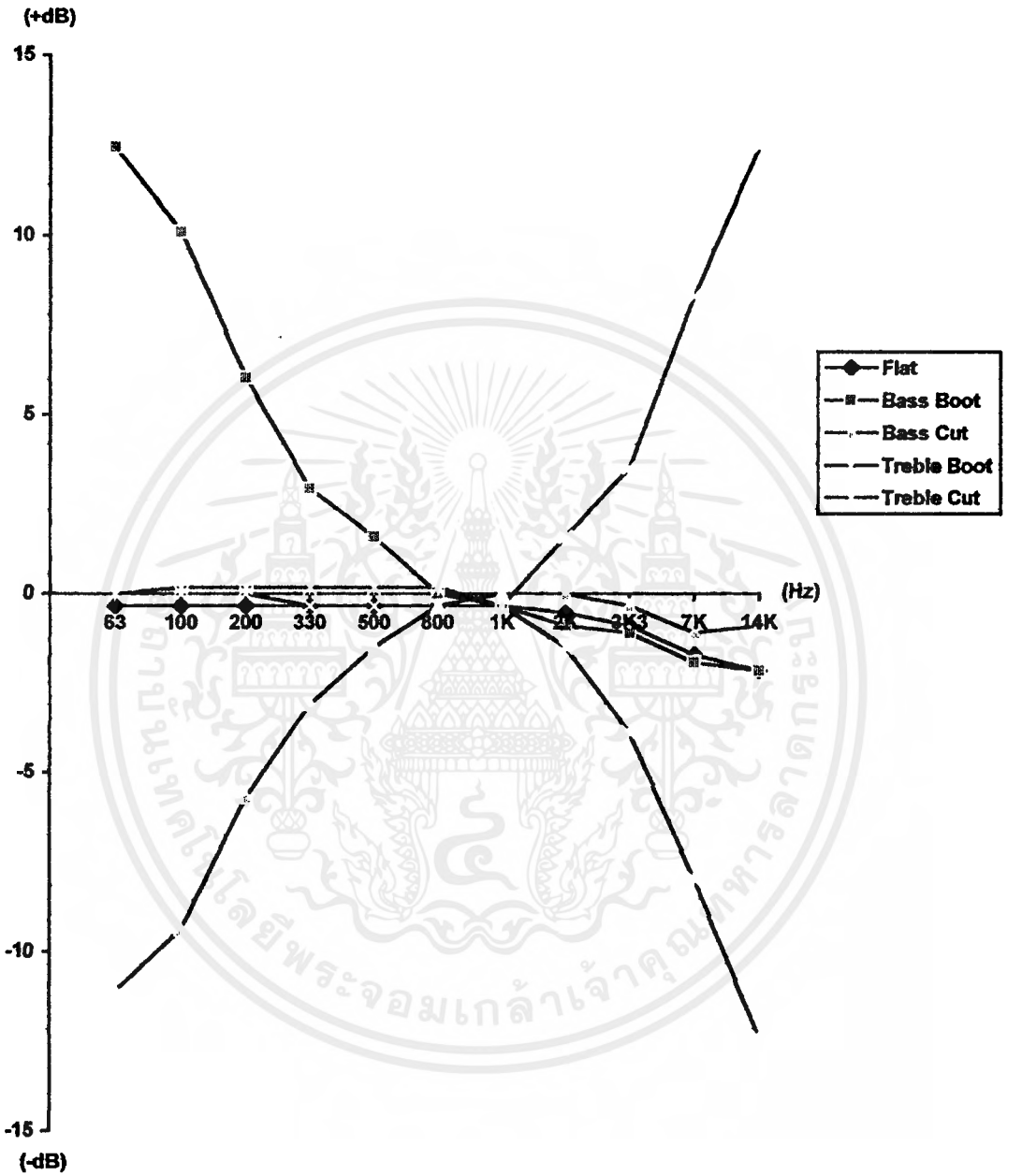
รูปที่ 7.2 วงจรตรวจสอบระหัส

เมื่อทำการรันโปรแกรมทั้ง Delphi และ MCS-51 ระหัสที่ MCS-51 รับรู้ได้จะปรากฏที่พอร์ท A และ พอร์ท B byte หลังจะอยู่ที่พอร์ท A และ byte แรกจะอยู่ที่ พอร์ท B จากการทดลองโปรแกรมดังกล่าวจะเกิดมีการแสดงผลทั้งพอร์ท A และพอร์ท B หรือแสดงผลเพียงพอร์ท A เท่านั้น โดยที่เขียนโปรแกรมรับค่าเพียง 8 bit เท่านั้น อาจสรุปได้ว่าการส่งระหัสจากคอมพิวเตอร์จาก 00H-FFH นั้นจะมีบางระหัสเป็น 2 byte และบางระหัสเป็น 1byte แต่ในที่นี้จะตัดระหัสเพียง 8 bit มาใช้งานเท่านั้นระหัสที่ใช้งานได้จะเลือกได้จากตารางที่ 7.1

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	00	06	18	1E	60	66	78	00	00	00	00	00	00	00	00	00
								80	86	98	9E	E6	9E	E6	F8	FE
1	60	60	60	60	60	60	60	60	60	60	60	60	60	60	60	60
	00	06	18	1E	60	66	78	7E	80	86	98	9E	E0	E6	F8	FE
2	78	78	78	78	78	78	78	78	06	06	06	06	06	06	06	06
	00	06	18	1E	60	66	78	7E	80	86	98	9E	E0	E6	F8	FE
3	78	78	78	78	78	78	78	78	78	78	78	78	78	78	78	78
	00	06	18	1E	60	66	78	7E	80	86	98	9E	E0	E6	F8	FE
4	7E	7E	7E	7E	7E	7E	7E	7E	18	18	18	18	18	18	18	18
	00	06	18	1E	60	66	78	7E	80	86	98	9E	E0	E6	F8	FE
5	66	66	66	66	66	66	66	66	66	66	66	66	66	66	66	66
	00	06	18	1E	60	66	78	7E	80	86	98	9E	E0	E6	F8	FE
6	7E	7E	7E	7E	7E	7E	7E	7E	1E	1E	1E	1E	1E	1E	1E	1E
	00	06	18	1E	60	66	78	7E	80	86	98	9E	E0	E6	F8	FE
7	7E	7E	7E	7E	7E	7E	7E	7E	7E	7E	7E	7E	7E	7E	7E	7E
	00	06	18	1E	60	66	78	7E	80	86	98	9E	E0	E6	F8	FE
8	00	06	18	1E	60	66	78	7E	60	60	60	60	60	60	60	60
									80	86	98	9E	E0	E6	F8	FE
9	78	78	78	78	78	78	78	78	78	78	78	78	78	78	78	78
	00	06	18	1E	60	66	78	7E	80	86	98	9E	E0	E6	F8	FE
A	7E	7E	7E	7E	7E	7E	7E	7E	66	66	66	66	66	66	66	66
	00	06	18	1E	60	66	78	7E	80	86	98	9E	E0	E6	F8	FE
B	7E	7E	7E	7E	7E	7E	7E	7E	7E	7E	7E	7E	7E	7E	7E	7E
	00	06	18	1E	60	66	78	7E	80	86	98	9E	E0	E6	F8	FE
C	00	06	18	1E	60	66	78	7E	78	78	78	78	78	78	78	78
									80	86	98	9E	E0	E6	F8	FE
D	7E	7E	7E	7E	7E	7E	7E	7E	7E	7E	7E	7E	7E	7E	7E	7E
	00	06	18	1E	60	66	78	7E	80	86	98	9E	E0	E6	F8	FE
E	00	06	18	1E	60	66	78	7E	7E	7E	7E	7E	7E	7E	7E	7E
									80	86	98	9E	E0	E6	F8	FE
F	00	06	48	1E	60	66	78	7E	80	86	98	9E	E0	E6	F8	FE

ตารางที่ 7.1 ระหัสที่ คอมพิวเตอร์ส่งมาแล้ว MCS-51 มองเห็น

7.1.3 ผลการทอสอบการตอบสนองความถี่เสียงของ PRE-TONE AMPLIFIER



รูปที่ 7.3 แสดงการตอบสนองความถี่เสียงของ PRE-TONE AMPLIFIER

- เมื่อไม่มีการเพิ่มและลดอัตราขยายความถี่ (Flat)
- เมื่อเพิ่มอัตราขยายความถี่ต่ำ (Boot Bass)
- เมื่อลดอัตราขยายความถี่ต่ำ (Cut Bass)
- เมื่อเพิ่มอัตราขยายความถี่สูง (Boot Treble)
- เมื่อลดอัตราขยายความถี่สูง (Cut Treble)

7.2 ปัญหาและอุปสรรค

- รหัสการติดต่อระหว่างไมโครคอนโทนเลอร์ (MCS-51) กับคอมพิวเตอร์ (PC) ยังไม่สามารถติดต่อกันได้อย่างสมบูรณ์ คือสามารถใช้รหัสการติดต่อได้เพียง 15 รหัสเท่านั้น (ตั้งแต่ F1 ถึง FF)

- ยังไม่สามารถแปลงรหัสบนโปรแกรมเคลไฟล์ที่ส่งออกไปให้ ไมโครคอนโทนเลอร์บอร์ด (MCS-51) ให้เข้าใจตรงกันได้ตามที่ต้องการได้ทั้งหมด

- เนื่องจากข้อมูลของโปรแกรมเคลไฟล์ที่มีอยู่ในท้องตลาดมีไม่มากพอ ทำให้เป็นปัญหาต่อการเขียนโปรแกรมควบคุมต่างๆ

- วงจรภาคควบคุมการทำงานของเครื่องเล่นเทปจากคอมพิวเตอร์ของเครื่องเล่นเทปที่ได้ออกแบบไว้ในครั้งแรกไม่สามารถใช้งานได้เนื่องจากอนาลอกสวิทช์ (4066) ที่ใช้ต่อขนานกับสวิทช์ที่ควบคุมของตัวเครื่องมีกระแสรั่วไหลจนทำให้เทปไม่สามารถทำงานได้ตามปกติ แก้โดยใช้ทรานซิสเตอร์แทน อนาลอกสวิทช์

- ที่ปรี-โทนคอนโทรลแอมป์ปริไฟล์ขณะเปิดเครื่องที่ บัลลาซท์ (BALANCE) , เสียงทุ้ม (BASS) และ เสียงแหลม (THEBLE) จะมีระดับที่ตรงกลาง หรือที่ลอจิก 0111 นั่นเอง แต่ถ้าปิดแล้วเปิดเครื่องเร็ว ๆ (ทิ้งระยะเวลาเพียงเล็กน้อย) เมื่อเปิดเครื่องที่ตำแหน่งดังกล่าวจะไม่อยู่ตรงกลางหรือลอจิก 0111 เนื่องจากการทำให้เอาท์พุทเป็นลอจิก 0111 ในขณะเปิดเครื่อง ทำโดยการให้ขาโหลดของไอซีนับขึ้นลงเบอร์ 74LS169 ให้เป็นลอจิก "0" ชั่วขณะเพื่อให้โหลดค่าที่ตั้งไว้ออกที่เอาท์พุทของตัวมัน โดยการการต่อตัวต้านทานจากขาโหลดกับแหล่งจ่ายค่า 10 กิโลโอห์ม และ ตัวเก็บประจุค่า 10 ไมโครฟารัดกับกราวด์ ขณะปิดเครื่องตัวเก็บประจุไม่สามารถคายประจุได้ทันที่ขาโหลดจึงไม่เป็นลอจิก 0 จึงไม่โหลดค่าที่ตั้งไว้ แก้โดยการต่อไดโอดแบบไบอัสกลับขนานกับตัวต้านทานและเพิ่มค่าของตัวต้านทานเพื่อเพิ่มเวลาในการให้ลดข้อมูลให้นานขึ้น ซึ่งปัญหานี้ก็พบที่ INPUT SELECTER เช่นกันและได้แก้ปัญหาเหมือนกัน

7.3 แนวทางการแก้ปัญหา

- ใช้รหัสที่สามารถใช้ได้ (F1 ถึง FF) แล้วให้ไมโครคอนโทนเลอร์แปลงรหัสเพื่อให้ได้ค่าที่ต้องการ

- ใช้โลจิกอานาไลเซอร์วัดข้อมูลในสายข้อมูลอนุกรมเพื่อดูว่าถูกต้องหรือไม่

- ศึกษาวิธีการแปลงรหัสเพื่อให้ได้รหัสที่ถูกต้อง

- ศึกษาโปรแกรม Delphi เพิ่มขึ้นเพื่อใช้ในการแก้ปัญหาในการเขียนโปรแกรม

- วงจรภาคควบคุมการทำงานของเครื่องเล่นเทปจากคอมพิวเตอร์ของเครื่องเล่นเทปที่ได้ออกแบบไว้ในครั้งแรกไม่สามารถใช้งานได้เนื่องจากอนาลอกสวิทช์ (4066) ที่ใช้ต่อขนานกับสวิทช์ท่ควบคุมของตัวเครื่องมีกระแสรั่วไหลจนทำให้เทปไม่สามารถทำงานได้ตามปกติ แก้โดยใช้ทรานซิสเตอร์แทน อนาลอกสวิทช์

7.4 สรุป

ในโครงงานการควบคุมระบบเสียงด้วยคอมพิวเตอร์นี้ (COMPUTER-CONTROL SOUND SYSTEM) เป็นการศึกษาวิธีการควบคุมเครื่องเสียงด้วยคอมพิวเตอร์วิธีหนึ่ง โดยการควบคุมวิธีนี้เป็นการควบคุมโดยใช้โปรแกรมเดลไฟล์ (DELPHI) โดยสร้างอุปกรณ์ที่มีในคอมโปเนนต์ที่มีในเดลไฟล์มาใช้และเขียนโปรแกรมเพื่อให้โปรแกรมเดลไฟติดต่อรับ-ส่งข้อมูลต่าง ๆ เพื่อควบคุมอุปกรณ์ต่าง ๆ และเพื่อการแสดงผลที่คอมพิวเตอร์ ข้อมูลต่าง ๆ จะผ่านพอร์ท อนุกรมของคอมพิวเตอร์ ในโครงงานนี้ใช้ พอร์ทอนุกรมที่ 2 (COM2) เพื่อติดต่อกับไมโครคอลโทรเลอร์เบอร์ MCS-51 ซึ่งถูกโปรแกรมเพื่อให้รับ-ส่งข้อมูลกับคอมพิวเตอร์ และติดต่อกับไอซี 8255 เป็นอินพุทพอร์ทและเอาท์พุทพอร์ท เพื่อรับข้อมูลมาส่งออกที่พอร์ทไปควบคุมการทำงานของอุปกรณ์ต่าง ๆ และส่งค่าหรือข้อมูลเพื่อแสดงผลที่คอมพิวเตอร์ อุปกรณ์ที่ใช้ในโครงงานนี้คือจูนเนอร์ , อีควอไลเซอร์ , เทป , ปรี-โทนคอนโทรล และยังสามารถเพิ่มอุปกรณ์เพื่อควบคุมการทำงานได้เพิ่มขึ้นอีกโดยการเขียนโปรแกรมเพิ่มเติม ในกระบวนการรับ-ส่งข้อมูล สัญญาณข้อมูลที่ใช้จะประอบด้วยส่วนต่าง ๆ คือ สัญญาณข้อมูลแจ้งการรับหรือการส่ง, สัญญาณข้อมูลเพื่อแจ้งการเลือกอุปกรณ์ที่คอมพิวเตอร์ต้องการติดต่อด้วย, สัญญาณข้อมูลที่ต้องการสื่อสาร โดยสัญญาณ ทั้งสามจะถูกควบคุมจากคอมพิวเตอร์

วิธีการควบคุมในโครงงานนี้เป็นเพียงวิธีการหนึ่งเท่านั้น การควบคุมนี้สามารถประยุกต์เพื่อใช้ในการควบคุมอุปกรณ์ต่าง ๆ ได้ โดยทำให้สัญญาณที่ต้องการติดต่อสื่อสารเป็นสัญญาณดิจิตอลก็สามารถควบคุมการทำงานและแสดงผลที่คอมพิวเตอร์ได้

การแสดงผลในโครงงานนี้ยังนับว่าแสดงผลได้ช้า หากผู้ใช้ระบบต้องการให้มีความรวดเร็วขึ้น อาจต้องใช้การติดต่อระหว่างคอมพิวเตอร์และไมโครคอนโทรเลอร์ทางพอร์ทขนาน และควรเขียนโปรแกรมเดลไฟล์เป็นโปรแกรมเดียวกันสำหรับการควบคุมทั้งหมด เนื่องจากโครงงานนี้ใช้แต่ละโปรแกรมแต่ละโปรแกรมสำหรับแต่ละอุปกรณ์เครื่องเสียง หรือทำการลดจำนวนบอร์ด-ไมโครคอนโทรเลอร์ก็จะทำให้การเลือกการติดต่อจากคอมพิวเตอร์ลดลงจึงสามารถลดระยะเวลาระหว่างรอบการทำงานลงได้เช่นกัน อีกหนึ่งปัญหาคือส่วนอินเตอร์เฟส จากโครงงานนี้ใช้การเพิ่มวงจรทางดิจิตอลเพื่อไปควบคุมหน้าที่การทำงานต่าง ๆ จะเห็นว่าวงจรมีขนาดใหญ่และยุ่งยาก และ

ยังเกิดความผิดพลาดจากวงจรได้มากด้วย ซึ่งในปัจจุบันมีชิฟสำเร็จรูปสำหรับอุปกรณ์ต่าง ๆ เช่น ปรี-โทนคอนโทรล , อีควอไลเซอร์ ที่ควบคุมด้วยไมโครโปรเซสเซอร์ซึ่งจะทำให้สะดวกและประยุกต์ได้มากขึ้น

****

ภาคผนวก ก

SOFT WARE

1. โปรแกรมบน MCS-51

1.1 โปรแกรม TAPE

```
;*******************************************************
        INCL"BASE.ASM"          ;tape
;*******************************************************

        ORG     0000H

INITIAL: MOV    SP,#060H        ;set sp
        MOV     P1,#000H        ;clr p1
        MOV     DPTR,#2FF3H     ;port control#1
        MOV     A,#08BH         ;mode0 10001011b
        MOVX    @DPTR,A
        MOV     PCON,#080H      ;smod=1,/16
        MOV     TMOD,#022H      ;timer1 mode2
        MOV     TH1,#256-12     ;auto load
        MOV     TL1,#256-12     ;4800 baud
        SETB    TCON.6          ;start timer1
        MOV     SCON,#050H      ;8bit UART mode1
        MOV     R1,#068H        ;add control

MEM:    MOV     DPTR,#2FF1H     ;port display control
TREC:   MOV     R0,#070H        ;add 70h,record
        MOVX    A,@DPTR
        MOV     @R0,#006H
        JB      A.1,TPLAYL
        MOV     @R0,#018H
```

```
TPLAYL: MOV   R0,#071H        ;add 71h,play left
        MOV   @R0,#006H
        JB    A.2,TPLAYR
        MOV   @R0,#018H
TPLAYR: MOV   R0,#072H        ;add 72h,play right
        MOV   @R0,#006H
        JB    A.3,TPLAY
        MOV   @R0,#018H
TPLAY:  MOV   R0,#073H        ;add 73h,play
        MOV   @R0,#006H
        JB    A.4,TPAUSE
        MOV   @R0,#018H
TPAUSE: MOV   R0,#074H        ;add 74h,pause
        MOV   @R0,#006H
        JB    A.5,TRECMU
        MOV   @R0,#018H
TRECMU: MOV   R0,#075H        ;add 75h,recmute
        MOV   @R0,#006H
        JB    A.6,TLEVEL
        MOV   @R0,#018H
TLEVEL: MOV   DPTR,#2FF2H     ;port display level
        MOV   R0,#076H        ;add 76h,level
        MOVX  A,@DPTR
        ANL   A,#007H
LE0:    CJNE  A,#000H,LE1     ;level 0
        MOV   @R0,#07EH
LE1:    CJNE  A,#001H,LE2     ;level 1
        MOV   @R0,#078H
LE2:    CJNE  A,#002H,LE3     ;level 2
        MOV   @R0,#066H
```

```
LE3:    CJNE   A,#003H,LE4     ;level 3
        MOV    @R0,#060H
LE4:    CJNE   A,#004H,LE5     ;level 4
        MOV    @R0,#01EH
LE5:    CJNE   A,#005H,LE6     ;level 5
        MOV    @R0,#018H
LE6:    CJNE   A,#006H,LE7     ;level 6
        MOV    @R0,#006H
LE7:    CJNE   A,#007H,CODE    ;level 7
        MOV    @R0,#098H
CODE:   JNB    RI,MEMO         ;What is job,rx or tx
        CLR    RI
        MOV    A,SBUF
        CJNE   A,#066H,NEXT    ;if then rx else next
        AJMP   RX
NEXT:   CJNE   A,#078H,MEMO    ;if then tx else mem
        AJMP   TX
RX:     JNB    RI,RX           ;sub rx
        CLR    RI
        MOV    A,SBUF
        CJNE   A,#060H,MEMO    ;if then datarx else mem
DATARX:JNB     RI,DATARX
        CLR    RI
        MOV    A,SBUF
        SJMP   PLAYL
MEMO:   AJMP   MEM
PLAYL:  CJNE   A,#07EH,PLAYR   ;play left
        MOV    @R1,#010H
        AJMP   WORK
```

```
PLAYR:  CJNE    A,#080H,FF          ;play right
        MOV     @R1,#008
        AJMP    WORK
FF:     CJNE    A,#086H,REW         ;f/f
        MOV     @R1,#004H
        AJMP    WORK
REW:    CJNE    A,#098H,RECMUTE     ;rew
        MOV     @R1,#020H
        AJMP    WORK
RECMUTE:CJNE    A,#09EH,REC         ;recmute
        MOV     @R1,#001H
        AJMP    WORK
REC:    CJNE    A,#0E0H,PAUSE       ;rec
        MOV     @R1,#040H
        AJMP    WORK
PAUSE:  CJNE    A,#0E6H,STOP        ;pause.
        MOV     @R1,#002H
        AJMP    WORK
STOP:   CJNE    A,#0F8H,POWER       ;stop
        MOV     @R1,#080H
        AJMP    WORK
POWER:  CJNE    A,#0FEH,MEMO        ;power
        MOV     P1,#00FH
        MOV     R3,#0AAH
DEL1:   DJNZ    R3,DEL1
        MOV     P1,#000H
        AJMP    MEM
```

```
WORK:   MOV    DPTR,#2FF0H      ;sub work
        MOV    A,@R1
        MOVX   @DPTR,A
        MOV    R3,#0AAH
DELAY3: MOV    R2,#0AAH
DELAY2: DJNZ   R2,DELAY2
        DJNZ   R3,DELAY3
        MOV    A,#000H
        MOVX   @DPTR,A
RXEXIT: LJMP   MEM              ;end rx .
TX:     JNB    RI,TX            ;sub tx
        CLR    RI
        MOV    A,SBUF
        CJNE   A,#060H,TXEXIT ;if then ok else mem
        MOV    P1,#0F0H         ;enable tx
        MOV    R4,#007H         ;do 7 add
        MOV    R0,#070H         ;first add
TXLOOP: MOV    A,@R0
        CLR    TI
        MOV    SBUF,A
DATATX: JNB    TI,DATATX
        CLR    TI
        INC    R0               ;inc add
WAIT:   JNB    RI,WAIT          ;wait tx
        CLR    RI
        DJNZ   R4,TXLOOP        ;dec r4 go txloop
        MOV    P1,#000H         ;clr p1
TXEXIT: LJMP   MEM              ;to memory
        END                     ;ending
;*****************************************
```

1.2 PRE-AMPLIFIER

```
;*****************************************************************
            INCL"BASE.ASM"              ;pre-amplifier
;*****************************************************************

            ORG     0000H

INITIAL:    MOV     SP,#060H            ;set sp
            MOV     P1,#000H            ;clr p1
            MOV     DPTR,#2FF3H         ;port control#1
            MOV     A,#093H             ;mode0 10010011b
            MOVX    @DPTR,A
            MOV     PCON,#080H          ;smod=1,1/16
            MOV     TMOD,#022H          ;timer1 mode2
            MOV     TH1,#256-12         ;auto load
            MOV     TL1,#256-12         ;4800 baud
            SETB    TCON.6              ;start timer1
            MOV     SCON,#050H          ;8bit UART mode1
            MOV     R1,#068H            ;address control

MEM:        MOV     DPTR,#2FF0H         ;port treble,bass
            MOV     R0,#070H            ;address 70h,treble
            MOVX    A,@DPTR
            ANL     A,#00FH
TRE1:       CJNE    A,#000H,TRE2
            MOV     @R0,#006H
TRE2:       CJNE    A,#001H,TRE3
            MOV     @R0,#018H
TRE3:       CJNE    A,#002H,TRE4
            MOV     @R0,#01EH
```

```
TRE4:   CJNE   A,#003H,TRE5
        MOV    @R0,#060H
TRE5:   CJNE   A,#004H,TRE6
        MOV    @R0,#066H
TRE6:   CJNE   A,#005H,TRE7
        MOV    @R0,#078H
TRE7:   CJNE   A,#006H,TRE8
        MOV    @R0,#07EH
TRE8:   CJNE   A,#007H,TRE9
        MOV    @R0,#080H
TRE9:   CJNE   A,#008H,TRE10
        MOV    @R0,#086H
TRE10:  CJNE   A,#009H,TRE11
        MOV    @R0,#098H
TRE11:  CJNE   A,#00AH,TRE12
        MOV    @R0,#09EH
TRE12:  CJNE   A,#00BH,TRE13
        MOV    @R0,#0E0H
TRE13:  CJNE   A,#00CH,TRE14
        MOV    @R0,#0E6H
TRE14:  CJNE   A,#00DH,TRE15
        MOV    @R0,#0F8H
TRE15:  CJNE   A,#00EH,OTRE
        MOV    @R0,#0FEH
OTRE:   MOV    R0,#071H          ;address 71h,bass
        MOVX   A,@DPTR
        ANL    A,#0F0H

BASS1:  CJNE   A,#000H,BASS2
        MOV    @R0,#006H
```

```
BASS2:   CJNE   A,#010H,BASS3
         MOV    @R0,#018H
BASS3:   CJNE   A,#020H,BASS4
         MOV    @R0,#01EH
BASS4:   CJNE   A,#030H,BASS5
         MOV    @R0,#060H
BASS5:   CJNE   A,#040H,BASS6
         MOV    @R0,#066H
BASS6:   CJNE   A,#050H,BASS7
         MOV    @R0,#078H
BASS7:   CJNE   A,#060H,BASS8
         MOV    @R0,#07EH
BASS8:   CJNE   A,#070H,BASS9
         MOV    @R0,#080H
BASS9:   CJNE   A,#080H,BASS10
         MOV    @R0,#086H
BASS10:  CJNE   A,#090H,BASS11
         MOV    @R0,#098H
BASS11:  CJNE   A,#0A0H,BASS12
         MOV    @R0,#09EH
BASS12:  CJNE   A,#0B0H,BASS13
         MOV    @R0,#0E0H
BASS13:  CJNE   A,#0C0H,BASS14
         MOV    @R0,#0E6H
BASS14:  CJNE   A,#0D0H,BASS15
         MOV    @R0,#0F8H
BASS15:  CJNE   A,#0E0H,OBASS
         MOV    @R0,#0FEH
OBASS:   MOV    DPTR,#2FF1H     ;port volume,balance
         MOV    R0,#072H        ;address 72h,volume
```

```
        MOVX    A,@DPTR
        ANL     A,#00FH
VOL1:   CJNE    A,#000H,VOL2
        MOV     @R0,#006H
VOL2:   CJNE    A,#001H,VOL3
        MOV     @R0,#018H
VOL3:   CJNE    A,#002H,VOL4
        MOV     @R0,#01EH
VOL4:   CJNE    A,#003H,VOL5
        MOV     @R0,#060H
VOL5:   CJNE    A,#004H,VOL6
        MOV     @R0,#066H
VOL6:   CJNE    A,#005H,VOL7
        MOV     @R0,#078H
VOL7:   CJNE    A,#006H,VOL8
        MOV     @R0,#07EH
VOL8:   CJNE    A,#007H,VOL9
        MOV     @R0,#080H
VOL9:   CJNE    A,#008H,VOL10
        MOV     @R0,#086H
VOL10:  CJNE    A,#009H,VOL11
        MOV     @R0,#098H
VOL11:  CJNE    A,#00AH,VOL12
        MOV     @R0,#09EH
VOL12:  CJNE    A,#00BH,VOL13
        MOV     @R0,#0E0H
VOL13:  CJNE    A,#00CH,VOL14
        MOV     @R0,#0E6H
VOL14:  CJNE    A,#00DH,VOL15
        MOV     @R0,#0F8H
```

```
VOL15:  CJNE    A,#00EH,OVOL
        MOV     @R0,#0FEH
OVOL:   MOV     R0,#073H        ;address 73h,balance
        MOVX    A,@DPTR
        ANL     A,#0F0H
BAL1:   CJNE    A,#000H,BAL2
        MOV     @R0,#006H
BAL2:   CJNE    A,#010H,BAL3
        MOV     @R0,#018H
BAL3:   CJNE    A,#020H,BAL4
        MOV     @R0,#01EH
BAL4:   CJNE    A,#030H,BAL5
        MOV     @R0,#060H
BAL5:   CJNE    A,#040H,BAL6
        MOV     @R0,#066H
BAL6:   CJNE    A,#050H,BAL7
        MOV     @R0,#078H
BAL7:   CJNE    A,#060H,BAL8
        MOV     @R0,#07EH
BAL8:   CJNE    A,#070H,BAL9
        MOV     @R0,#080H
BAL9:   CJNE    A,#080H,BAL10
        MOV     @R0,#086H
BAL10:  CJNE    A,#090H,BAL11
        MOV     @R0,#098H
BAL11:  CJNE    A,#0A0H,BAL12
        MOV     @R0,#09EH

BAL12:  CJNE    A,#0B0H,BAL13
        MOV     @R0,#0E0H
```

```
BAL13:  CJNE    A,#0C0H,BAL14
        MOV     @R0,#0E6H
BAL14:  CJNE    A,#0D0H,BAL15
        MOV     @R0,#0F8H
BAL15:  CJNE    A,#0E0H,OBAL
        MOV     @R0,#0FEH
OBAL:   MOV     DPTR,#2FF2H     ;port on/off,select
        MOV     R0,#074H        ;address 74h,on/off
        MOVX    A,@DPTR
ONOFF:  MOV     @R0,#006H
        JB      A.0,INPUT
        MOV     @R0,#018H
INPUT:  MOV     R0,#075H        ;address 75h input
        MOVX    A,@DPTR
        ANL     A,#00EH
AUX:    CJNE    A,#000H,TUNER   ;aux
        MOV     @R0,#006H
TUNER:  CJNE    A,#002H,CD      ;tuner
        MOV     @R0,#018H
CD:     CJNE    A,#004H,TAPE    ;cd
        MOV     @R0,#01EH
TAPE:   CJNE    A,#008H,CODE    ;tape
        MOV     @R0,#060H
CODE:   JNB     RI,OTO          ;What is job,rx or tx
        CLR     RI
        MOV     A,SBUF
        CJNE    A,#066H,NEXT    ;if then rx else next
        AJMP    RX
NEXT:   CJNE    A,#078H,OTO     ;if then tx else mem
        AJMP    TX
```

```
RX:       JNB     RI,RX           ;sub rx
          CLR     RI
          MOV     A,SBUF
          CJNE    A,#01EH,OTO     ;if then datarx else mem
DATARX:   JNB     RI,DATARX
          CLR     RI
          MOV     A,SBUF
TO1:      CJNE    A,#006H,TO2     ;if then group1 else to2
          AJMP    GROUP1
TO2:      CJNE    A,#018H,OTO     ;if then group2 else mem
          AJMP    GROUP2
OTO:      AJMP    MEM

GROUP1:   JNB     RI,GROUP1       ;sub group1
          CLR     RI
          MOV     A,SBUF
BASUP:    CJNE    A,#07EH,BASDW   ;bass up
          MOV     @R1,#008H
          AJMP    WORK
BASDW:    CJNE    A,#080H,TREUP   ;bass down
          MOV     @R1,#004H
          AJMP    WORK
TREUP:    CJNE    A,#086H,TREDW   ;treble up
          MOV     @R1,#002H
          AJMP    WORK

TREDW:    CJNE    A,#098H,BALL    ;treble down
          MOV     @R1,#001H
          AJMP    WORK
```

```
BALL:    CJNE    A,#0E6H,BALR      ;balance left
         MOV     @R1,#040H
         AJMP    WORK
BALR:    CJNE    A,#0F8H,VOLUP     ;balance right
         MOV     @R1,#080H
         AJMP    WORK
VOLUP:   CJNE    A,#09EH,VOLDW     ;volume up
         MOV     @R1,#020H
         AJMP    WORK
VOLDW:   CJNE    A,#0E0H,RXEXIT    ;volume down
         MOV     @R1,#010H
         AJMP    WORK

GROUP2:  JNB     RI,GROUP2         ;sub group2
         CLR     RI
         MOV     A,SBUF
SELECT:  CJNE    A,#07EH,POWER     ;select
         MOV     @R1,#080H
         AJMP    OPER
POWER:   CJNE    A,#080H,RXEXIT    ;power
         MOV     @R1,#010H
         AJMP    OPER

WORK:    MOV     P1,@R1            ;sub work
         MOV     R4,#004H
DELAY3:  MOV     R3,#0AAH
DELAY2:  MOV     R2,#0AAH
DELAY1:  DJNZ    R2,DELAY1
         DJNZ    R3,DELAY2
         DJNZ    R4,DELAY3
```

```
        MOV     P1,#000H
        AJMP    MEM

OPER:   MOV     DPTR,#2FF2H     ;sub operated
        MOV     A,@R1
        MOVX    @DPTR,A
        MOV     R3,#0AAH
DELAY5: MOV     R2,#0AAH
DELAY4: DJNZ    R2,DELAY4
        DJNZ    R3,DELAY5
        MOV     A,#000H
        MOVX    @DPTR,A
RXEXIT: AJMP    MEM             ;end rx

TX:     JNB     RI,TX           ;sub tx
        CLR     RI
        MOV     A,SBUF
        CJNE    A,#01EH,TXEXIT  ;if then ok else mem

        MOV     DPTR,#2FF2H     ;port enable
        MOV     A,#020H         ;data enable
        MOVX    @DPTR,A         ;enable tx
        MOV     R4,#006H        ;do 6 address
        MOV     R0,#070H        ;frist address
TXLOOP: MOV     A,@R0
        CLR     TI
        MOV     SBUF,A
DATATX: JNB     TI,DATATX
        CLR     TI
        INC     R0              ;inc address
```

```
WAIT:   JNB     RI,WAIT         ;wait rx
        CLR     RI
        DJNZ    R4,TXLOOP       ;dec r4 go txloop
        MOV     A,#000H         ;level disable
        MOVX    @DPTR,A         ;disable tx
TXEXIT: LJMP    MEM             ;to memory

        END                     ;ending

;*****************************************************************
```

2. โปรแกรม DELPHI

2.1 โปรแกรม TAPE

```
program Tape1;

uses
    Forms,
    Tape in 'TAPE.PAS' {Form1};

{$R *.RES}

begin
    Application.CreateForm(TForm1, Form1);
    Application.Run;
end.
```

```
unit Tape;

interface

uses
  SysUtils, WinTypes, WinProcs, Messages,Classes, Graphics, Controls,Forms, Dialogs,
  ExtCtrls, StdCtrls, Buttons;

type
  TForm1 = class(TForm)
    BitBtn1: TBitBtn;BitBtn2: TBitBtn;BitBtn3: TBitBtn;
    BitBtn4: TBitBtn;BitBtn5: TBitBtn;BitBtn6: TBitBtn;
    BitBtn7: TBitBtn;BitBtn8: TBitBtn;BitBtn9: TBitBtn;
    Shape1: TShape;Shape2: TShape;Shape3: TShape;
    Shape4: TShape;Shape5: TShape;Shape6: TShape;
    Shape7: TShape;Shape8: TShape;Shape9: TShape;
    Shape10: TShape;Shape11: TShape;Shape12: TShape;
    Shape13: TShape;Shape14: TShape;Shape15: TShape;
    Shape16: TShape;Shape17: TShape;Shape18: TShape;
    Shape19: TShape;Shape20: TShape;Shape21: TShape;
    Shape22: TShape;Shape23: TShape;Shape24: TShape;
    Shape25: TShape;Shape26: TShape;Shape27: TShape;
    Shape28: TShape;Shape29: TShape;Shape30: TShape;
    Shape31: TShape;Shape32: TShape;Shape33: TShape;
    Shape34: TShape;Shape35: TShape;Shape36: TShape;
    Shape37: TShape;Shape38: TShape;Shape39: TShape;
    Shape40: TShape;Shape41: TShape;Shape42: TShape;
    Shape43: TShape;Shape44: TShape;Shape45: TShape;
    Shape46: TShape;Shape47: TShape;Shape48: TShape;
    Shape49: TShape;
    Label1: TLabel;Label2: TLabel;Label3: TLabel;
    Label4: TLabel;Label5: TLabel;Label6: TLabel;
    Label7: TLabel;
```

```
    Bevel1: TBevel;Bevel2: TBevel;Bevel3: TBevel;
    Timer1: TTimer;
    Shape50: TShape;Shape51: TShape;Shape52: TShape;
    procedure BitBtn1Click(Sender: TObject);
    procedure BitBtn2Click(Sender: TObject);
    procedure BitBtn3Click(Sender: TObject);
    procedure BitBtn4Click(Sender: TObject);
    procedure BitBtn5Click(Sender: TObject);
    procedure BitBtn6Click(Sender: TObject);
    procedure BitBtn7Click(Sender: TObject);
    procedure BitBtn8Click(Sender: TObject);
    procedure BitBtn9Click(Sender: TObject);
    procedure Timer1Timer(Sender: TObject);
  private { Private declarations }
  public { Public declarations }
  end;
var
  Form1: TForm1;
implementation {$R *.DFM}

procedure Delay(msec : LongInt);{DELAY}
var Time : LongInt;
begin
   Time := GetTickCount;
   repeat until GetTickCount > Time + msec;
end;
```

```
procedure TForm1.BitBtn1Click(Sender: TObject);{POWER}
begin
   Port[$2F8]:=$F5; Delay(1);Port[$2F8]:=$F4; Delay(1);Port[$2F8]:=$FF;
end;
procedure TForm1.BitBtn2Click(Sender: TObject);{REW}
begin
   Port[$2F8]:=$F5; Delay(1);Port[$2F8]:=$F4; Delay(1);Port[$2F8]:=$FA;
end;
procedure TForm1.BitBtn3Click(Sender: TObject);{PLAY REW}
begin
   Port[$2F8]:=$F5; Delay(1);Port[$2F8]:=$F4; Delay(1);Port[$2F8]:=$F7;
end;
procedure TForm1.BitBtn4Click(Sender: TObject);{F/F}
begin
   Port[$2F8]:=$F5; Delay(1);Port[$2F8]:=$F4; Delay(1);Port[$2F8]:=$F9;
end;
procedure TForm1.BitBtn5Click(Sender: TObject);{PLAY F/F}
begin
   Port[$2F8]:=$F5; Delay(1);Port[$2F8]:=$F4; Delay(1);Port[$2F8]:=$F8;
end;
procedure TForm1.BitBtn6Click(Sender: TObject);{REC MUTE}
begin
   Port[$2F8]:=$F5; Delay(1);Port[$2F8]:=$F4; Delay(1);Port[$2F8]:=$FB;
end;
procedure TForm1.BitBtn7Click(Sender: TObject);{RECORD}
begin
   Port[$2F8]:=$F5; Delay(1);Port[$2F8]:=$F4; Delay(1);Port[$2F8]:=$FC;
end;
```

```
procedure TForm1.BitBtn8Click(Sender: TObject);{PAUSE}
begin
  Port[$2F8]:=$F5; Delay(1);Port[$2F8]:=$F4; Delay(1);Port[$2F8]:=$FD;
end;

procedure TForm1.BitBtn9Click(Sender: TObject);{STOP}
begin
  Port[$2F8]:=$F5; Delay(1);Port[$2F8]:=$F4; Delay(1);Port[$2F8]:=$FE;
end;

procedure TForm1.Timer1Timer(Sender: TObject);{TX AUTO}
var BA,BB,BC,BD,BE,BF,BG,BH : Integer;
begin
  Port[$2F8] := $F6; Delay(1);
  Port[$2F8] := $F4; Delay(1);
  BA := Port[$2F8]; Port[$2F8] := SF0;Delay(1);{RECORD}
    if BA=241 then Shape20.Show else Shape20.Hide;
    if BA<>241 then Shape27.Show else Shape27.Hide;

  BB := Port[$2F8]; Port[$2F8] := SF0;Delay(1);{PLAY LEFT}
    if BB=241 then Shape23.Show else Shape23.Hide;
    if BB<>241 then Shape28.Show else Shape28.Hide;

  BC := Port[$2F8]; Port[$2F8] := $F0;Delay(1);{PLAY RIGHT}
    if BC=241 then Shape24.Show else Shape24.Hide;
    if BC<>241 then Shape29.Show else Shape29.Hide;
```

```
BD := Port[$2F8]; Port[$2F8] := SF0;Delay(1);{PLAY}
  if BD=241 then Shape25.Show else Shape25.Hide;
  if BD<>241 then Shape30.Show else Shape30.Hide;

BE := Port[$2F8]; Port[$2F8] := SF0;Delay(1);{PAUSE}
  if BE=241 then Shape22.Show else Shape22.Hide;
  if BE<>241 then Shape31.Show else Shape31.Hide;

BF := Port[$2F8]; Port[$2F8] := SF0;Delay(1);{REC MUTE}
  if BF=241 then Shape21.Show else Shape21.Hide;
  if BF<>241 then Shape32.Show else Shape32.Hide;

BG := Port[$2F8]; Port[$2F8] := SF0;Delay(1);{LEVEL}
  if BG=250 then
    begin
      Shape4.Show;Shape9.Show;Shape33.Hide;Shape41.Hide;{}
      Shape34.Show;Shape42.Show;Shape35.Show;Shape43.Show;
      Shape36.Show;Shape44.Show;Shape37.Show;Shape45.Show;
      Shape38.Show;Shape46.Show;Shape39.Show;Shape47.Show;
      Shape40.Show;Shape48.Show;
    end  else ;
  if BG=241 then
    begin
      Shape4.Show;Shape9.Show;Shape33.Hide;Shape41.Hide;
      Shape5.Show;Shape10.Show;Shape34.Hide;Shape42.Hide;{}
      Shape35.Show;Shape43.Show;Shape36.Show;Shape44.Show;
      Shape37.Show;Shape45.Show;Shape38.Show;Shape46.Show;
      Shape39.Show;Shape47.Show;Shape40.Show;Shape48.Show;
    end  else ;
```

```
if BG=242 then
  begin
    Shape4.Show;Shape9.Show;Shape33.Hide;Shape41.Hide;
    Shape5.Show;Shape10.Show;Shape34.Hide;Shape42.Hide;
    Shape6.Show;Shape11.Show;Shape35.Hide;Shape43.Hide;{}
    Shape36.Show;Shape44.Show;Shape37.Show;Shape45.Show;
    Shape38.Show;Shape46.Show;Shape39.Show;Shape47.Show;
    Shape40.Show;Shape48.Show;
  end  else ;
if BG=243 then
  begin
    Shape4.Show;Shape9.Show;Shape33.Hide;Shape41.Hide;
    Shape5.Show;Shape10.Show;Shape34.Hide;Shape42.Hide;
    Shape6.Show;Shape11.Show;Shape35.Hide;Shape43.Hide;
    Shape7.Show;Shape12.Show;Shape36.Hide;Shape44.Hide;{}
    Shape37.Show;Shape45.Show;Shape38.Show;Shape46.Show;
    Shape39.Show;Shape47.Show;Shape40.Show;Shape48.Show;
  end  else ;
if BG=244 then
  begin
    Shape4.Show;Shape9.Show;Shape33.Hide;Shape41.Hide;
    Shape5.Show;Shape10.Show;Shape34.Hide;Shape42.Hide;
    Shape6.Show;Shape11.Show;Shape35.Hide;Shape43.Hide;
    Shape7.Show;Shape12.Show;Shape36.Hide;Shape44.Hide;
    Shape8.Show;Shape13.Show;Shape37.Hide;Shape45.Hide;{}
    Shape38.Show;Shape46.Show;Shape39.Show;Shape47.Show;
    Shape40.Show;Shape48.Show;
  end  else ;
if BG=245 then
```

```
    begin
      Shape4.Show;Shape9.Show;Shape33.Hide;Shape41.Hide;
      Shape5.Show;Shape10.Show;Shape34.Hide;Shape42.Hide;
      Shape6.Show;Shape11.Show;Shape35.Hide;Shape43.Hide;
      Shape7.Show;Shape12.Show;Shape36.Hide;Shape44.Hide;
      Shape8.Show;Shape13.Show;Shape37.Hide;Shape45.Hide;
      Shape14.Show;Shape17.Show;Shape38.Hide;Shape46.Hide;{}
      Shape39.Show;Shape47.Show;Shape40.Show;Shape48.Show;
    end  else ;
  if BG=246 then
    begin
      Shape4.Show;Shape9.Show;Shape33.Hide;Shape41.Hide;
      Shape5.Show;Shape10.Show;Shape34.Hide;Shape42.Hide;
      Shape6.Show;Shape11.Show;Shape35.Hide;Shape43.Hide;
      Shape7.Show;Shape12.Show;Shape36.Hide;Shape44.Hide;
      Shape8.Show;Shape13.Show;Shape37.Hide;Shape45.Hide;
      Shape14.Show;Shape17.Show;Shape38.Hide;Shape46.Hide;
      Shape15.Show;Shape18.Show;Shape39.Hide;Shape47.Hide;{}
      Shape40.Show;Shape48.Show;
    end  else ;
  if BG=247 then
    begin
      Shape4.Show;Shape9.Show;Shape33.Hide;Shape41.Hide;
      Shape5.Show;Shape10.Show;Shape34.Hide;Shape42.Hide;
      Shape6.Show;Shape11.Show;Shape35.Hide;Shape43.Hide;
      Shape7.Show;Shape12.Show;Shape36.Hide;Shape44.Hide;
      Shape8.Show;Shape13.Show;Shape37.Hide;Shape45.Hide;
      Shape14.Show;Shape17.Show;Shape38.Hide;Shape46.Hide;
      Shape15.Show;Shape18.Show;Shape39.Hide;Shape47.Hide;
```

```
        Shape16.Show;Shape19.Show;Shape40.Hide;Shape48.Hide;
      end  else ;

  BH := Port[$2F8]; Port[$2F8] := $F0;Delay(1);{USER TAPE}
    if BH=241 then
      begin
        Shape3.Show;Shape26.Show;Shape49.Show;Shape50.Show;
      end
    else;
      begin
        Shape3.Hide;Shape26.Hide;Shape49.Hide;Shape50.Hide;
      end;
    if BB=BC then
      begin
        Shape20.Hide;Shape21.Hide;Shape22.Hide;Shape23.Hide;
        Shape24.Hide;Shape25.Hide;Shape4.Hide;Shape9.Hide;
        Shape27.Show;Shape28.Show;Shape29.Show;Shape30.Show;
        Shape31.Show;Shape32.Show;Shape33.Show;Shape41.Show;
      end else;
end;

end.
```

2.2 โปรแกรม PRE-AMPLIFIER

```
program Amp1;

uses
  Forms,
  Amp in 'AMP.PAS' {Form1};

{$R *.RES}

begin
  Application.CreateForm(TForm1, Form1);
  Application.Run;
end.
```

```
unit Amp;

interface

uses

  SysUtils, WinTypes, WinProcs, Messages,Classes, Graphics, Controls,Forms,

  Dialogs, StdCtrls, ExtCtrls;

type

  TForm1 = class(TForm)

    Button1: TButton;Button2: TButton;Button3: TButton;

    Button4: TButton;Button5: TButton;Button6: TButton;

    Button7: TButton;Button8: TButton;Button9: TButton;

    Button10: TButton;

    Shape1: TShape;Shape2: TShape;Shape3: TShape;

    Shape4: TShape;Shape5: TShape;Shape6: TShape;

    Shape7: TShape;Shape8: TShape;Shape9: TShape;

    Shape10: TShape;Shape11: TShape;Shape12: TShape;

    Shape13: TShape;Shape14: TShape;Shape15: TShape;

    Shape16: TShape;Shape17: TShape;Shape18: TShape;

    Shape19: TShape;Shape20: TShape;Shape21: TShape;

    Shape22: TShape;Shape23: TShape;Shape24: TShape;

    Shape25: TShape;Shape26: TShape;Shape27: TShape;

    Shape28: TShape;Shape29: TShape;Shape30: TShape;

    Shape31: TShape;Shape32: TShape;Shape33: TShape;

    Shape34: TShape;Shape35: TShape;Shape36: TShape;

    Shape37: TShape;Shape38: TShape;Shape39: TShape;

    Shape40: TShape;Shape41: TShape;Shape42: TShape;

    Shape43: TShape;Shape44: TShape;Shape45: TShape;

    Shape46: TShape;Shape47: TShape;Shape48: TShape;

    Shape49: TShape;Shape50: TShape;Shape51: TShape;

    Shape52: TShape;Shape53: TShape;Shape54: TShape;

    Shape55: TShape;Shape56: TShape;Shape57: TShape;
```

Shape58: TShape;Shape59: TShape;Shape60: TShape;

Shape61: TShape;Shape62: TShape;Shape63: TShape;

Shape64: TShape;Shape65: TShape;Shape66: TShape;

Shape67: TShape;Shape68: TShape;Shape69: TShape;

Shape70: TShape;Shape71: TShape;Shape72: TShape;

Shape73: TShape;Shape74: TShape;Shape75: TShape;

Shape76: TShape;Shape77: TShape;Shape78: TShape;

Shape79: TShape;Shape80: TShape;Shape81: TShape;

Shape82: TShape;Shape83: TShape;Shape84: TShape;

Shape85: TShape;Shape86: TShape;Shape87: TShape;

Shape88: TShape;Shape89: TShape;Shape90: TShape;

Shape91: TShape;Shape92: TShape;Shape93: TShape;

Shape94: TShape;Shape95: TShape;Shape96: TShape;

Shape97: TShape;Shape98: TShape;Shape99: TShape;

Shape100: TShape;Shape101: TShape;Shape102: TShape;

Shape103: TShape;Shape104: TShape;Shape105: TShape;

Shape106: TShape;Shape107: TShape;Shape108: TShape;

Shape109: TShape;Shape110: TShape;Shape111: TShape;

Shape112: TShape;Shape113: TShape;Shape114: TShape;

Shape115: TShape;Shape116: TShape;Shape117: TShape;

Shape118: TShape;Shape119: TShape;Shape120: TShape;

Shape121: TShape;Shape122: TShape;Shape123: TShape;

Shape124: TShape;Shape125: TShape;Shape126: TShape;

Shape127: TShape;Shape128: TShape;Shape129: TShape;

Shape130: TShape;

Label1: TLabel;Label2: TLabel;Label3: TLabel;

Label4: TLabel;Label5: TLabel;Label6: TLabel;

Label7: TLabel;Label8: TLabel;Label9: TLabel;

Bevel1: TBevel;Bevel2: TBevel;Bevel3: TBevel;

Bevel4: TBevel;Bevel5: TBevel;Bevel6: TBevel;

```
    Timer1: TTimer;
    procedure Button1Click(Sender: TObject);
    procedure Button2Click(Sender: TObject);
    procedure Button3Click(Sender: TObject);
    procedure Button4Click(Sender: TObject);
    procedure Button5Click(Sender: TObject);
    procedure Button6Click(Sender: TObject);
    procedure Button7Click(Sender: TObject);
    procedure Button8Click(Sender: TObject);
    procedure Button9Click(Sender: TObject);
    procedure Button10Click(Sender: TObject);
    procedure Timer1Timer(Sender: TObject);
  private { Private declarations }
  public { Public declarations }
  end;
var
  Form1: TForm1;
implementation {$R *.DFM}

procedure Delay(msec : LongInt);{DELAY}
var Time : LongInt;
begin
  Time := GetTickCount;
  repeat until GetTickCount > Time + msec;
end;
```

```
procedure TForm1.Button1Click(Sender: TObject);{POWER}
begin
  Port[$2F8]:=$F5; Delay(1);Port[$2F8]:=$F3; Delay(1);
  Port[$2F8]:=$F2; Delay(1);Port[$2F8]:=$F8;
end;
procedure TForm1.Button2Click(Sender: TObject);{BASS DOWN}
begin
  Port[$2F8]:=$F5; Delay(1);Port[$2F8]:=$F3; Delay(1);
  Port[$2F8]:=$F1; Delay(1);Port[$2F8]:=$F8;
end;
procedure TForm1.Button3Click(Sender: TObject);{BASS UP}
begin
  Port[$2F8]:=$F5; Delay(1);Port[$2F8]:=$F3; Delay(1);
  Port[$2F8]:=$F1; Delay(1);Port[$2F8]:=$F7;
end;
procedure TForm1.Button4Click(Sender: TObject);{BALANCE LEFT}
begin
  Port[$2F8]:=$F5; Delay(1);Port[$2F8]:=$F3; Delay(1);
  Port[$2F8]:=$F1; Delay(1);Port[$2F8]:=$FD;
end;
procedure TForm1.Button5Click(Sender: TObject);{BALANCE RIGHT}
begin
  Port[$2F8]:=$F5; Delay(1);Port[$2F8]:=$F3; Delay(1);
  Port[$2F8]:=$F1; Delay(1);Port[$2F8]:=$FE;
end;
```

```
procedure TForm1.Button6Click(Sender: TObject);{TREBLE DOWN}
begin
  Port[$2F8]:=$F5; Delay(1);Port[$2F8]:=$F3; Delay(1);
  Port[$2F8]:=$F1; Delay(1);Port[$2F8]:=$FA;
end;
procedure TForm1.Button7Click(Sender: TObject);{TREBLE UP}
begin
  Port[$2F8]:=$F5; Delay(1);Port[$2F8]:=$F3; Delay(1);
  Port[$2F8]:=$F1; Delay(1);Port[$2F8]:=$F9;
end;
procedure TForm1.Button8Click(Sender: TObject);{VOLUME DOWN}
begin
  Port[$2F8]:=$F5; Delay(1);Port[$2F8]:=$F3; Delay(1);
  Port[$2F8]:=$F1; Delay(1);Port[$2F8]:=$FC;
end;
procedure TForm1.Button9Click(Sender: TObject);{VOLUME UP}
begin
  Port[$2F8]:=$F5; Delay(1);Port[$2F8]:=$F3; Delay(1);
  Port[$2F8]:=$F1; Delay(1);Port[$2F8]:=$FB;
end;
procedure TForm1.Button10Click(Sender: TObject);{SELECT}
begin
  Port[$2F8]:=$F5; Delay(1);Port[$2F8]:=$F3; Delay(1);
  Port[$2F8]:=$F2; Delay(1);Port[$2F8]:=$F7;
end;
```

```
procedure TForm1.Timer1Timer(Sender: TObject);{TX AUTO}
var DA,DB,DC,DD,DE,DF : Integer;
begin
  Port[$2F8] := $F6; Delay(1);
  Port[$2F8] := $F3; Delay(1);
  DA := Port[$2F8]; Port[$2F8] := SF0;Delay(1);{TREBEL}
    if DA=241  then Shape16.Show else Shape16.Hide;
    if DA<>241 then Shape71.Show else Shape71.Hide;
    if DA=242  then Shape17.Show else Shape17.Hide;
    if DA<>242 then Shape72.Show else Shape72.Hide;
    if DA=243  then Shape18.Show else Shape18.Hide;
    if DA<>243 then Shape73.Show else Shape73.Hide;
    if DA=244  then Shape19.Show else Shape19.Hide;
    if DA<>244 then Shape74.Show else Shape74.Hide;
    if DA=245  then Shape20.Show else Shape20.Hide;
    if DA<>245 then Shape75.Show else Shape75.Hide;
    if DA=246  then Shape21.Show else Shape21.Hide;
    if DA<>246 then Shape76.Show else Shape76.Hide;
    if DA=247  then Shape22.Show else Shape22.Hide;
    if DA<>247 then Shape77.Show else Shape77.Hide;
    if DA=248  then Shape23.Show else Shape23.Hide;
    if DA<>248.then Shape78.Show else Shape78.Hide;
    if DA=249  then Shape24.Show else Shape24.Hide;
    if DA<>249 then Shape79.Show else Shape79.Hide;
    if DA=250  then Shape25.Show else Shape25.Hide;
    if DA<>250 then Shape80.Show else Shape80.Hide;
    if DA=251  then Shape26.Show else Shape26.Hide;
    if DA<>251 then Shape81.Show else Shape81.Hide;
```

```
  if DA=252  then Shape27.Show else Shape27.Hide;
  if DA<>252 then Shape82.Show else Shape82.Hide;
  if DA=253  then Shape28.Show else Shape28.Hide;
  if DA<>253 then Shape83.Show else Shape83.Hide;
  if DA=254  then Shape29.Show else Shape29.Hide;
  if DA<>254 then Shape84.Show else Shape84.Hide;
  if DA=255  then Shape30.Show else Shape30.Hide;
  if DA<>255 then Shape85.Show else Shape85.Hide;

DB := Port[$2F8]; Port[$2F8] := $F0;Delay(1);{BASS}
  if DB=241  then Shape1.Show else Shape1.Hide;
  if DB<>241 then Shape86.Show else Shape86.Hide;
  if DB=242  then Shape2.Show else Shape2.Hide;
  if DB<>242 then Shape87.Show else Shape87.Hide;
  if DB=243  then Shape3.Show else Shape3.Hide;
  if DB<>243 then Shape88.Show else Shape88.Hide;
  if DB=244  then Shape4.Show else Shape4.Hide;
  if DB<>244 then Shape89.Show else Shape89.Hide;
  if DB=245  then Shape5.Show else Shape5.Hide;
  if DB<>245 then Shape90.Show else Shape90.Hide;
  if DB=246  then Shape6.Show else Shape6.Hide;
  if DB<>246 then Shape91.Show else Shape91.Hide;
  if DB=247  then Shape7.Show else Shape7.Hide;
  if DB<>247 then Shape92.Show else Shape92.Hide;
  if DB=248  then Shape8.Show else Shape8.Hide;
  if DB<>248 then Shape93.Show else Shape93.Hide;
  if DB=249  then Shape9.Show else Shape9.Hide;
  if DB<>249 then Shape94.Show else Shape94.Hide;
```

```
    if DB=250  then Shape10.Show else Shape10.Hide;
    if DB<>250 then Shape95.Show else Shape95.Hide;
    if DB=251  then Shape11.Show else Shape11.Hide;
    if DB<>251 then Shape96.Show else Shape96.Hide;
    if DB=252  then Shape12.Show else Shape12.Hide;
    if DB<>252 then Shape97.Show else Shape97.Hide;
    if DB=253  then Shape13.Show else Shape13.Hide;
    if DB<>253 then Shape98.Show else Shape98.Hide;
    if DB=254  then Shape14.Show else Shape14.Hide;
    if DB<>254 then Shape99.Show else Shape99.Hide;
    if DB=255  then Shape15.Show else Shape15.Hide;
    if DB<>255 then Shape100.Show else Shape100.Hide;

DC := Port[$2F8]; Port[$2F8] := $F0;Delay(1);{VOLUME}
    if DC=241  then Shape46.Show else Shape46.Hide;
    if DC<>241 then Shape101.Show else Shape101.Hide;
    if DC=242  then Shape47.Show else Shape47.Hide;
    if DC<>242 then Shape102.Show else Shape102.Hide;
    if DC=243  then Shape48.Show else Shape48.Hide;
    if DC<>243 then Shape103.Show else Shape103.Hide;
    if DC=244  then Shape49.Show else Shape49.Hide;
    if DC<>244 then Shape104.Show else Shape104.Hide;
    if DC=245  then Shape50.Show else Shape50.Hide;
    if DC<>245 then Shape105.Show else Shape105.Hide;
    if DC=246  then Shape51.Show else Shape51.Hide;
    if DC<>246 then Shape106.Show else Shape106.Hide;
    if DC=247  then Shape52.Show else Shape52.Hide;
    if DC<>247 then Shape107.Show else Shape107.Hide;
```

```
if DC=248  then Shape53.Show else Shape53.Hide;
if DC<>248 then Shape108.Show else Shape108.Hide;
if DC=249  then Shape54.Show else Shape54.Hide;
if DC<>249 then Shape109.Show else Shape109.Hide;
if DC=250  then Shape55.Show else Shape55.Hide;
if DC<>250 then Shape110.Show else Shape110.Hide;
if DC=251  then Shape56.Show else Shape56.Hide;
if DC<>251 then Shape111.Show else Shape111.Hide;
if DC=252  then Shape57.Show else Shape57.Hide;
if DC<>252 then Shape112.Show else Shape112.Hide;
if DC=253  then Shape58.Show else Shape58.Hide;
if DC<>253 then Shape113.Show else Shape113.Hide;
if DC=254  then Shape59.Show else Shape59.Hide;
if DC<>254 then Shape114.Show else Shape114.Hide;
if DC=255  then Shape60.Show else Shape60.Hide;
if DC<>255 then Shape115.Show else Shape115.Hide;

DD := Port[$2F8]; Port[$2F8] := SF0;Delay(1);{BALANCE}
if DD=241  then Shape31.Show else Shape31.Hide;
if DD<>241 then Shape116.Show else Shape116.Hide;
if DD=242  then Shape32.Show else Shape32.Hide;
if DD<>242 then Shape117.Show else Shape117.Hide;
if DD=243  then Shape33.Show else Shape33.Hide;
if DD<>243 then Shape118.Show else Shape118.Hide;
if DD=244  then Shape34.Show else Shape34.Hide;
if DD<>244 then Shape119.Show else Shape119.Hide;
if DD=245  then Shape35.Show else Shape36.Hide;
if DD<>245 then Shape120.Show else Shape120.Hide;
```

```
if DD=246  then Shape36.Show else Shape36.Hide;
if DD<>246 then Shape121.Show else Shape121.Hide;
if DD=247  then Shape37.Show else Shape37.Hide;
if DD<>247 then Shape122.Show else Shape122.Hide;
if DD=248  then Shape38.Show else Shape38.Hide;
if DD<>248 then Shape123.Show else Shape123.Hide;
if DD=249  then Shape39.Show else Shape39.Hide;
if DD<>249 then Shape124.Show else Shape124.Hide;
if DD=250  then Shape40.Show else Shape40.Hide;
if DD<>250 then Shape125.Show else Shape125.Hide;
if DD=251  then Shape41.Show else Shape41.Hide;
if DD<>251 then Shape126.Show else Shape126.Hide;
if DD=252  then Shape42.Show else Shape42.Hide;
if DD<>252 then Shape127.Show else Shape127.Hide;
if DD=253  then Shape43.Show else Shape43.Hide;
if DD<>253 then Shape128.Show else Shape128.Hide;
if DD=254  then Shape44.Show else Shape44.Hide;
if DD<>254 then Shape129.Show else Shape129.Hide;
if DD=255  then Shape45.Show else Shape45.Hide;
if DD<>255 then Shape130.Show else Shape130.Hide;

DE := Port[$2F8]; Port[$2F8] := SF0;Delay(1);{POWER}
if DE=241  then Shape65.Show else Shape65.Hide;
if DE<>241 then Shape66.Show else Shape66.Hide;

DF := Port[$2F8]; Port[$2F8] := $F0;Delay(1);{SELECT}
if DF=244  then Shape61.Show else Shape61.Hide;
if DF<>244 then Shape67.Show else Shape67.Hide;
if DF=243  then Shape62.Show else Shape62.Hide;
```

```
    if DF<>243 then Shape68.Show else Shape68.Hide;
    if DF=242  then Shape63.Show else Shape63.Hide;
    if DF<>242 then Shape69.Show else Shape69.Hide;
    if DF=241  then Shape64.Show else Shape64.Hide;
    if DF<>241 then Shape70.Show else Shape70.Hide;
    if DE<>241 then
      begin
        Shape64.Hide;Shape70.Show;Shape1.Hide;Shape86.Show;
        Shape16.Hide;Shape71.Show;Shape31.Hide;Shape116.Show;
        Shape46.Hide;Shape101.Show;
      end  else;
end;

end.
```

ภาคผนวก ข

1. โปรแกรมทดสอบบน MCS-51

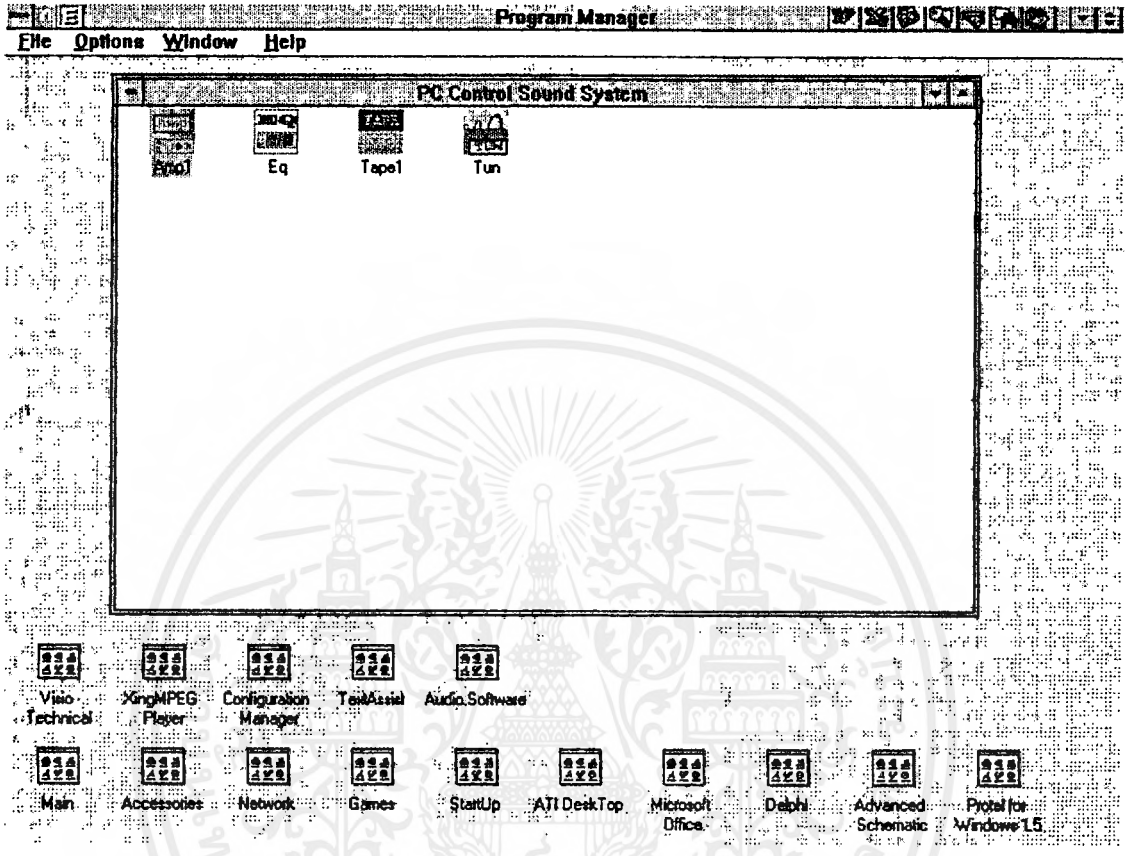
```
;************************************************
        INCL"BASE.ASM"          ;data test
;************************************************

        ORG     0000H

INITIAL: MOV    DPTR,#2FF3H     ;port control#1
        MOV     A,#080H         ;mode1 10000000b
        MOVX    @DPTR,A
        MOV     DPTR,#4FF3H     ;port control#2
        MOV     A,#080H         ;mode1 10000000b
        MOVX    @DPTR,A
        MOV     PCON,#080H      ;smod=1,1/16
        MOV     TMOD,#022H      ;timer1 mode2
        MOV     TH1,#256-12     ;auto load
        MOV     TL1,#256-12     ;4800 baud
        SETB    TCON.6          ;start timer1
        MOV     SCON,#050H      ;8bit UART mode1
        MOV     DPTR,#2FF0H

RX1:    JNB     RI,RX1          ;byte after
        CLR     RI
        MOV     A,SBUF
        MOV     DPTR,#2FF0H     ;port A
        MOVX    @DPTR,A
RX2:    JNB     RI,RX2          ;byte before
        CLR     RI
```

```
        MOV     A,SBUF
        MOV     DPTR,#2FF1H       ;port B
        MOVX    @DPTR,A

        END                       ;ending

;*************************************************
```

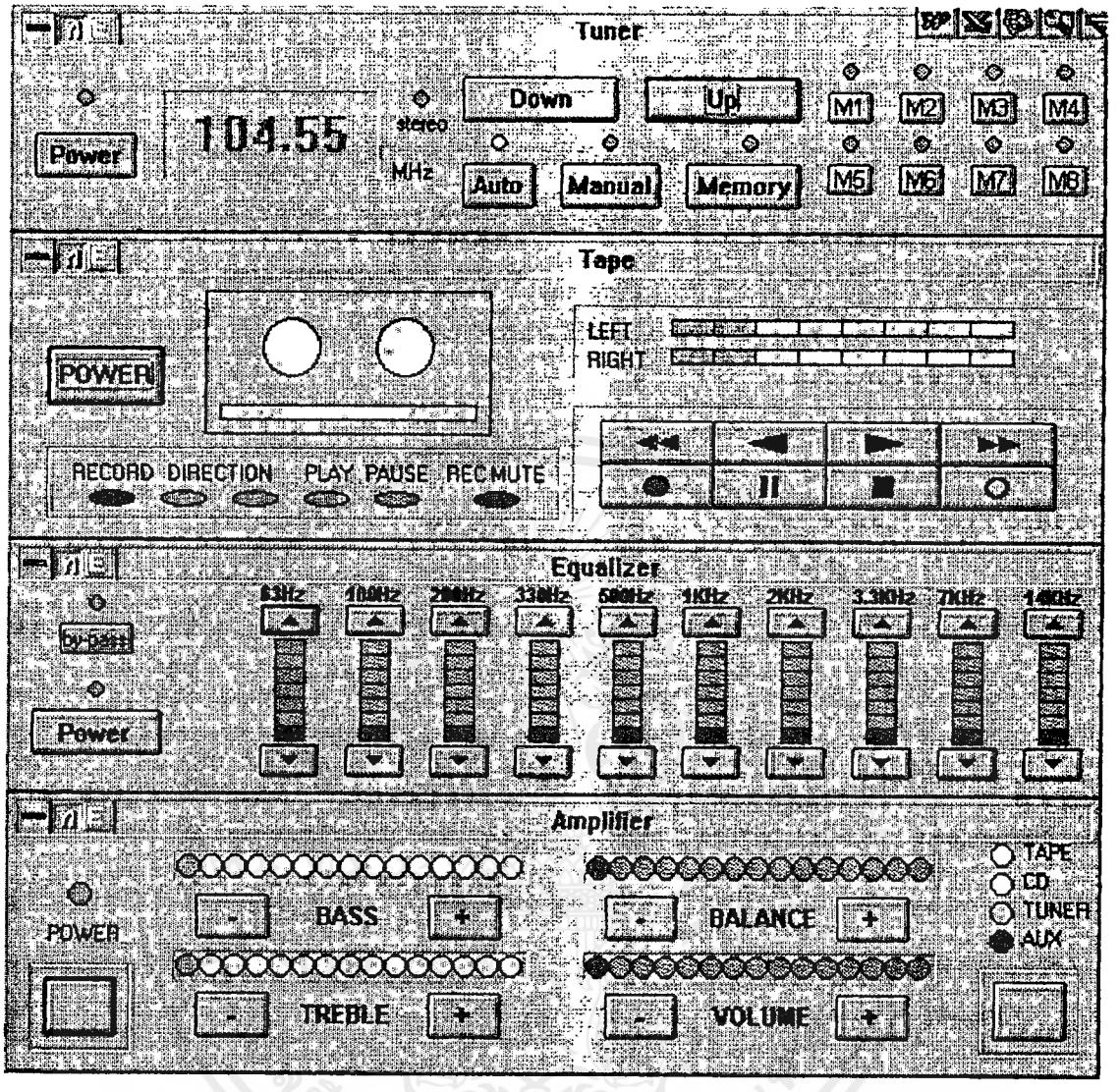
2. โปรแกรมทดสอบบน Delphi

```
unit Tast;
interface
uses
  SysUtils, WinTypes, WinProcs, Messages,
  Classes, Graphics, Controls, Forms, Dialogs, StdCtrls;
type
  TForm1 = class(TForm)
    Button1: TButton;
    procedure Button1Click(Sender: TObject);
  private { private declarations }
  public { Public declarations }
  end;
var
  Form1: TForm1;
implementation {$R *.DFM}

procedure TForm1.Button1Click(Sender: TObject);
begin
   Port[$2F8]:=$XX; {XX=00H-FFH}
end;

end.
```

ภาคผนวก ค



รูปที่ 1 แสดง ICON ของโปรแกรม PC Control Sound System



รูปที่ 2 แสดง หน้าจอบนคอมพิวเตอร์ขณะใช้งาน

หนังสืออ้างอิง

1. บุญเลิศ เอี่ยมทัศนา , " DELPHI " , เริ่มเขียนโปรแกรมบนวินโดวส์ด้วย DELPHI 1.0 และ 2.0.
2. นุกูลกระจาย , " บอร์แลนด์ปาสคาล " , การเขียนแอปพลิเคชั่นในวิรดดวส์ด้วยบอร์แลนด์-ปาสคาล.
3. นุกูลกระจาย , " บอร์แลนด์ปาสคาล 7.0 " , การเขียนโปรแกรมด้วยบอร์แลนด์ปาสคาล 7.0.
4. บริษัท ซีเอ็ดยูเคชั่น ," MCS-51 (80C535) " , โมโครโปรเซสเซอร์ 2.
5. สุเจตน์ จันทรังษ์ , " MCS-51 " , มหานครวิทยาลัย , ไมโครคอนโทรลเลอร์ชิพเดี่ยว 8051.
6. บริษัท ซีเอ็ดยูเคชั่น , " การสื่อสารข้อมูลพอร์ทอนุกรม" , คัมภีการใช้งานการสื่อสาร-อนุกรมบน PC.
7. นุกูล กระจาย , " ภาษาปาสคาล " , การเขียนโปรแกรมและประมวลผลข้อมูลด้วย-เทอร์โบปาสคาล.
8. พ.ต.ประพัฒน์ อุทโยกาศ , " เทอร์โบปาสคาล " , เรียนเทอร์โบปาสคาลด้วยตนเอง.
9. สุนทร วิทูสุรพจน์ , " คอนโทรลเลอร์ 8051 " , การโปรแกรมภาษาแอสเซมบลีของไมโคร-คอนโทรลเลอร์ตระกูล 8051 .
10. รัชนัย อินทุใส , ไตรภพ อินทุใส , " MCS-51 " , ไมโครคอนโทรลเลอร์ 8051 .
11. วิริยชัย วิไลเทเวศร์ ," ยูนเนอร์เอฟเอ็มซินธีไซเซอร์ " , วารสารเซมิคอนดัคเตอร์-อิเล็กทรอนิกส์ , ฉบับที่ 124 , 2536 , หน้า14-25.
12. วิทยา เปี่ยมเจริญ , " ดีซีโทนคอนโทรลกับสเตริโอไวด์ " , รวมโครงาน 90 (CEW) , หน้า 55-58.