ปีการศึกษา 2531

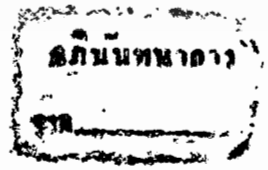
การใช้งานการสื่อสารข้อมูลแบบอนุกรมกับเครื่องดนตรี

โดย

นาย บดินทร์ ซื่อตรง

อาจารย์ที่ปรึกษา

อ. สมยศ จุณณะปิยะ



การใช้งานการสื่อสารข้อมูลแบบอนุกรมกับเครื่องดนตรี

MUSICAL INSTRUMENT DIGITAL INTERFACE

โดย นายบดินทร์ ซื่อตรง รหัสประจำตัว 28.1124

อาจารย์ที่ปรึกษา

อ. สมยศ จุณณะปิยะ

วิทยานิพนธ์สำหรับปริญญาวิศวกรรมศาสตร์บัณฑิต

สาขาวิชาวิศวกรรมโทรคมนาคม

คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้า เจ้าคุณทหาร ลาดกระบัง

ปีการศึกษา 2531

ปริญญานิพนธ์ปีการศึกษา 2531

ภาควิชา วิศวกรรมโทรคมนาคม

คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้า เจ้าคุณทหาร ลาดกระบัง

เรื่อง การใช้งานการสื่อสารข้อมูลแบบอนุกรมกับเครื่องดนตรี

ผู้จัดทำ นาย บดินทร์ ซื่อตรง รหัสประจำตัว 28.1124

.....อาจารย์ที่ปรึกษา

(อาจารย์ สมยศ จุณณะปิยะ)

การใช้งานการสื่อสารข้อมูลแบบอนุกรมกับเครื่องดนตรี

บดินทร์ ซื่อตรง

อ.สมยศ จุณณะปิยะ อาจารย์ที่ปรึกษา

บทคัดย่อ

ปริญญานิพนธ์ฉบับนี้ เป็นเรื่องเกี่ยวกับการประยุกต์ใช้งานการสื่อสารข้อมูลแบบอนุกรมกับเครื่องดนตรี (Musical Instrument Digital Interface :MIDI) โดยจะใช้คอมพิวเตอร์ไอบีเอ็มพีซีเป็นมีดี้ซีเควนเซอร์ (MIDI Sequencer) ผู้ใช้สามารถป้อนโน๊ตเพลง และข้อมูลในการเล่น เช่น ความดัง,โปรแกรมเสียง ฯลฯ ของเครื่องดนตรีแต่ละเครื่องที่นำมาพ่วงกับคอมพิวเตอร์ในระบบมีดี้ได้ถึง 16 แชนแนล จากนั้นคอมพิวเตอร์จะเปลี่ยนข้อมูลเหล่านี้ให้เป็นรหัสมีดี้แล้วส่งให้เครื่องดนตรีเพื่อสั่งให้เครื่องดนตรีทำงานตามที่ผู้ใช้ได้โปรแกรมไว้ ทั้งนี้สามารถควบคุมเครื่องดนตรีได้ 16 แชนแนลเช่นกัน

MUSICAL INSTRUMENT DIGITAL INTERFACE

Bordin Suetrong

Mr. Somyot Junnapiya Advisor

Abstract

This thesis is an application of Musical Instrument Digital Interface (MIDI), the datacommunication for musical instruments that anables synthesizers, sequencers, computers, rhythm machine, etc. to be interconnected through a standard interface.

This thesis uses the IBM PC-XT computer to be a MIDI sequencer. Users can program musical notes and any performances up to 16 channels. To control musical instruments such as synthesizers, computer will send these datas in MIDI format to the instruments so, the instruments can operate as users's edit.

สารบัญ

บทที่ 1

บทนำ

ในปริญญานิพนธ์นี้จะกล่าวถึงเนื้อหาในบทต่างๆดังนี้

บทที่ 2 หลักการ และ ข้อกำหนดของระบบมีดี้

จะกล่าวถึงหลักการ รวมทั้งข้อกำหนดต่างๆทั้งทางฮาร์แวร์และซอฟแวร์ ประเภทและความหมายของข้อมูลที่สื่อสารกันในระบบมีดี้

บทที่ 3 การสร้าง

จะกล่าวถึงมีดี้ซีเควนเซอร์ (MIDI sequencer) การทำงาน และ การสร้าง

บทที่ 4 การทดลองและผลการทดลอง

เป็นการทดลองและผลที่ได้จากการทดลองสร้างในการศึกษาโครงงาน

บทที่ 5 บทวิจารณ์และสรุป

สรุปเนื้อหาของโครงงาน และ ข้อเสนอแนะ

ภาคผนวก

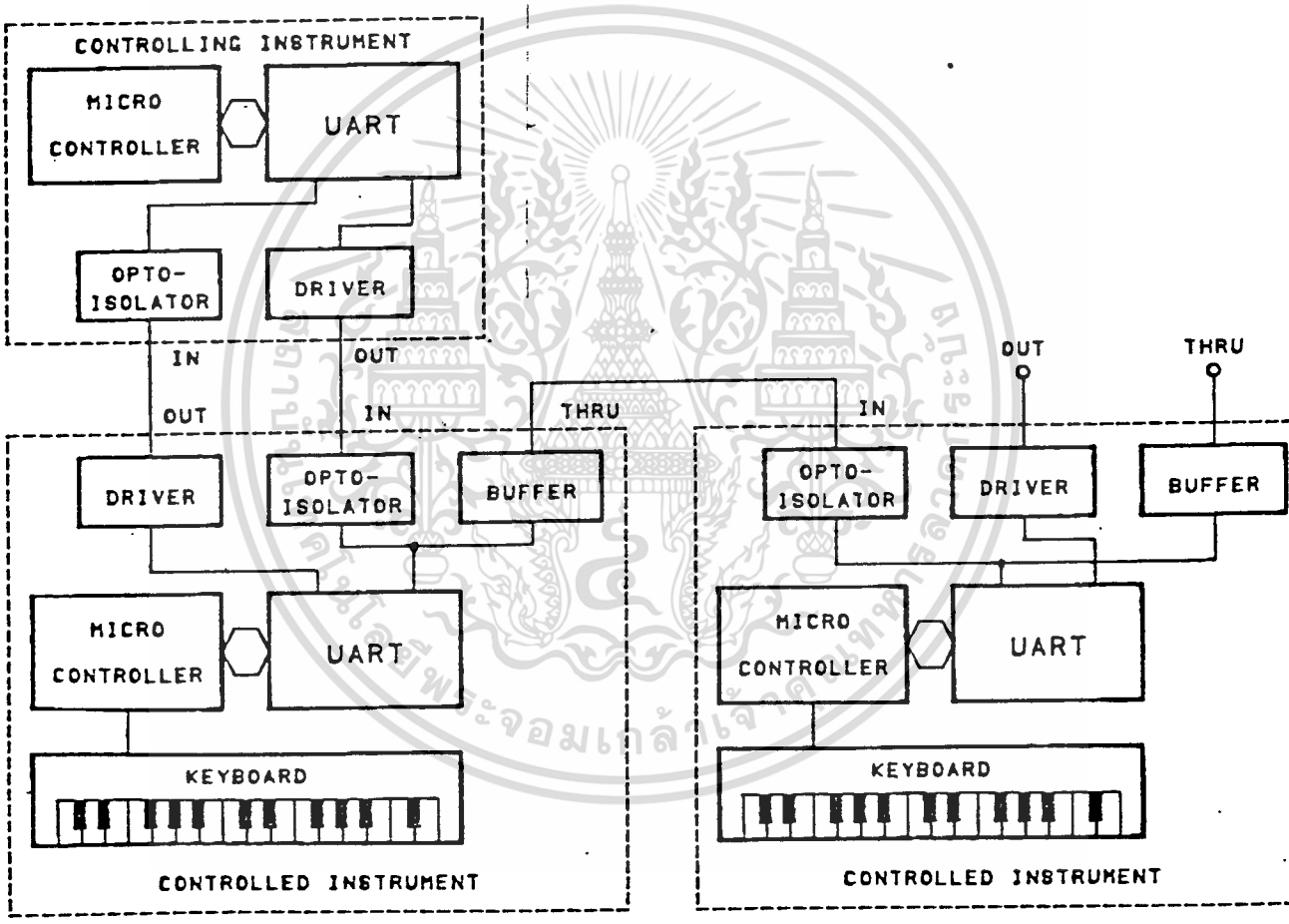
เป็นรายละเอียดของโปรแกรมที่เขียนขึ้นมาสำหรับโครงงานนี้ และสรุปลักษณะ, ความหมายของข้อความต่างๆในระบบมีดี้

บทที่ 2.

หลักการ และ ข้อกำหนดในระบบมีดี้

2.1 หลักการ

มีดี้ (MIDI) ย่อมาจาก Musical Instrument Digital Interface เป็นระบบมาตรฐานสำหรับการสื่อสารข้อมูลระหว่างเครื่องดนตรี โดยจะสื่อสารกันแบบ อซิงโครนัส (asynchronous) ซึ่งเขียนแผนภาพแสดงการทำงานได้ดังนี้



จากรูป อุปกรณ์ควบคุม (controlling instrument) เช่น คอมพิวเตอร์ สามารถควบคุมการทำงานของเครื่องดนตรีได้โดยคอมพิวเตอร์จะส่งรหัสมีดี้ไปยังคีย์บอร์ด 1 และใช้ออพโต้ไอโซเลเตอร์ (optoisolator) เป็นตัวคัปปลิ้งสัญญาณ ส่วน UART จะเปลี่ยนข้อมูลแบบขนานให้เป็นแบบอนุกรม (และเปลี่ยนจากแบบอนุกรมให้เป็นแบบขนานด้วย)

ข้อมูลที่ได้จะเข้ามายังส่วนควบคุมภายในเครื่องดนตรี (micro controller) เพื่อตีความ แล้วสั่งให้คีย์บอร์ดทำงานตามรหัสมีดี้ ที่รับได้

ข้อมูลจากออพโต้ไอโซเลเตอร์ส่วนหนึ่งจะผ่านไปยังบัฟเฟอร์แล้วส่งออกไป เรียกว่า มีดี้ ธรู (MIDI thru) ซึ่งก็คือการลอกแบบข้อมูลที่ได้รับมานั่นเอง ข้อมูลส่วนนี้จะถูกส่งไปให้คีย์บอร์ด 2 จะเห็นได้ว่า คีย์บอร์ดตัวแรกกับตัวที่สองจะติดต่อกันได้ โดย คีย์บอร์ดตัวแรกจะควบคุมคีย์บอร์ดตัวที่สอง ซึ่งอาศัยข้อมูลที่ได้จากมาจากคอมพิวเตอร์

ถ้าไม่อาศัยข้อมูลจากคอมพิวเตอร์ เมื่อมีการเล่น คีย์บอร์ด 1 (ใช้คนเล่น) ส่วนควบคุมภายในคีย์บอร์ด 1 จะส่งรหัสมีดี้ออกไป และคอมพิวเตอร์จะได้รับข้อมูลนี้แล้วนำมาทำขบวนการต่างๆได้ เช่น เก็บข้อมูลนี้ไว้ในดิสค์ ในลักษณะนี้จะคล้ายกับเทปบันทึกเสียง แต่แทนที่จะเก็บเป็นสัญญาณเสียง คอมพิวเตอร์จะเก็บในรูปของรหัสมีดี้ เมื่อต้องการเล่นซ้ำ (play back) ทางคอมพิวเตอร์ก็จะส่งรหัสมีดี้นี้กลับไปยังเครื่องดนตรี และเครื่องดนตรีก็จะทำงาน (เล่น) ตามรหัสมีดี้เดิม ซึ่งเป็นการสร้างเสียงขึ้นมาใหม่ (regenerate) นั่นเอง

จะเห็นว่าเมื่อเครื่องดนตรีสามารถสื่อสารกันภายใต้มาตรฐานเดียวกันแล้ว เราสามารถนำไปประยุกต์ใช้งานได้อย่างกว้างขวาง เช่น

1.ควบคุมเครื่องดนตรีได้ทีละหลายๆเครื่อง : เนื่องจากเครื่องดนตรีทุกเครื่องสื่อสารด้วยภาษาเดียวกัน

2.สามารถส่งรหัสมีดี้ไปควบคุมกลองอิเล็คโทรนิค (Drum machine setups) เพื่อให้มีจังหวะตรงกับเครื่องดนตรีอื่นๆ (synchronized) ได้

3.อุปกรณ์แต่งเสียง (sound effect) ต่างๆสามารถควบคุมได้ด้วยรหัสมีดี้

4.ใช้ในการสร้างเสียง (voice quality) ของซินธิไซเซอร์ (voice editing software)

5.ข้อมูลจากเครื่องดนตรีเครื่องหนึ่งอาจนำไปใช้กับอีกเครื่องได้ (Data transfer)

6.ถ้าใช้การเก็บรหัสมีดี้ แทนที่จะเก็บเป็นสัญญาณเสียง เราสามารถเล่นซ้ำได้ดังที่กล่าวไปแล้ว ซึ่งจะคล้ายกับการบันทึกเสียง (Tapeless recording)

7.ประโยชน์จากการสื่อสาร เช่นสามารถส่งข้อความมีดี้ผ่านโมเด็ม (modem) ได้ ดังนั้น แม้จะอยู่กันคนละประเทศ นักดนตรีก็ยังสามารถส่งดนตรีของเขาไปให้ห้องอัดเสียงได้ทางโทรศัพท์โดยไม่มีความเพี้ยนของเสียงดนตรีเลยเนื่องจากส่งเป็นโค๊ด

สำหรับเครื่องดนตรีที่ไม่ใช้ไฟฟ้า (acoustic) ก็สามารถใช้พ่วงกับเครื่องดนตรีอื่นด้วยระบบมีดี้ได้ ถ้าเราสามารถทำให้มันส่งข้อมูลออกไปตามข้อกำหนดของระบบมีดี้

2.2 ข้อกำหนดของระบบมีดี้

2.2.1 ข้อกำหนดโดยทั่วไป

มีดี้ เป็นมาตรฐานที่จะให้ ซินธิไซเซอร์ (synthesizer), ซีเควนเซอร์ (sequencer), คอมพิวเตอร์, อุปกรณ์สร้างจังหวะ ฯลฯ สามารถติดต่อสื่อสารกันได้

แต่ละอุปกรณ์ที่ใช้ระบบมีดี้ มักจะ รับ-ส่ง ข้อความในระบบมีดี้ได้ โดยทางด้านรับจะรับข้อความ (message) ที่เป็นรหัสมีดี้ แล้วทำงานตามความหมายของข้อความนั้น อุปกรณ์ทางฮาร์ดแวร์ที่สำคัญประกอบด้วยออพโต้ไอโซเลเตอร์, Universal Asynchronous Receiver-Transmitter (UART) ส่วนทางด้านส่งก็จะส่งรหัสมีดี้ออกไป โดยมี UART และ ตัวขับสัญญาณ (line driver) เป็นองค์ประกอบสำคัญ

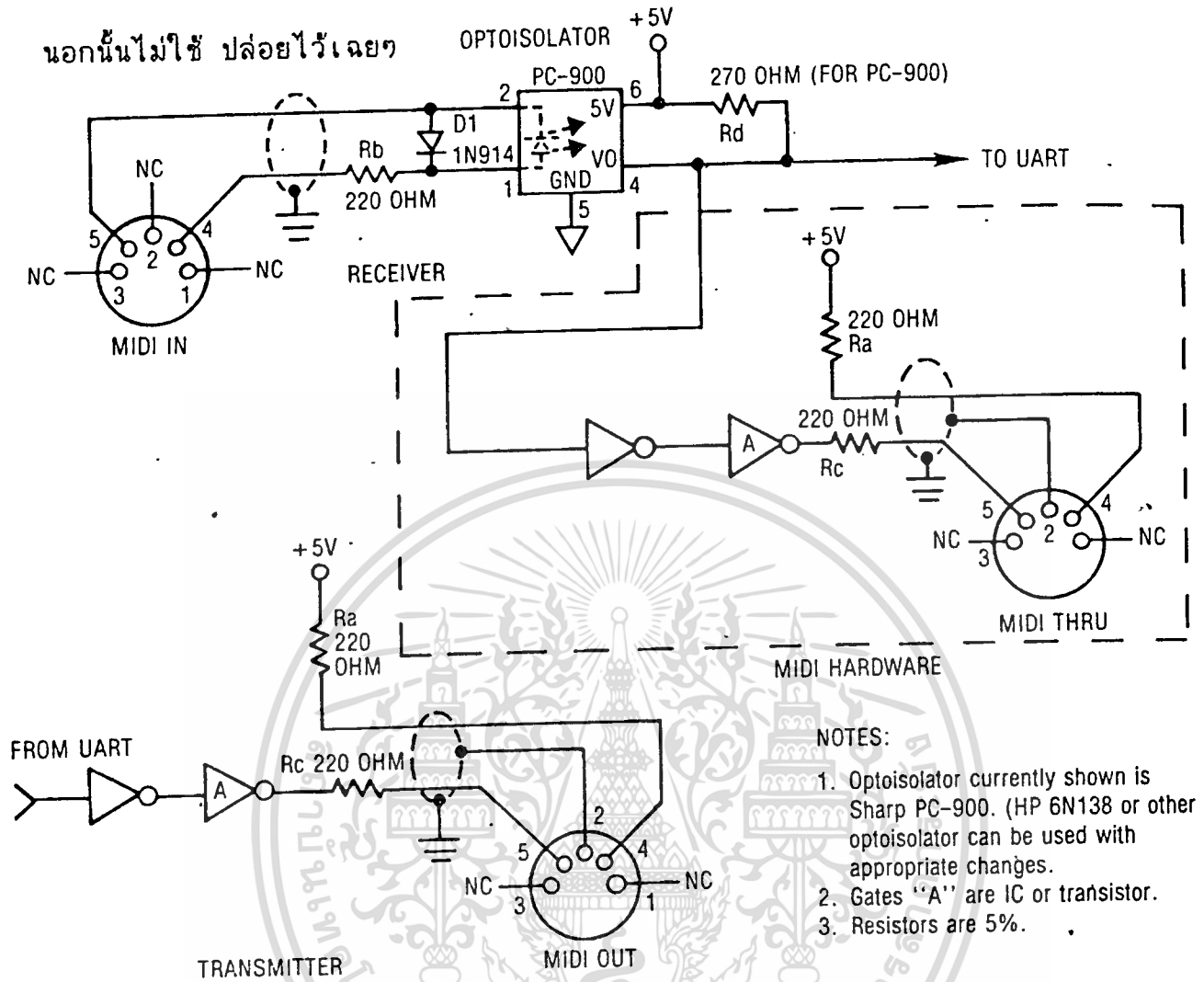
2.2.2 ฮาร์ดแวร์

สื่อสารแบบอซิงโครนัส โดยใช้ความเร็วในการรับ-ส่งที่ 31.25 kBaud มีบิทเริ่มและบิทหยุด (start bit,stop bit) อย่างละ 1 บิท, บิทข้อมูลมี 8 บิท รวมทั้งหมด 10 บิท ต่อ ข้อมูล 1 ไบท์ซึ่งใช้เวลาส่ง 320 ไมโครวินาที มีวงจรเป็นดังรูป 2.1

จากรูป 2.1 กระแสในลูป = 1.5 mA โดยที่ โลจิก 0 จะมีกระแสไหล ทางด้านรับควรมีออพโต้ไอโซเลเตอร์เป็นตัวคัปปลิ้งสัญญาณ ซึ่งเป็นชนิดทำงานที่ความเร็วสูง เช่น Sharp PC-900 เวลาในการสวิทช์ ควรน้อยกว่า 2 ไมโครวินาที

2.2.3 อุปกรณ์สำหรับเชื่อมต่อ (connector)

เป็นแบบ DIN 5 ขา (แจ๊คกลม) ตัวเมีย ขาที่ใช้คือขา 5, 4 ส่วนขา 2 ต่อกราวนด์



รูป 2.1 ฮาร์ดแวร์ของระบบมีดี้

ความยาวของสาย (MIDI cable) ไม่เกิน 15 เมตร และควรมีการพันเกลียว (twist) เพื่อลดสัญญานรบกวน

ชื่อของคอนเนคเตอร์ : MIDI IN (ข้อมูลเข้า), MIDI OUT (ข้อมูลออก) และ MIDI THRU ซึ่งเป็นข้อมูลขาออกที่ได้มาจากการก๊อปปี้ข้อมูลจาก มีดี้ อิน โดยตรง

2.2.4 รูปแบบของข้อมูล (data format)

แต่ละข้อความในระบบมีดี้ มักมีหลายไบท์ โดยประกอบด้วยไบท์สถานะ (status byte) ตามด้วยไบท์ข้อมูล (data byte) อีก 1 หรือ 2 ไบท์ ยกเว้นข้อความเกี่ยวกับเวลา (real-time message) และ ข้อความพิเศษ (exclusive message) ซึ่งจะกล่าวถึงภายหลัง ประเภทของข้อความ จะแบ่งออกเป็น 2 ประเภทใหญ่ๆ คือ

แชนแนล กับ ระบบ

ข้อความเกี่ยวกับแชนแนล

ในไบท์สถานะ จะใช้ 4 บิทเป็นตัวกำหนดแชนแนลของไบท์สถานะ (สามารถกำหนดได้ 16 แชนแนล) ทางด้านรับจะตอบสนองเฉพาะไบท์สถานะที่มีแชนแนลตรงกับแชนแนลของตนเท่านั้น ถ้าไม่ตรงกันจะไม่สนใจ

ข้อความเกี่ยวกับแชนแนล มี 2 ประเภท

Voice - ใช้ควบคุมเสียง ข้อความเกี่ยวกับเรื่องเสียงจะถูกส่งผ่านทางแชนแนลเสียง (voice channel)

Mode - ใช้กำหนดลักษณะการตอบสนองต่อข้อความเกี่ยวกับเรื่องเสียงของเครื่องดนตรี โดยจะส่งมาทาง แชนแนลพื้นฐาน (Basic channel) ของเครื่องดนตรี

ข้อความเกี่ยวกับระบบ

ข้อความเกี่ยวกับระบบจะไม่มีการกำหนดแชนแนล ดังนั้น ไม่ว่าด้านรับจะเป็นแชนแนลไหน ก็จะตอบสนองต่อข้อความเกี่ยวกับระบบเสมอ

ข้อความเกี่ยวกับระบบ มี 3 แบบ

1. ข้อความทั่วไป (common message) เป็นข้อความสำหรับทุกอุปกรณ์ในระบบ
2. ข้อความเกี่ยวกับเวลา จะมีแต่ไบท์สถานะเท่านั้น ไม่มีไบท์ข้อมูล สำหรับข้อความประเภทนี้จะส่งเมื่อไหร่ก็ได้ แม้แต่ส่งคั่นระหว่างไบท์ของข้อความอื่น
3. ข้อความพิเศษ (exclusive message) มีหลายไบท์ข้อมูล และจะจบด้วยไบท์ที่เรียกว่า End Of Exclusive (EOX) หรือไบท์สถานะอื่นๆ เป็นข้อความที่ใช้เฉพาะกับอุปกรณ์ของผู้ผลิตที่กำหนดไว้ในรหัสประจำเครื่องของแต่ละบริษัทผู้ผลิตหรือ Identification Codes (ID codes) เท่านั้น ถ้าอุปกรณ์ทางด้านรับมีรหัสประจำเครื่องไม่ตรงกับข้อความพิเศษที่ได้ มันจะไม่สนใจไบท์ข้อมูลที่ตามมา ผู้ผลิตแต่ละรายสามารถกำหนดรูปแบบ ความหมายข้อมูลของตนเองได้ โดยส่งเป็นข้อความพิเศษ ที่มีรหัสประจำเครื่องของตน (ซึ่งเป็นรหัสเดียวกันกับทางด้านรับ)

ประเภทของข้อมูล (DATA TYPES)

ไบท์สถานะ : มี 8 บิทโดยที่บิทที่มีนัยสำคัญสูงสุด (MSB) ถูกเซ็ท (เป็น 1) ไบท์สถานะ จะเป็นตัวบอกถึงประเภทของข้อความ และบอกให้รู้ว่าไบท์ข้อมูลที่ตามมาจะเอาไปใช้เพื่ออะไร

สถานะที่กำลังทำงาน (running status) : สำหรับข้อความเกี่ยวกับเสียงและ โหมดเท่านั้น เมื่อด้านรับได้รับไบท์สถานะและทำขบวนการต่างๆ ด้านรับจะยังคงอยู่ในสถาณะนั้นจนกว่าจะได้รับไบท์สถานะใหม่ที่ไม่เหมือนเดิม ดังนั้น ถ้ามีการส่งไบท์สถานะเดิมซ้ำมาอีกครั้ง อาจทำเมื่อต้องการแก้ไขไบท์ข้อมูลที่ผิดโดยส่งใหม่ ภายใต้สถานะที่กำลังทำงานข้อความที่สมบูรณ์ต้องประกอบด้วยไบท์ข้อมูลที่ถูกต้องตามข้อกำหนด

สถานะที่กำลังทำงานจะหยุดเมื่อมีไบท์สถานะใหม่เข้ามา เว้นแต่ ถ้ามีข้อความเกี่ยวกับเวลาเข้ามา มันจะขัดจังหวะสถานะที่กำลังทำงานแต่เพียงชั่วคราวเท่านั้น

สถาณะที่ไม่สามารถทำงานได้ (Unimplement Status) : ทางด้านรับจะไม่สนใจไบท์สถานะที่มันไม่สามารถทำงานตามได้ และจะไม่สนใจไบท์ข้อมูลที่ตามมา

สถานะที่ไม่มีการกำหนดความหมาย (Undefined Status) : จะไม่มีการใช้ไบท์สถานะที่ไม่มีความหมาย ควรระวังในขณะเปิดเครื่อง หรือ บิดเครื่อง อาจส่งข้อมูลผิดพลาดได้

ไบท์ข้อมูล : จะถูกส่งตามหลังไบท์สถานะ โดยมีไบท์ข้อมูลประมาณ 1-2 ไบท์ (ยกเว้นข้อความเกี่ยวกับเวลา) ไบท์ข้อมูลมี 8 บิท โดยบิทที่มีนัยสำคัญต่ำสุดถูกรีเซ็ท (เป็น 0) จำนวน และ ขอบเขต (ค่าที่มีได้) ของไบท์ข้อมูล จะเป็นไปตามที่ไบท์สถานะกำหนด ในแต่ละไบท์สถานะจะส่งไบท์ข้อมูลจำนวนที่ถูกต้องตามไปเสมอ การตอบสนองต่อข้อความจะมีได้ก็ต่อเมื่อด้านรับได้รับไบท์ข้อมูลตามต้องการแล้ว ทางด้านรับจะไม่สนใจกับไบท์ข้อมูลที่ตามหลังไบท์สถานะที่มันไม่รู้จัก

โหมดต่างๆในแต่ละแชนแนล (channel Mode)

ซินธิไซเซอร์จะประกอบด้วย ส่วนที่สร้างเสียง (sound generator) เสียง

ที่เกิดขึ้นเรียกว่า voice การกำหนดเสียงเป็นขบวนการเกี่ยวกับข้อมูลพวกการเล่นโน๊ต (note on) กับการเลิกเล่นโน๊ต (note off) จากคีย์บอร์ด ซึ่งเสียงที่มีโน๊ตต่างๆจะถูกเล่นได้ในจังหวะที่ถูกต้อง

ข้อความเกี่ยวกับโหมด (mode message) ใช้กำหนดลักษณะการสร้างเสียงของซินธิไซเซอร์ ได้แก่ Omni (On/Off), Poly และ Mono

Mono on จะเป็นการกำหนดให้เสียงในแชนแนลเสียงมีได้ 1 เสียงเท่านั้นในเวลาเดียวกัน (Mono phonic)

Mono off (Poly on) : ซินธิไซเซอร์ที่อยู่ในโหมดนี้สามารถสร้างเสียงได้พร้อมๆกันทีละหลายๆเสียง (Poly phonic) ทั้งนี้จะขึ้นอยู่กับความสามารถในการสร้างเสียงของซินธิไซเซอร์ด้วย

สำหรับ Omni ถ้า Omni on จะตัวรับจะสามารถรับข้อความเกี่ยวกับเสียงได้จากทุกแชนแนลเสียงโดยไม่มีความต่างกันของเชนแนล ถ้า Omni off ด้านรับจะรับเฉพาะข้อความเกี่ยวกับเสียงที่มีเชนแนลตรงกับของตนเท่านั้น

ทางด้านรับที่ถูกกำหนดเป็นแชนแนลพื้นฐานจะมีลักษณะการทำงานได้ 4 โหมดดังนี้

โหมด 1. Omni On , Poly

โหมด 2. Omni On , Mono

โหมด 3. Omni Off, Poly

โหมด 4. Omni Off, Mono

ทางด้านส่งและรับจะทำงานได้ทีละ 1 โหมดเท่านั้นในเวลาเดียวกัน และตามปกติจะส่งและรับในโหมดเดียวกัน

ด้านรับจะรับรู้ข้อความเกี่ยวกับโหมดทางแชนแนลพื้นฐานเท่านั้น ส่วนข้อความเกี่ยวกับเสียงอาจได้รับทางแชนแนลพื้นฐานหรือแชนแนลอื่นๆก็ได้ ซึ่งเรียกรวมกันว่าแชนแนลเสียง

โดยทั่วไปมักกำหนดให้แชนแนลแรก (0) เป็นแชนแนลพื้นฐาน

2.2.5 ความหมายของข้อความต่างๆในระบบมีดี้

1. ข้อความที่เกี่ยวกับเสียงในแต่ละแชนแนล (channel voice message)

การเล่นโน๊ต (note on)

เป็นข้อความที่บอกให้รู้ถึงการเล่นโน๊ต (เช่นการเอานิ้วกดไปที่คีย์บอร์ด) มี 3 ไบท์

1. ไบท์สถานะคือ 1001nnnn (ฐานสอง) โดยที่ nnnn คือหมายเลขแชนแนล (0-15)
2. ไบท์ข้อมูลแรกจะบอกให้รู้ว่าโน๊ตไหนที่ถูกกด (note number) มีค่าได้ตั้งแต่ 0-127 โดย 0 จะเป็นโน๊ตที่ต่ำที่สุด, 127 เป็นโน๊ตที่สูงที่สุด แต่โดยทั่วไปเครื่องดนตรีจะรับรู้โน๊ตได้น้อยกว่านี้ เช่นซินธิไซเซอร์ที่มี 5 ออคเตฟจะรับรู้หมายเลขโน๊ตได้ตั้งแต่ 36-96 หมายเลขโน๊ตของโดกลาง (middle C) คือ 60
3. ไบท์ข้อมูลต่อมาจะบอกถึงน้ำหนักในการเล่นหรือความเร็วที่กดคีย์(key velocity) เหมือนกับการเล่นเปียโนถ้ากดคีย์เปียโนด้วยความเร็วจะได้เสียงที่ดัง และถ้ากดช้าก็จะได้เสียงที่เบาเช่นกัน ความเร็วของคีย์ มีค่าได้ตั้งแต่ 0-127 โดยความเร็ว =127 จะเป็นเสียงที่ดังที่สุด ส่วนความเร็ว=0 จะไม่มีเสียงออกมา (note off) ซึ่งมีความหมายเหมือนการยกนิ้วขึ้นจากคีย์ สำหรับเครื่องดนตรีที่ไม่มีการตอบสนองต่อความเร็วของคีย์จะกำหนดให้ข้อมูลส่วนนี้มีค่าเท่ากับ 64

การเลิกเล่นโน๊ต (note off)

เป็นข้อความที่บอกให้รู้ถึงการเลิกสร้างเสียง(เช่นเกิดขึ้นในขณะที่ยกนิ้วขึ้นจากคีย์) มี3ไบท์

1. ไบท์สถานะ คือ 1000nnnn (ฐานสอง) โดยที่ nnnn คือ หมายเลขแชนแนล
2. ไบท์ข้อมูลแรกบอกให้รู้ว่าโน๊ตตัวไหนที่จะให้เลิกสร้างเสียง (key number)
3. ความเร็วที่ปล่อยคีย์ (key off (release) velocity) เป็นไบท์ข้อมูลที่บอกถึงความเร็วในการปล่อยคีย์ ในกรณีที่อุปกรณ์ไม่มีความเร็วของคีย์จะให้ไบท์นี้มีค่าเป็น 64

เมื่อเครื่องดนตรีได้รับข้อความเกี่ยวกับการเล่นโน๊ต มันจะผลิตเสียงค้างอยู่อย่างนั้น จนกว่าจะได้รับข้อความเกี่ยวกับการเลิกเล่นโน๊ตของโน๊ตตัวนั้นมันถึงจะหยุด

ทำเสียง ดังนั้นเมื่อมีการส่งข้อความเกี่ยวกับการเล่นโน๊ตออกไปแล้ว จะต้องส่งข้อความเกี่ยวกับการเลิกเล่นโน๊ตตามมาด้วยทุกครั้ง มิฉะนั้นจะเกิดเสียงค้าง

การกำหนดความดันของคีย์แบบโพลีโฟนิค (Polyphonic Key pressure)

คีย์บอร์ดในระบบมีดี้บางเครื่องไม่เพียงแต่ตอบสนองความเร็วของคีย์ได้เท่านั้น ยังตอบสนองน้ำหนัก (ความดัน) ของนิ้วที่กดลงไปอีกหลังจากที่กดคีย์ลงไปสุดแล้ว มี 3 ไบท์

1. ไบท์สถานะ คือ 1010nnnn
2. หมายเลขของโน๊ต
3. ความดันของคีย์ (key pressure value) มีค่าตั้งแต่ 0-127 โดย 0 หมายถึงไม่มีความดันเลย และ 127 จะมีความดันสูงสุด

ประโยชน์ของโพลีโฟนิคคีย์เพรชเชอร์ คือ จะให้ความรู้สึกของเครื่องเป่า เช่น ทรัมเป็ต เมื่อตั้งเสียงของซินธิไซเซอร์เป็นเสียงทรัมเป็ต การกดคีย์ลงไปอีกหลังจากสุดคีย์แล้วจะมีลักษณะเหมือนการที่เราเป่าลมเข้าไปในทรัมเป็ตอีก

คีย์บอร์ดที่มีโพลีโฟนิคคีย์เพรชเชอร์จะมีระบบที่ซับซ้อน และมีราคาแพงมาก ซึ่งจะหาคีย์บอร์ดที่มีระบบนี้ได้น้อยราย

การกำหนดความดันรวม (Overall Pressure)

ต่างกับโพลีโฟนิคคีย์เพรชเชอร์ตรงที่ไม่สามารถบอกความดันแยกกันของแต่ละคีย์ที่ถูกกดได้แต่จะบอกเป็นความดันเฉลี่ยของทุกๆคีย์ในเวลานั้น มีอยู่ด้วยกัน 2 ไบท์

1. ไบท์ข้อมูลคือ 1101nnnn (nnnn คือ หมายเลขแชนแนล)
2. ความดันของแชนแนล (channel pressure value) มีค่าตั้งแต่ 0-127 โดยที่ 0 จะไม่มีความดัน และ 127 มี ความดันมากที่สุด

ข้อมูลเกี่ยวกับการควบคุม (Control Change)

ใช้ในการปรับแต่งเสียง สำหรับข้อมูลเกี่ยวกับการควบคุมในปัจจุบันยังไม่มีมาตรฐานที่แน่นอนระหว่างแต่ละบริษัทผู้ผลิต มี 3 ไบท์

1. ไบท์สถานะ คือ 1011nnnn
2. หมายเลขตัวควบคุม (controller number) มีค่าตั้งแต่ 0-127 ดูความหมาย

ได้จาก MIDI Implementation Sheet ของแต่ละบริษัทผู้ผลิต

3. ค่าที่จะให้แก่ตัวควบคุม (controller value)

การเอื้อนเสียง (Pitch Bend)

เป็นลักษณะของการเอื้อนเสียงจากสูงไปต่ำ หรือจากต่ำไปสูง โดยที่ไม่ต้องเปลี่ยนตำแหน่งของคีย์ที่กดการเอื้อนเสียงมี 2 ไบท์

1. ไบท์สถานะ คือ 1110nnnn

2. ค่าที่จะให้เอื้อน (pitch bend value)

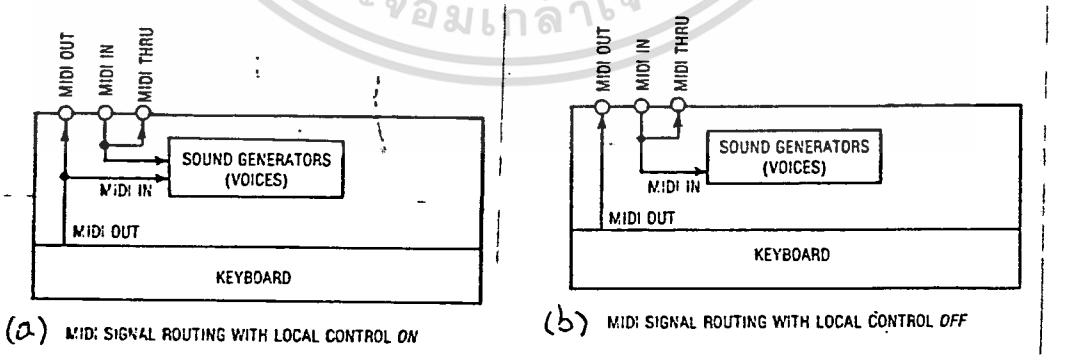
ข้อมูลเกี่ยวกับโปรแกรมเสียง (Program Change) หรือบางทีเรียกว่า Program Select จะบอกให้รู้ถึงการใช้โปรแกรมเสียงต่างๆในเครื่องดนตรี มี 2 ไบท์

1. ไบท์สถานะ คือ 1100nnnn

2. โปรแกรมเสียงที่ใช้ (selected program number) มีได้ตั้งแต่ 0 ถึง 127

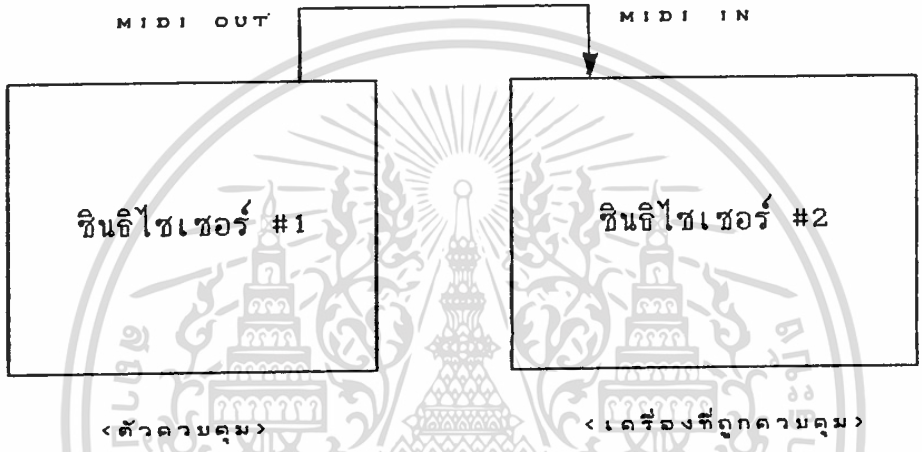
<u>ข้อมูลเกี่ยวกับลักษณะการทำงานในแต่ละแชนแนล</u> (Channel Mode Messages) เป็นข้อความเกี่ยวกับโหมดในการทำงาน เป็นส่วนหนึ่งของข้อความเกี่ยวกับการควบคุม

การควบคุมคีย์บอร์ดแบบ โลคอล/รีโหมด (Local/Remote Keyboard Control ใช้ในการผ่วงเครื่องดนตรีเข้าด้วยกันโดยควบคุมจากเครื่องเดียว



รูป 2.3 ทางเดินของสัญญาณมีดี้

จากรูป 2.3(a) ถ้าเป็นการควบคุมแบบโลคอล (Local Control On) ข้อมูลทางด้านเสียง (เช่นการเล่นโน๊ต) จากคีย์บอร์ดจะเข้าไปยังส่วนสร้างเสียงและข้อมูลเดียวกันนี้จะออกไปทางมีดี้เอ๊าท์ด้วย (ข้อมูลจากมีดี้เอ๊าท์ใช้ส่งให้กับอุปกรณ์อื่น) ดังนั้นในการพ่วงซินธิไซเซอร์เข้าด้วยกันดังรูป 2.4 เมื่อเล่นซินธิไซเซอร์ #1 จะมีเสียงออกมาจากทั้งซินธิไซเซอร์ #1 และซินธิไซเซอร์ #2 (ซินธิไซเซอร์#2 เล่นตามข้อมูลที่ส่งมาจากซินธิไซเซอร์ #1



รูป 2.4

ในกรณีที่เป็นการควบคุมแบบรีโหมด (Local control off) (รูป 2.3(b)) ข้อมูลจากคีย์บอร์ดจะส่งไปที่ มีดี้เอ๊าท์ โดยตรง (ไม่ส่งให้ส่วนสร้างเสียง) ดังนั้นในรูป 2.4 ถ้าอยู่ในลักษณะการควบคุมแบบรีโหมด เมื่อเล่นที่ซินธิไซเซอร์ #1 จะมีเสียงออกมาจาก ซินธิไซเซอร์ #2 เท่านั้น

ข้อมูลเกี่ยวกับการควบคุมคีย์บอร์ดแบบ โลคอล/รีโหมด มี 3 ไบท์

1. ไบท์สถานะ คือ 1011nnnn (ซึ่งเป็นไบท์สถานะของข้อความเกี่ยวกับการควบคุม)

   nnnn คือ แชนแนลพื้นฐาน

2. หมายเลขตัวควบคุม = 122

3. ถ้าเป็นแบบรีโหมดจะให้ไบท์นี้เป็น 0, ถ้าเป็นแบบโลคอลจะเป็น 127

การเลิกเล่นโน๊ตทุกตัว (All Note Off)

เป็นการหยุดการสร้างเสียง (หยุดทั้งแชนแนลโดยไม่สนว่าเป็นโน๊ตตัวไหน) มี 3 ไบท์

1. ไบท์สถานะ คือ 1011nnnn (เป็นไบท์สถานะของข้อความเกี่ยวกับการควบคุมนั่นเอง) nnnn คือแชนแนลพื้นฐาน
2. หมายเลขตัวควบคุม = 123
3. เป็นไบท์ที่ไม่มีความหมายอะไร ใส่ไปให้ครบ 3 ไบท์เท่านั้น (dummy byte) เนื่องจากข้อความเกี่ยวกับการควบคุมมี ความยาว 3 ไบท์

การเลือกโหมด Omni/Poly/Mono

ได้อธิบายเกียวกับโหมดเหล่านี้ไปแล้วในตอนต้น ทุกครั้งที่เปลี่ยนโหมดจะเลิกเล่นโน๊ตทุกตัว รูปแบบของข้อความนี้ก็คล้ายๆกับข้อความเกี่ยวกับการควบคุมอื่นๆ ประกอบด้วย

1. ไบท์สถานะ คือ 1011nnnn (nnnn เป็นแชนแนลพื้นฐาน)
2. หมายเลขตัวควบคุมของแต่ละโหมด เป็นดังนี้

   Omni Off = 124

   Omni On = 125

   Mono On = 126

   Poly On = 127

3. ไบท์ที่ 3 จะให้เป็น 0 สำหรับ ตัวควบคุมหมายเลข 123,124,125,127 ในกรณีของโหมดโมโน (126) ไบท์นี้จะเป็นการกำหนดแชนแนลที่จะให้เป็นโมโน

<u>ข้อมูลทั่วไปของระบบ</u> (System Common Message) (สำหรับทุกแชนแนล)

เนื่องจากข้อมูลประเภทนี้เป็นข้อมูลของระบบ ดังนั้นจะไม่มีการกำหนดแชนแนล นั่นคือทุกแชนแนลต้องตอบสนองต่อข้อมูลทั่วไปของระบบ

การชี้ตำแหน่งของเพลง (Song Position Pointer)

บอกให้รู้ถึงตำแหน่งของเพลงที่กำลังเล่นอยู่ การบอกตำแหน่งจะบอกว่าเพลงได้ดำเนินไปแล้วกี่บีท (MIDI beat) บอกได้สูงสุด 16,384 บีท โดย หนึ่งบีทมีค่าเท่ากับ โน๊ตขเบ็จสองชั้นหนึ่งตัว

การเลือกเพลง (Song Select)

คล้ายๆกับการเลือกโปรแกรมเสียง แต่แทนที่จะเป็นโปรแกรมเสียง จะเป็นโปรแกรม

รูปแบบ (pattern) สั้นๆของเพลงแทน (ความยาวไม่กี่ห้อง) โดยมากจะใช้ในกลองอิเลคโทรนิค แต่ละเพลงก็คือแต่ละจังหวะ เช่น เพลงที่หนึ่ง (song1) เป็นจังหวะร๊อค และเพลงที่สอง (song2) เป็นจังหวะวอลซ์

ข้อความพิเศษของระบบ (System Exclusive) : ได้อธิบายแล้วในตอนต้น

ข้อความเกี่ยวกับเวลาของระบบ (System Real Time Message) (ทุกแชนแนล) เป็นข้อความที่ทำให้เครื่องดนตรีต่างๆเล่นอยู่ในจังหวะเดียวกัน (Synchronized) เช่นในการพ่วงซินธิไซเซอร์ กับ กลองอิเล็คโทรนิคเข้าด้วยกันจำเป็นอย่างยิ่งที่อุปกรณ์ทั้งสองนี้จะต้องทำงานอยู่ในจังหวะเดียวกัน ไม่เช่นนั้นก็คงจะฟังไม่รู้เรื่องถ้าปล่อยให้ต่างฝ่ายต่างเล่น

การรีเซ็ทระบบ (System Reset)

เป็นข้อความที่บอกให้อุปกรณ์ระบบมีดี้ทุกชิ้นทำการรีเซ็ทตัวเอง (กลับไปอยู่ในสภาพแรกหลังจากเปิดไฟเข้าเครื่อง)

สัญญาณนาฬิกา (Timing Clock)

ในการกำหนดจังหวะจะมีการส่งสัญญาณนาฬิกาออกไปด้วยอัตรา 24 ลูก ต่อ โน๊ตตัวดำหนึ่งตัว (ส่งออกไปอย่างต่อเนื่อง) เพื่อให้เป็นจังหวะอ้างอิง

ในระบบควรใช้สัญญาณนาฬิกาเดียวกัน (มีอุปกรณ์ส่งสัญญาณนาฬิกาเพียงตัวเดียว)

การกลับไปที่จุดเริ่มต้น (Start From First Measure)

เป็นข้อความที่บอกให้ซีเควนเซอร์ และ กลองอิเลคโทรนิคกลับไปที่จุดเริ่มต้นของเพลง แล้วเริ่มเล่นเพลงในทันทีที่ได้รับสัญญาณนาฬิกา นั่นคือจะบอกให้แต่ละอุปกรณ์เริ่มเล่นพร้อมๆกัน

การหยุด (Stop)

ข้อความนี้จะบอกให้หยุดการกระทำทุกอย่างที่มีผลมาจากการกำหนดจังหวะ (timing-sensitive)

การเริ่มเล่นต่อ (Continue Start)

บอกให้อุปกรณ์ทำการเล่นต่อหลังตำแหน่งเพลงที่ถูกหยุดครั้งสุดท้าย

การตรวจสอบสภาวะการทำงาน (Active Sensing)

เป็นข้อความที่ใช้แก้ปัญหาตัวส่งกับตัวรับถูกตัดขาดออกจากกัน (เช่นสายหลุด) ซึ่งจะก่อให้เกิดความผิดพลาด เช่น ส่งข้อความให้เล่นโน๊ตออกไปแล้วเกิดสายหลุด เมื่อทางด้านส่ง ส่งข้อความให้เลิกเล่นโน๊ตตามไป ก็จะไปไม่ถึงด้านรับ ทำให้เกิดเสียงค้างที่เครื่องดนตรีด้านรับ ปัญหานี้จะเกิดความเสียหายมากในการแสดงสด

แอคถีฟเซ็นซิ่ง จะทำงานโดยการส่งข้อความที่เรียกว่าแอคถีฟเซ็นซิ่งออกไปเรื่อยๆ เมื่อด้านรับได้รับแอคถีฟเซ็นซิ่งแล้ว ถ้าภายในเวลาสั้นๆที่กำหนดไม่ได้รับข้อความนี้เข้ามาอีก มันจะหยุดการสร้างเสียง

ที่กล่าวมาแล้วทั้งหมดเป็นความหมายของข้อความในระบบมีดี้ต่างๆ แต่โดยทั่วไปเครื่องดนตรีแต่ละชิ้นจะไม่สามารถตอบสนองต่อข้อความเหล่านี้ได้ทุกข้อความ ซึ่งความสามารถในการตอบสนองต่อข้อความต่างๆของแต่ละอุปกรณ์จะดูได้ใน MIDI Implementation Sheet ของอุปกรณ์นั้นๆ

บทที่ 3

การสร้าง

3.1 มีดี้ซีเควนเซอร์ (MIDI sequencer)

เป็นอุปกรณ์ที่ใช้ควบคุมเครื่องดนตรี โดยซีเควนเซอร์จะส่งข้อมูลในการเล่นไปให้เครื่องดนตรี ซีเควนเซอร์มีประโยชน์มากสำหรับสตูดิโอ นักดนตรีจะป้อนโน๊ตเพลงที่เขาแต่งหรือต้องการจะเล่นรวมทั้งรายละเอียดในการเล่นอื่นๆเช่น โปรแกรมเสียง ความดัง ให้กับซีเควนเซอร์ จากนั้นมันจะส่งสัญญาณควบคุมออกไปสั่งให้เครื่องดนตรีเล่นตามโน๊ตและข้อมูลอื่นๆที่ป้อนให้ สำหรับ มีดี้ซีเควนเซอร์ สัญญาณควบคุมที่ส่งออกไปก็คือข้อความในรหัสมีดี้นั่นเอง ดังนั้นเมื่อใช้ซีเควนเซอร์ แล้วเครื่องดนตรีจะถูกเล่นได้อย่างถูกต้องโดยไม่ต้องใช้นักดนตรี (ไม่ต้องใช้คนเล่น)

การป้อนข้อมูลในการเล่นให้กับซีเควนเซอร์สามารถทำได้โดย

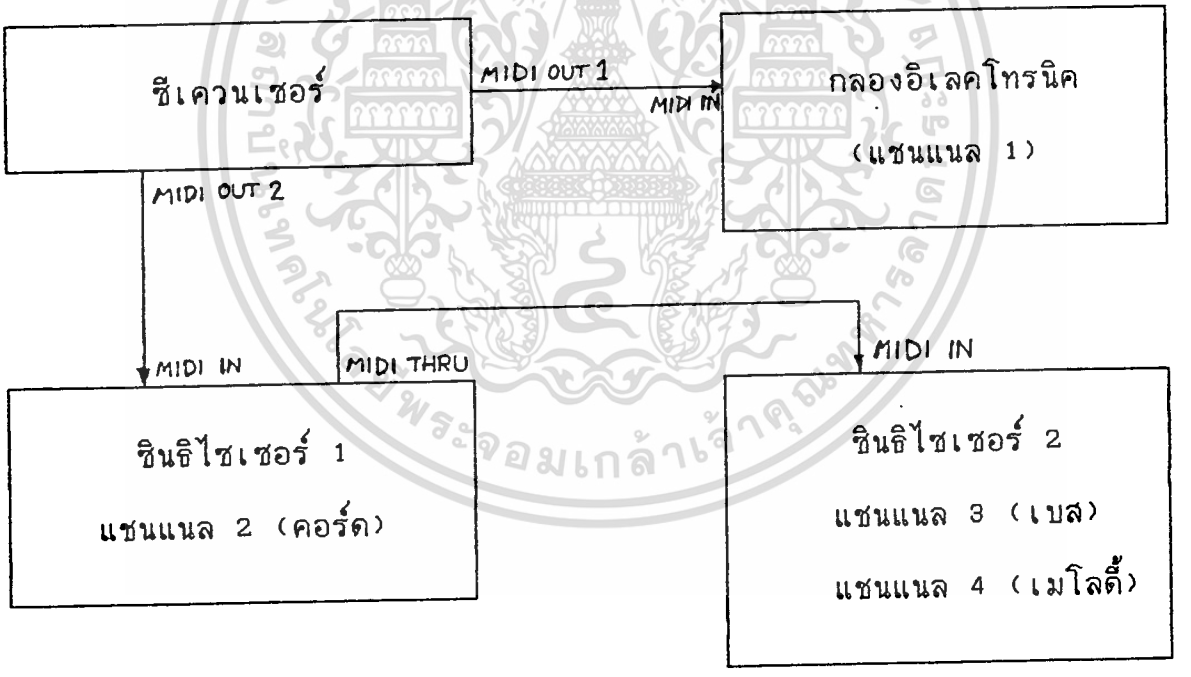
1. ป้อนข้อมูลในการเล่นทีละขั้น (step programming) เช่น บอกเป็นโน๊ตที่ละตัว วิธีนี้มีข้อดีคือสามารถบอกข้อมูลได้ถูกต้องแม่นยำ การป้อนโปรแกรมทำเมื่อไหร่ก็ได้ เช่นวันนี้โปรแกรมท่อนแรกก่อนแล้วค่อยมาโปรแกรมท่อนต่อไปวันหลัง นอกจากนั้นยังง่ายต่อการตรวจสอบ แก้ไขข้อมูล แต่มีข้อเสียคือยุ่งยากในการป้อนข้อมูล ในการบอกเป็นโน๊ต ผู้ที่เขียนโน๊ตไม่เป็นจะไม่สามารถใช้งานได้ดังนั้นการป้อนข้อมูลทีละขั้น จึงเหมาะที่จะใช้กับงานที่ต้องการความถูกต้องแม่นยำสูง เช่นในการบันทึกเสียง

2. ป้อนข้อมูลจากเหตุการณ์จริง (real time programming) เช่น นักดนตรีเล่นคีย์บอร์ดข้อมูลในการเล่นที่เกิดขึ้นที่คีย์บอร์ดจะถูกส่งไปให้ซีเควนเซอร์ ข้อมูลเหล่านี้อาจนำไปทำขบวนการอื่นอีกเช่นแก้ไข หรือ quantize ซึ่งเป็นการปรับปรุงข้อมูลให้เหมาะสมเช่นจังหวะในการเล่นของนักดนตรีอาจยังไม่ถูกต้อง ถ้าโน๊ตที่นักดนตรีเล่นมีความยาวใกล้เคียงกับโน๊ตตัวดำ (ในกรณีที่คนเล่นมักมีความคลาดเคลื่อนทางจังหวะ (timing) อยู่เสมอ) ซีเควนเซอร์อาจตัดสินให้เป็นโน๊ตตัวดำไปเลยวิธีนีมีข้อดีคือง่ายในการป้อนข้อมูลถ้านักดนตรีสามารถเล่นได้อย่างที่ต้องการ แต่มีข้อเสียคือ ดนตรีที่ได้จะมีความถูกต้องแม่นยำน้อย เนื่องจากขึ้นอยู่กับความสามารถในการเล่นของนัก

ดนตรี และไม่สามารถใช้การป้อนข้อมูลจากเหตุการณ์จริงได้กับผู้ที่เล่นดนตรีไม่เป็น ดังนั้นการป้อนข้อมูลแบบนี้เหมาะกับนักดนตรีที่ต้องการเก็บดนตรีของตนไว้โดยไม่สนใจความถูกต้องนัก เช่น เพื่อเตือนความจำสำหรับนักแต่งเพลงเมื่อคิดแนวเพลงใหม่ๆได้ หรือใช้เป็นดนตรีประกอบ (back up) ในการซ้อมดนตรี ซีเควนเซอร์ที่ใช้การป้อนข้อมูลจากเหตุการณ์จริงก็จะทำหน้าที่เหมือนเทปบันทึกเสียงนั่นเอง

ในขณะที่กำลังป้อนข้อมูลด้วยวิธีนี้ ซีเควนเซอร์จะส่งสัญญาณจังหวะอ้างอิง(reference timing) ออกมาด้วยเพื่อให้นักดนตรีใช้เป็นจังหวะอ้างอิง

ในการเชื่อมต่อซีเควนเซอร์เข้ากับเครื่องดนตรีอื่นๆทำได้ดังรูป 3.1 เครื่องดนตรีแต่ละเครื่องจะกำหนดให้มีแชนแนลเสียงไม่เหมือนกัน ซีเควนเซอร์จะส่งข้อมูล



รูป 3.1 การต่อซีเควนเซอร์เข้ากับเครื่องดนตรี

รวมๆของทุกแชนแนลออกไปซึ่งทางเครื่องดนตรีจะรับเอาเฉพาะข้อมูลของแชนแนลตัวเองเท่านั้น เช่น ส่งข้อมูลเกี่ยวกับการตีกลองออกไปให้กลองอิเลคโทรนิคทางแชนแนล 1 และส่งข้อมูลการเล่นคอร์ดออกไปทางแชนแนล 2 ให้กับซินธิไซเซอร์ 1

สำหรับซินธิไซเซอร์บางเครื่องสามารถทำงานแยกกันได้หลายแชนแนลในเวลาเดียวกันได้อย่างอิสระ (multi channel) (แต่อาจจำกัดจำนวนเสียงโพลีโฟนิคที่ผลิตได้ หรือบางแชนแนล อาจเป็นโมโนโฟนิค) ดังนั้นถ้าซินธิไซเซอร์ 2 ทำงานแบบมัลติแชนแนล เราอาจส่งข้อมูลการเล่นเบสไปให้ ซินธิไซเซอร์ 2 ทางแชนแนล 3 และส่งทำนองหลัก (melody line) ไปทางแชนแนล 4 ของซินธิไซเซอร์ 2 ได้

3.2 การสร้าง

สำหรับโครงงานนี้จะใช้คอมพิวเตอร์ IBM PC-XT ทำหน้าที่เป็นซีเควนเซอร์ที่ทำงานในระบบมีดี้โดยมีความสามารถในการทำงานดังรูป 3.2

<u>รายละเอียดของ Main Menu</u>

1. **File** : เป็นการทำงานเกี่ยวกับแฟ้มข้อมูล ประกอบด้วย

    1.1 **Load** ใช้เรียกไฟล์จากดิสค์

    1.2 **Save** ใช้เก็บข้อมูลการเล่นลงดิสค์

    1.3 **Rename** สำหรับเปลี่ยนชื่อไฟล์

    1.4 **Change Dir** สามารถเปลี่ยน directory ได้ตามต้องการ

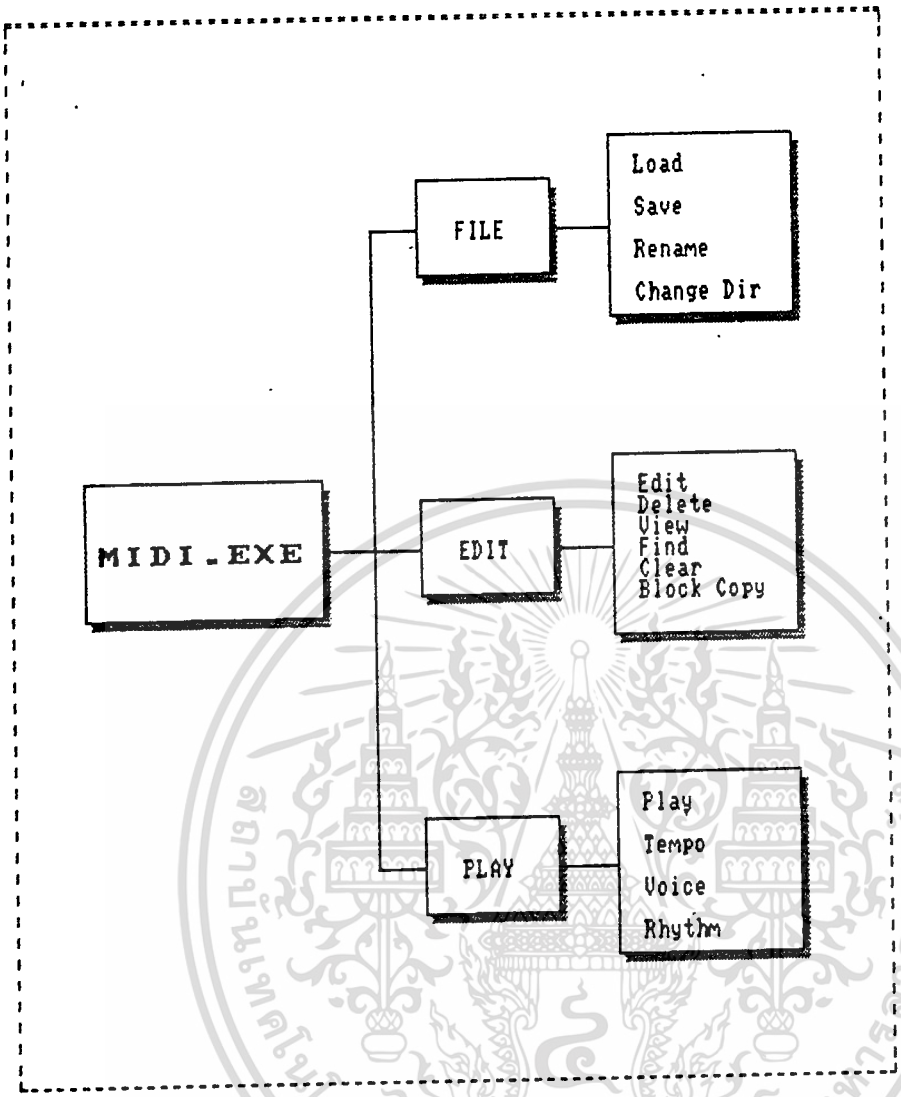
2. **Edit** : ทำงานเกี่ยวกับการป้อนข้อมูล ประกอบด้วย

    2.1 **Edit** เป็นการป้อนข้อมูลการเล่น เช่น โน๊ตที่จะเล่น, ความยาวของโน๊ต (durration เช่น ตัวกลม ตัวดำ ขเบ็จ), ความดัง, ความเร็วคีย์, การยืดเสียง (sustain), โปรแกรมเสียง, แชนแนลที่จะให้ส่ง และเวลาที่จะให้ส่งคำสั่งต่างๆ (ว่าจะให้ส่งเมื่อไหร่) ในกรณีที่คำสั่งเป็นการสั่งให้เล่นโน๊ต (note on) คอมพิวเตอร์จะใส่คำสั่งเลิกเล่นโน๊ต (note off) ให้เองโดยอัตโนมัติโดยคำนวณเวลาที่จะให้เลิกเล่นโน๊ตจากความยาว ของโน๊ตตัวนั้น

    2.2 **Delete** ใช้ลบข้อมูลที่ไม่ต้องการ ในกรณีที่ลบคำสั่งการเล่นโน๊ต คอมพิวเตอร์ก็จะลบคำสั่งการเลิกเล่นโน๊ตของโน๊ตตัวนั้นให้ด้วย (ถ้าพบ)

    2.3 **View** ใช้ขอดูข้อมูลการเล่นที่เขียนไปทั้งหมด

    2.4 **Find** ใช้เรียกดูข้อมูลจากเวลาอ้างอิงที่กำหนด (ในการ edit จะต้อง



รูป 3.2 การทำงานของซอฟท์แวร์ที่เขียนขึ้น

กำหนดคำสั่ง, เวลาที่จะให้คอมพิวเตอร์ส่งคำสั่งนั้นไปให้เครื่องดนตรีเสมอ)

2.5 **Clear** เป็นการลบข้อมูลทั้งหมด

2.6 **Block Copy** ในกรณีที่จะต้องเขียนข้อมูลที่ซ้ำกันสามารถลอกข้อมูลส่วนนั้นไปใช้ได้เลยไม่ต้องเสียเวลาเขียนซ้ำ

3. **Play** : เป็นฟังก์ชันสำหรับการเล่นดนตรี ประกอบด้วย

3.1 **Play** ข้อมูลในการเล่นต่างๆที่ edit ไว้จะถูกส่งออกไปตามมาตรฐานของระบบมีดี้ไปยังเครื่องดนตรี ซึ่งเครื่องดนตรีก็จะทำงานตามที่ edit ไว้

3.2 **Tempo** เป็นการกำหนดความเร็วในการเล่น (tempo) มีค่าได้ตั้งแต่ 20-200 โดย tempo = 20 จะเป็นจังหวะที่ช้าที่สุด

3.3 **Voice** เป็นการกำหนดโปรแกรมเสียงดนตรีที่จะเล่น เช่นถ้าในซินธิไซเซอร์ ที่จะเล่น ตั้งโปรแกรม 1 ไว้เป็นเสียงเปียโน ถ้าเราเลือกโปรแกรม 1 เราก็จะได้เสียงเปียโนออกมา (เป็นผลมาจากการที่คอมพิวเตอร์ส่งข้อความของการเลือกโปรแกรมเสียงที่ 1 ออกไป) ซึ่งการเลือกโปรแกรมเสียงที่จะเล่นสามารถกำหนดไว้ในขณะที่ edit ได้เช่นกัน

3.4 **Rhythm** จากที่ได้กล่าวไปแล้วเกี่ยวกับการทำให้แต่ละอุปกรณ์ทำงานในจังหวะเดียวกัน (synchronized) เช่น ในการสร้างจังหวะของกลองอิเลคโทรนิคจะอาศัยสัญญาณนาฬิกาที่ส่งไปด้วยอัตรา 24 ลูกต่อโน๊ตตัวดำหนึ่งตัว (24 pulses per quaternote: 24 ppqn) เป็นจังหวะอ้างอิง ในกรณีที่ Rhythm On คอมพิวเตอร์จะส่งสัญญาณนาฬิกาออกไปด้วย และถ้า Rhythm Off จะไม่มีการส่งสัญญาณนาฬิกา ออกไป การกระทำต่างๆที่ต้องอาศัยสัญญาณนาฬิกา(มักเป็นเรื่องของการสร้างจังหวะ) ก็จะไม่เกิดขึ้น

อนึ่งหน้าที่ของซีเควนเซอร์คือช่วยเล่นดนตรีแทนคน **การสร้างคุณภาพของเสียง เช่นการกำหนดรูปร่างของคลื่นเสียง (wave form) ในซินธิไซเซอร์เป็นเรื่องของอุปกรณ์ประเภทวอยส์เอดิเตอร์ (voice editor) หรือ patch libraly ไม่ใช่หน้าที่ของซีเควนเซอร์** เนื่องจากซินธิไซเซอร์ของแต่ละผู้ผลิตจะมีวิธีในการสร้าง

คุณภาพของเสียงต่างกัน เช่นซินธิไซเซอร์ของยามาฮ่า (YAMAHA) จะใช้หลักของ frequency modulation ในการสร้างฮาร์โมนิค (harmonic) ต่างๆ ส่วนคาสิโอ (CASIO) จะใช้วิธีที่เรียกว่า phase distortion ดังนั้นวอยส์เอดิเตอร์จะผลิตแยกกันตามยี่ห้อของซินธิไซเซอร์ เช่นวอยส์เอดิเตอร์ของยามาฮ่าก็จะใช้ได้กับซินธิไซเซอร์ของยามาฮ่าเท่านั้น (ในตระกูลDXเช่นDX7) ส่วนวอยส์เอดิเตอร์ของคาสิโอก็จะใช้กับซินธิไซเซอร์ของคาสิโอเท่านั้น (ในตระกูล CZ หรือ VZ) ซึ่งก็มักจะเป็นการรวบรวมเสียงสำเร็จรูปที่นักดนตรีเรียกมาใช้ได้ง่ายๆ ข้อความในระบบมีดี้ที่ใช้ก็มักเป็นพวกข้อความพิเศษเกี่ยวกับระบบ

3.2.1 ฮาร์ดแวร์

ในส่วนของมีดี้อินเตอร์เฟสการ์ด (MIDI Interface card) มีวงจรเป็นดังรูป3.3 หัวใจสำคัญของวงจรอยู่ที่ไอซีเบอร์ 8251 ซึ่งเป็น USART ทำหน้าที่เปลี่ยนข้อมูลแบบขนานจากคอมพิวเตอร์ให้เป็นแบบอนุกรมตามข้อกำหนดของมีดี้ และแบบอนุกรมเป็นขนาน ในการใช้งานจะใช้โหมดอซิงโครนัสโดยมีบิทเริ่มและบิทหยุดอย่างละ 1 บิท ความเร็วในการส่งและรับ 31.25 kBd. ซึ่งได้มาจากสัญญาณนาฬิกา 2 MHz หารด้วย 64 (64x Baud rate factor)

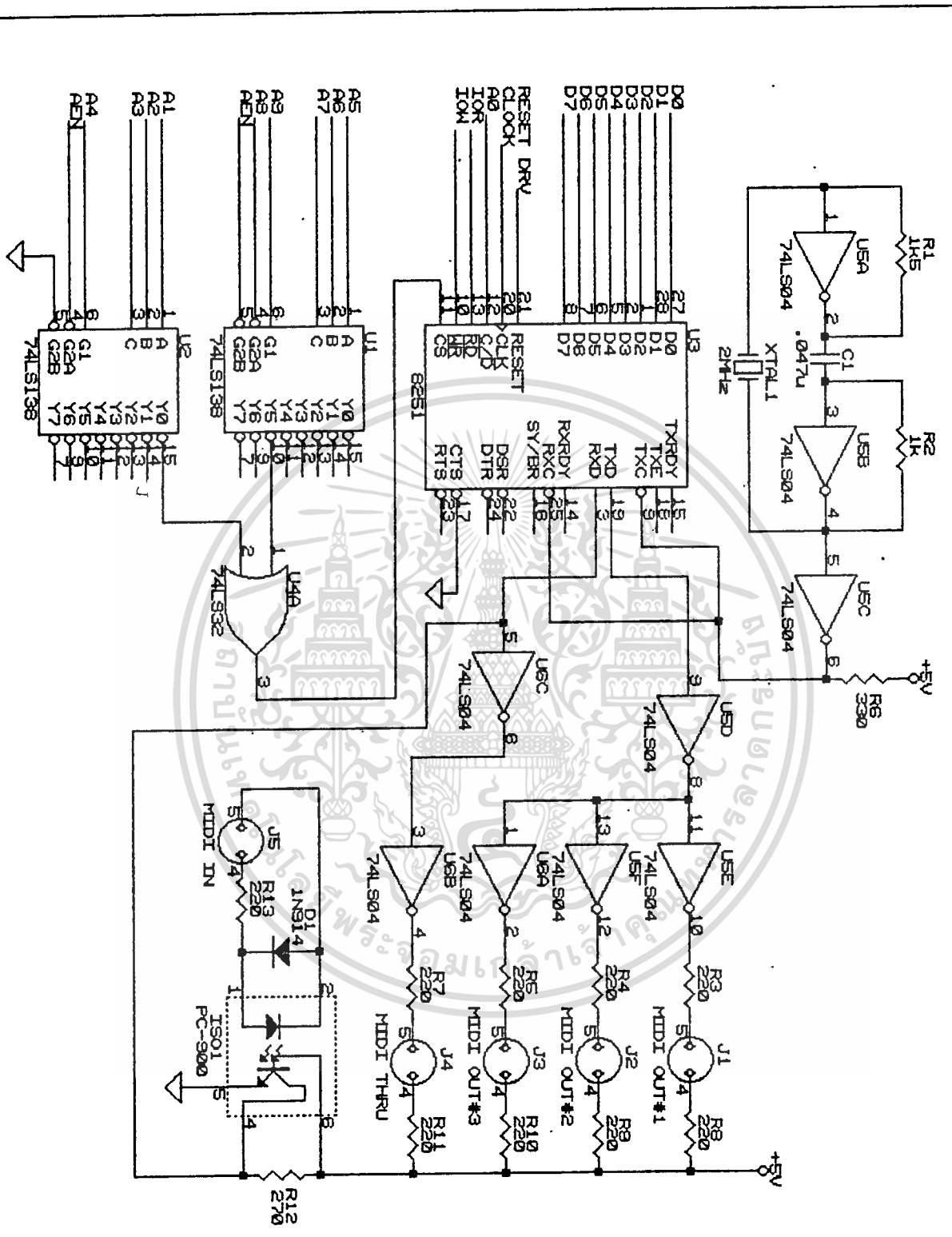
74LS138 เป็นตัวถอดรหัสแอดเดรส (decoder) ซึ่งแอดเดรส (address) ของ 8251 สำหรับพอร์ทข้อมูล(data port)คือ 2B0H และพอร์ทควบคุม(control port) คือ 2B1H

สัญญาณนาฬิกาของระบบ(system clock)เป็นสัญญาณนาฬิกาของคอมพิวเตอร์ คือ 4.77 MHz

อุปกรณ์ประเภทคอนเนคเตอร์ต่างๆใช้ตามที่มาตรฐานทางฮาร์ดแวร์ของมีดี้กำหนด ซึ่งได้กล่าวไปแล้ว

3.2.2 ซอฟท์แวร์

เขียนด้วยภาษาซีร่วมกับเทอร์โบซีทูล โดยใช้เทอร์โบซีเป็นคอมไพเลอร์ โดยแยกออกเป็นโมดูลต่างๆแล้วนำมาลิงค์ (link) กันภายหลังเนื่องจากโปรแกรมยาวมาก โมดูล



รูป 3.3 วงจรมิดี้อินเตอร์เฟสการ์ด

ต่างๆมีดังนี้

1. MIDI.C เป็นส่วนของโปรแกรมหลัก ประกอบด้วยการให้ค่าเริ่มต้นแก่ 8251 (initialize 8251), main menu, การเก็บภาพจากหน้าจอก่อนเข้ามายังโปรแกรม MIDI และคืนภาพหน้าจอนั้นให้หลังจากออกจากโปรแกรม, การให้ค่าเริ่มต้นแก่ตัวแปรต่างๆ และสร้างกราฟิคหน้าจอ โดยจะเรียกไฟล์ชื่อMDCONFIG.SYS แล้วนำค่าต่างๆจากไฟล์นี้ไปใส่ในวีดีโอแรม (video ram) วิธีนี้จะทำกราฟิคได้เร็วมาก (เร็วกว่าผ่านไบออส)

2. MENU.C เป็นส่วนที่ทำหน้าที่สร้างเมนู

3. WINDOW.C เป็นส่วนที่ทำหน้าที่สร้างวินโดว์

โปรแกรมส่วนต่อไปนี้จะถูกเรียกใช้ทาง main menu

4. FILE.C ทำหน้าที่เกี่ยวกับดิสค์

5. EDIT.C ทำหน้าที่เกี่ยวกับการเขียนข้อมูลในการเล่น

6. PLAY.C ทำหน้าที่เกี่ยวกับการเล่น (ส่งข้อมูลไปให้เครื่องดนตรี)

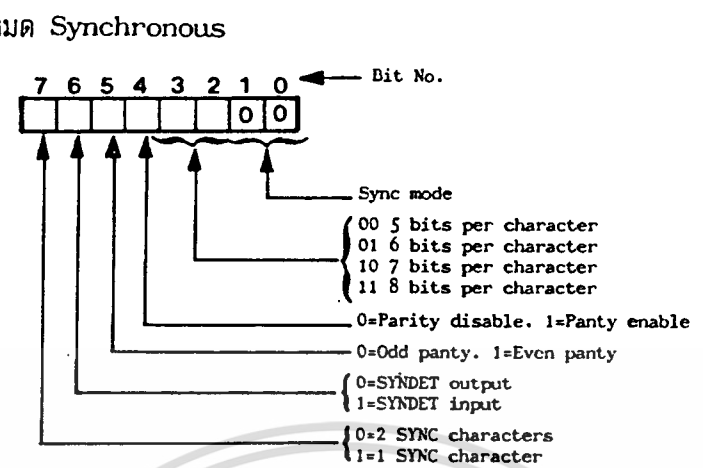
7. VOICE.C ทำหน้าที่เกี่ยวกับโปรแกรมเสียงที่จะเล่น

การโปรแกรม 8251

จะส่งคำสั่งเลือกโหมด (control word หรือ mode word) ไปให้รีจิสเตอร์ควบคุมของ 8251 ก่อนเพื่อเลือกโหมดที่จะใช้ โดยส่งไปที่พอร์ทควบคุม ซึ่งมีแอดเดรสที่ 2B1H จากนั้นจึงส่งคำสั่งควบคุม (command word) ไปให้พอร์ทควบคุม

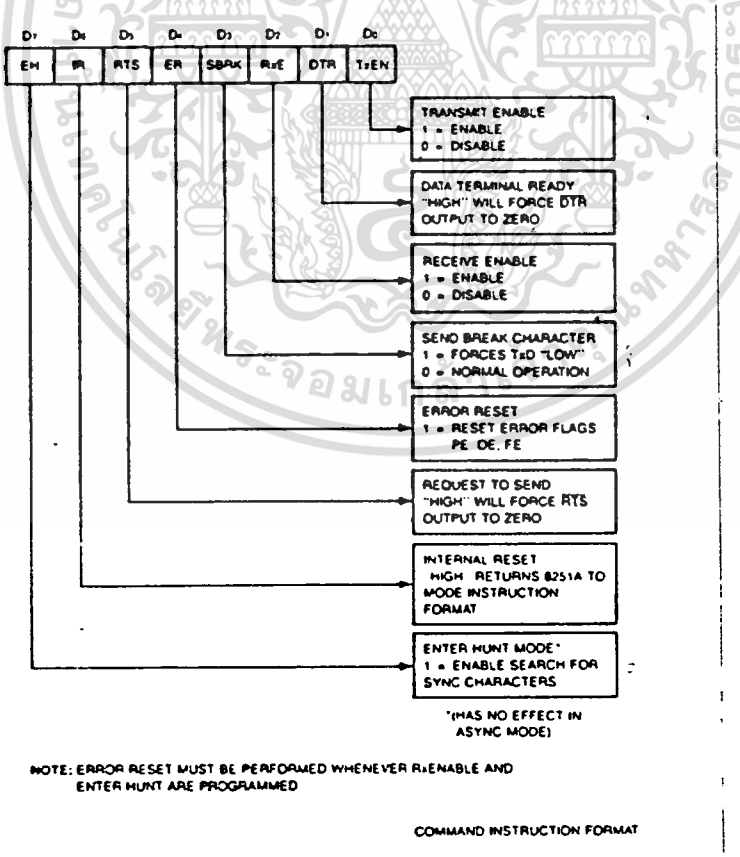
อนึ่ง ในโหมดอซิงโครนัสนั้น หลังจากที่ได้ส่งคำสั่งเลือกโหมดให้ 8251 แล้ว ข้อมูลไบท์ต่อไปที่ถูกส่งให้กับรีจิสเตอร์ควบคุมจะถือว่าเป็นคำสั่งควบคุมทั้งสิ้น จนกว่า 8251 จะได้รับการรีเซ็ทอีกจึงจะถือว่าข้อมูลที่ส่งมาให้กับรีจิสเตอร์ควบคุมนั้นเป็นคำสั่งในการเลือกโหมด

การจัดเรียงบิทในโหมดเวิร์ดเป็นดังรูป 3.4 ค่าที่จะส่งไปเป็นโหมดเวิร์ดสำหรับโครงงานนี้คือ 0CFH นั่นคือโหมดอซิงโครนัส, 64x Baud rate factor, 8 บิทข้อมูล, ไม่มีบิทพาริตี้ (parity) และบิทหยุด 2 บิท โดยส่งไปยังพอร์ท 2B1



รูป 3.4 การจัดเรียงบิทในโหมดเวิร์ด

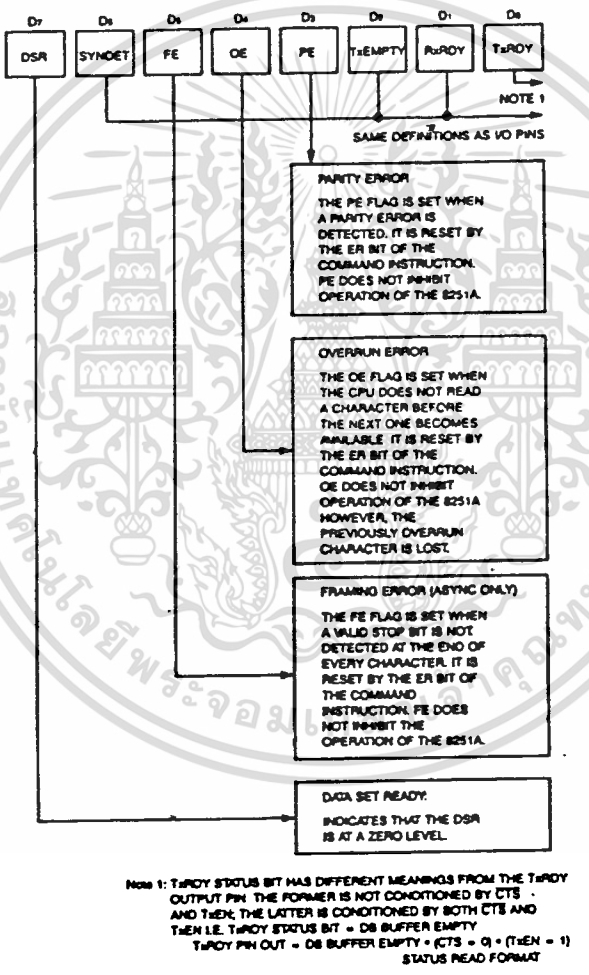
การเรียงบิทในคอมมานด์เวิร์ดเป็นดังรูป 3.5 ค่าที่ส่งไปคือ 15H ซึ่งเป็นการ



รูป 3.5 การจัดเรียงบิทในคอมมานด์เวิร์ด

เอนาเบิล (enable) การส่ง-รับ และรีเซ็ทแฟลกแสดงข้อผิดพลาด(error flags) โดยส่งไปให้พอร์ท 2B1 เช่นกัน ในขณะนี้ 8251 จะสามารถทำงานรับ-ส่งข้อมูลได้แล้ว

ในการส่งข้อมูล คอมพิวเตอร์จะต้องตรวจดูรีจิสเตอร์สถานะของ 8251 เสีย ก่อนว่าพร้อมที่จะรับหรือส่งข้อมูลหรือยัง รีจิสเตอร์สถานะของ 8251 เป็นดังรูป 3.6



รูป 3.6 การจัดเรียงบิทบนรีจิสเตอร์สถานะ

ในกรณีที่คอมพิวเตอร์ส่งข้อมูลออกไปให้เครื่องดนตรีจะต้องมีการตรวจดูบิท D0 (LSB) ของรีจิสเตอร์สถานะเสียก่อน ถ้าบิทนี้เป็น 1 จะหมายถึง Tx Ready หมายความว่า 8251 พร้อมที่จะรับข้อมูลจากคอมพิวเตอร์เพื่อทำการส่งออกไปได้ และถ้าบิทนี้เป็น 0 หมายความว่า 8251 ยังไม่พร้อมที่จะส่ง (Tx buffer ยังไม่ว่าง) **ดังนั้นคอมพิวเตอร์จะรอจนกว่าบิท D0 ของรีจิสเตอร์สถานะมีค่าเป็น 1 จึงจะส่งข้อมูลไปให้ 8251**

ในกรณีที่รับข้อมูลจากเครื่องดนตรี จะต้องตรวจดูบิท D1 ของรีจิสเตอร์สถานะ ถ้าบิทนี้เป็น 1 จะหมายถึง Rx Ready นั่นคือ 8251 รับข้อมูลจากเครื่องดนตรีไว้เรียบร้อยพร้อมที่จะส่งให้กับคอมพิวเตอร์แล้ว และถ้าบิทนี้เป็น 0 แสดงว่า 8251 ยังไม่ได้รับข้อมูลจากเครื่องดนตรี **ดังนั้นคอมพิวเตอร์ต้องรอจนกว่า บิท D1 ของรีจิสเตอร์สถานะมีค่าเป็น 1 เสียก่อนจึงจะอ่านข้อมูลจาก 8251**

จะอ่านค่าของรีจิสเตอร์สถานะได้จากพอร์ทควบคุม (2B1) และการรับ-ส่งข้อมูลจะผ่านทางพอร์ทข้อมูล (2B0)

การโปรแกรม 8251 เขียนเป็นโฟลว์ชาร์ตได้ดังรูป 3.6

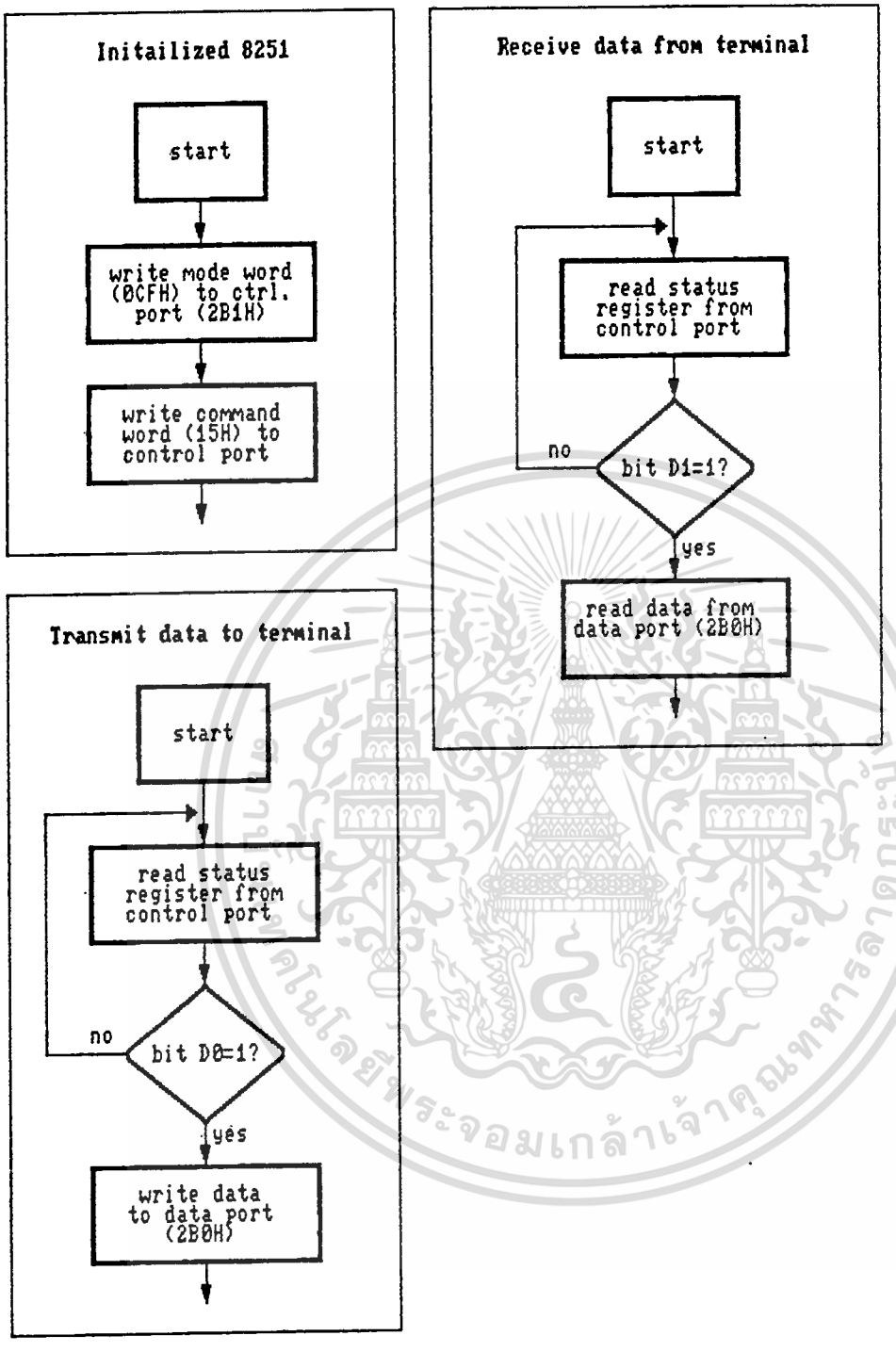
การเก็บข้อมูล (ในการอีดิต)

เก็บแบบลิงค์ลิสท์ (Link list) โดยข้อมูลแต่ละเรคอร์ด(record)(ภาษาซีเรียก structure) จะประกอบด้วย

Time : เป็นข้อมูลประเภทจำนวนเต็ม (unsigned integer) ซึ่งจะบอกถึงเวลาที่จะให้เครื่องดนตรีทำงานตามคำสั่งต่างๆ หรือเวลาที่จะให้คอมพิวเตอร์ส่งคำสั่งในระบบมีดี้ไปให้เครื่องดนตรีนั่นเอง เช่น time=0 ส่งโน๊ตโดออกไป, time=24 ส่งโน๊ตซอล และ โน๊ตมี ออกไป สำหรับ time นั้นจะมีอัตรา = 24 ต่อโน๊ตตัวดำหนึ่งตัว

Even : เป็นข้อมูลประเภทสตริง (string) มีความยาว 5 ไบท์ evenจะบอกถึงคำสั่งต่างๆที่จะให้เครื่องดนตรีทำเมื่อถึงเวลาที่กำหนด คำสั่งที่ใช้มีดังนี้

1.การบอกให้เล่นโน๊ต บอกเป็นตัวอักษร และ ออคเตฟของโน๊ตตัวนั้น เช่น c2, G3, a4 ในกรณีที่ป้อนเข้าไปเป็นอักษรตัวเล็ก คอมพิวเตอร์จะปรับให้เป็นอักษร



รูป 3.6 การโปรแกรม 8251

ตัวใหญ่เพื่อความสวยงามสำหรับโน๊ตประเภท ชาร์พ(#), แฟลต(b) การเขียนให้เขียนเป็นเครื่องหมายชาร์พ (#) เช่น C2#, F3# ห้ามเขียนในลักษณะของแฟลต เช่น C2# ห้ามเขียนเป็น D2b เพราะการเขียนแบบนี้จะไปสับสนกับสนกับตัวบี (b) และจะต้องเขียนโปรแกรมยาวขึ้นถ้าต้องการให้เขียนได้ทั้งสองแบบ

2.PRO : เป็นคำสั่งที่ใช้เลือกโปรแกรมเสียงที่จะเล่น (program select)

3.SUS : เป็นลักษณะของการเหยียบ sustain paddle ของเปียโน

4.VOL : กำหนดความดังของเสียง (ไม่ใช่ความเร็วคีย์ ในที่นี้คือ volume)

Data : เป็นข้อมูลของ even เช่นในกรณีของการเล่นโน๊ต data ก็จะเป็นความเร็วคีย์ในกรณีของ PRO จะมี data เป็นโปรแกรมเสียงที่จะเล่น เป็นต้น (เป็น unsigned int.)

Durr : เป็นข้อมูลสำหรับการเล่นโน๊ต นั่นคือเป็นค่าความยาวของโน๊ตโดยจะบอกเป็นตัวเลข durration = 24 จะหมายถึงโน๊ตตัวดำ (เป็น unsigned int.)

Chann : บอกให้รู้ถึงแชนแนลที่จะส่ง (0-15) เป็น int.

นอกจากนี้จะมีข้อมูลประเภทพอยน์เตอร์ (pointer) อีกสองตัว เอาไว้ลิงค์กับข้อมูลในเรคอร์ดอื่นๆที่อยู่ติดกัน ข้อมูลแต่ละเรคอร์ดจะถูกเรียงตามเวลาจากน้อยไปมาก (ก่อนไปหลัง)

ดังนั้นข้อมูล 1 เรคอร์ดจะมี 1 คำสั่ง (even) ซึ่งจะใช้หน่วยความจำทั้งสิ้น 13 ไบท์ (การเก็บข้อมูลประเภท integer ในภาษาซีใช้ 2 ไบท์)

การเล่น

เป็นการนำข้อมูลที่อีดิตไว้มาตีความแล้วส่งออกไปให้เครื่องดนตรีในรูปแบบของมีดี้ เนื่องจากข้อมูลจากการอีดิตจะถูกเรียงไว้แล้วตามเวลาที่จะให้เล่น ดังนั้นเรคอร์ดแรกจะเป็นเหตุการณ์ที่เกิดขึ้นครั้งแรก จนถึงเรคอร์ดสุดท้ายก็จะเป็นเหตุการณ์สุดท้าย

การที่จะให้ส่งข้อมูลได้ถูกต้องตามเวลาที่กำหนดจำเป็นต้องมีจังหวะ (timer) เอาไว้อ้างอิงซึ่งเป็นฮาร์แวร์สำหรับโปรเจ็คนี้จะใช้ 8253 ซึ่งเป็น Programmable Interval Timer ที่มีอยู่แล้วในไอบีเอ็มโดยใช้สัญญาณคล๊อคจากแชนแนล 0

8253 มีอยู่ด้วยกัน 3 เคาน์เตอร์ แต่ละเคาน์เตอร์มีขนาด 16 บิท ในไอบีเอ็ม คล๊อคที่ป้อนให้ 8253 (อินพุทคล๊อคของทุกแชนแนล) จะใช้คล๊อคของระบบหารด้วย 4 นั่นคือ 4.772727/4 =1.1931817 MHz เอาท์พุทที่ได้จากแชนแนล 0 จะใช้ในการกำหนดวันเวลา (time-of-day clock tick) ส่วนเอาท์พุทจากแชนแนล 1 จะส่งไปยัง DMA controller เพื่อใช้ในการรีเฟรชไดนามิคแรม (dynamic ram) และเอาท์พุทจากแชนแนล 2 จะใช้เป็นความถี่ที่จะส่งไปให้ลำโพงเพื่อสร้างเสียง

แอดเดรสของเคาน์เตอร์ 0, 1, 2 คือ 40H, 41H, 42H ตามลำดับ

และแอดเดรสของพอร์ทควบคุมคือ 43H

สัญญาณคล๊อคจากแชนแนล 0 จะส่งไปอินเทอร์รัพท์ฮาร์ดแวร์ทาง IRQ0 (int8) เพื่อใช้เป็นนาฬิกาของระบบ ซึ่งเราสามารถอ่านค่าเวลาออกมาดูได้ด้วยการอินเทอร์รัพท์ซอฟท์แวร์ (int 21H) ฟังก์ชัน 44 (2CH) จากที่กล่าวมาเราสามารถใช้ 8253 ที่มีอยู่แล้วในไอบีเอ็มเป็นไทม์เมอร์ได้และเราสามารถกำหนดเวลาของไทม์เมอร์ว่าจะให้เดินเร็วหรือช้าได้อีกด้วยโดยการโปรแกรม 8253 แชนแนล 0 เสียใหม่ให้เป็นไปตามที่เราต้องการ

การโปรแกรม 8253

ทำได้โดยขั้นแรกเป็นการเลือกโหมดการทำงาน โดยการเขียนข้อมูลเข้าไปในรีจิสเตอร์ควบคุม ซึ่งมีรูปแบบของคอนโทรลเวิร์ดดังนี้

CONTROL WORD D7-D0

D7	D6	D5	D4	D3	D2	D1	D0
SC1	SC0	RL1	RL0	M2	M1	M0	BCP

SC1 กับ SC0 หมายถึงการเลือกเคาน์เตอร์ที่จะโปรแกรม ในกรณีนี้คือ 00 (เลือกเคาน์เตอร์ 0)

RL1,RL0 คือลักษณะการเขียนข้อมูลเข้าไปให้เคาน์เตอร์ในกรณีนี้คือ11(read/load ไบท์ที่มีนัยสำคัญต่ำก่อน เสร็จแล้วตามด้วยไบท์ที่มีนัยสำคัญสูง)

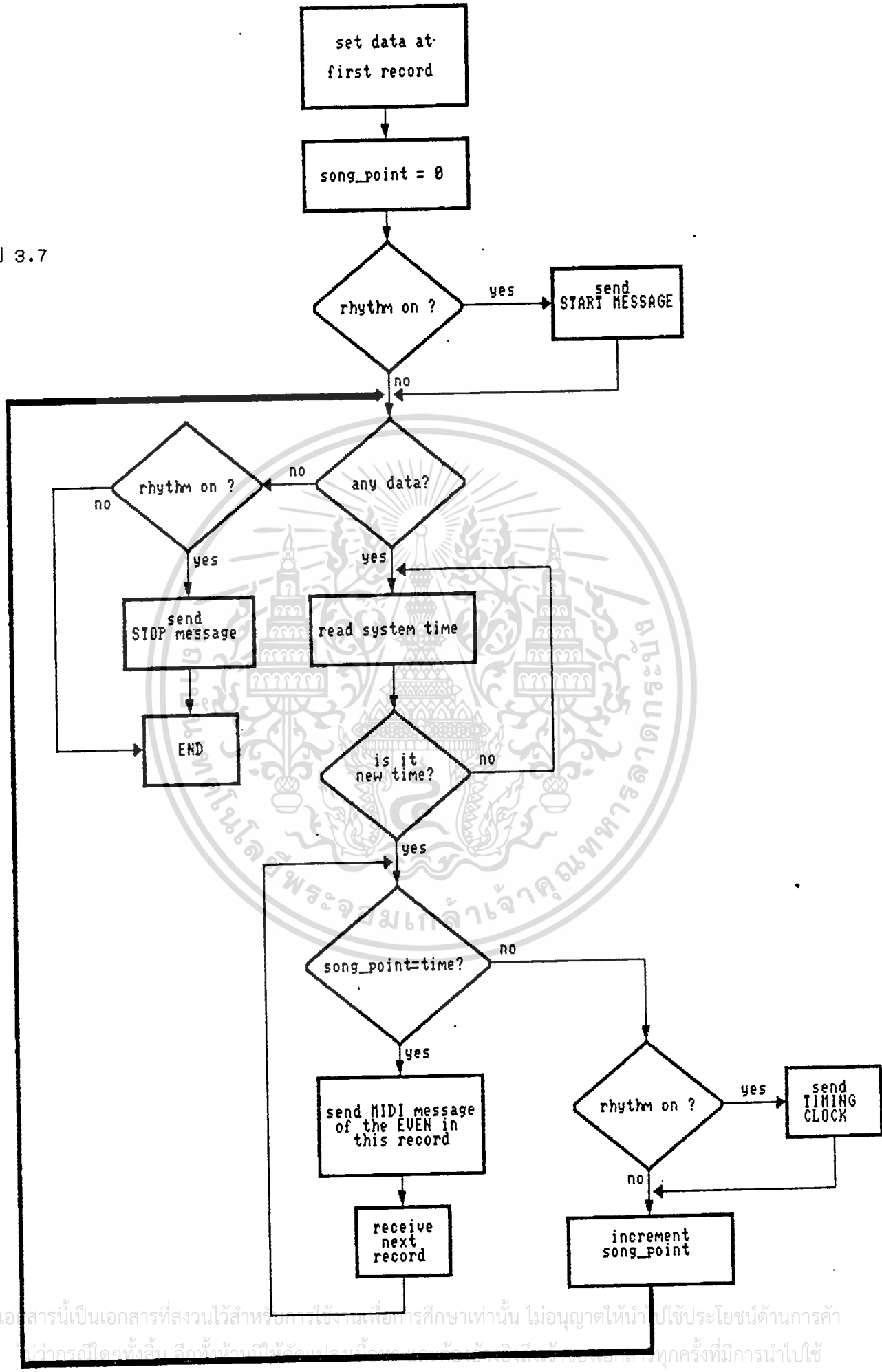
D3, D2, D1 เป็นการเลือกโหมดการทำงาน ในกรณีนี้คือ 011 (โหมด 3)

DO ใช้กำหนดลักษณะการนับของเคาน์เตอร์ ถ้า DO=1 จะนับแบบ BCD และถ้า DO=0 จะนับแบบไบนารี่ ในกรณีนี้ใช้ DO=0

ดังนั้นในการโปรแกรม 8253 แชนแนล 0 เราจะส่งค่า 36H ซึ่งเป็นคอนโทรลเวิร์ดไปยังพอร์ทควบคุม (43H) ค่าต่อไปจะส่งไปให้เคาน์เตอร์ 0 (พอร์ท 40H) เพื่อเป็นตัวหาร ซึ่งความถี่ของเอาท์พุทก็ได้มาจากอินพุทคล็อค (1.193 MHz) หารด้วยตัวหาร (16 บิท) นี่เอง โดยจะส่งไบท์ที่มีนัยสำคัญต่ำไปก่อน ตามด้วยไบท์ที่มีนัยสำคัญสูง

ในการเริ่มเล่นจะต้องจัดให้ข้อมูลอยู่ที่ตำแหน่งเรคอร์ดแรกเสียก่อน เนื่องจากแต่ละเรคอร์ดจะมีค่าของเวลาที่ให้ทำคำสั่งในเรคอร์ดนั้นอยู่ (เวลาอาจซ้ำกันได้นั่นคือเป็นเหตุการณ์ที่เกิดขึ้นพร้อมกันนั่นเอง) เรามีไทม์เมอร์ที่มีเวลาอ้างอิงเดินไปเรื่อยๆ โดยใช้ตัวแปรที่ชื่อ song_point ซึ่งจะมีค่าเปลี่ยนไปเรื่อยๆตามไทม์เมอร์ ในการอีดิตเราจะให้เวลาแรก = 0 ในการเริ่มเล่นเพลง song_point แรกจะมีค่าเท่ากับ 0 หลังจากที่เวลาผ่านไป song_point ก็จะเพิ่มขึ้นเรื่อยๆ (เพิ่มทีละหนึ่ง) คอมพิวเตอร์จะตรวจดูว่าขณะนั้น song_point มีค่าเท่ากับเวลาของเรคอร์ด(เวลาที่กำหนดไว้ในตอนอีดิต) หรือไม่ ถ้าเวลาตรงกันเมื่อไหร่คอมพิวเตอร์จะนำคำสั่งที่อยู่ในเรคอร์ดนั้นไปตีความ (นำคำสั่งไปเทียบกับตาราง) แล้วทำงานตามคำสั่งนั้น วิธีที่จะนำคำสั่งที่อีดิตไว้มาตีความ แล้วส่งออกไปด้วยเวลาที่ถูกต้อง แสดงได้ด้วยโฟล์วชาร์ทดังรูป 3.7

รูป 3.7



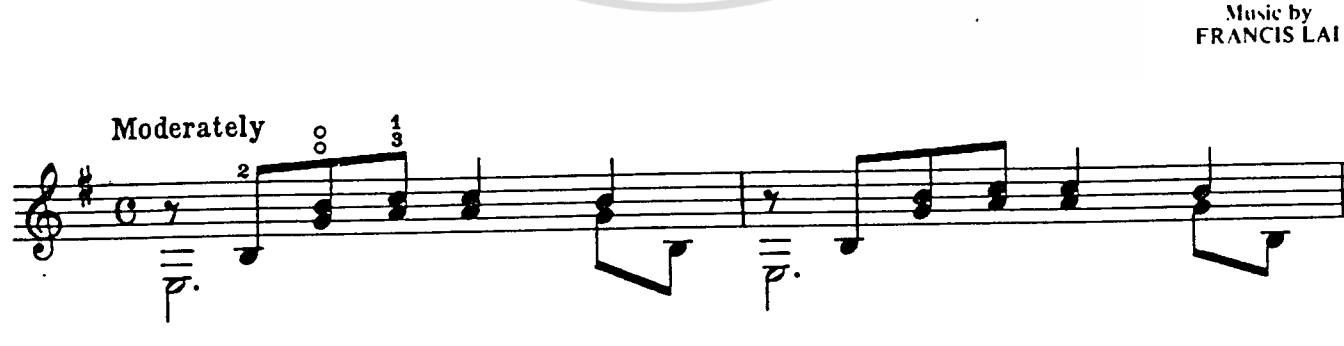
บทที่ 4

การทดลองและผลการทดลอง

ได้ทดลองป้อนโปรแกรมเพลงให้คอมพิวเตอร์ แล้วให้คอมพิวเตอร์เปลี่ยนข้อมูลเหล่านี้ให้เป็นไปตามมาตรฐานของระบบมีดี้ส่งไปยังเครื่องดนตรี พบว่า สามารถทำการส่งข้อมูลออกไปได้อย่างถูกต้อง คอมพิวเตอร์สามารถควบคุมการทำงานของเครื่องดนตรีได้ทั้ง 16 แชนแนล รวมทั้งยังสามารถรับข้อมูลจากเครื่องดนตรีเข้ามาได้อีกด้วย แต่ในการรับพบว่ามีปัญหาอยู่บ้าง เช่น เมื่อมีข้อมูลเข้ามามากๆและเร็วๆ บางครั้งเครื่องดนตรีส่งข้อมูลใหม่เข้ามาในขณะที่คอมพิวเตอร์ยังไม่ได้รับข้อมูลเก่าจาก 8251 เข้าไปจึงเกิดข้อผิดพลาด (over run error) แต่ปัญหาที่สำคัญของไอซีเบอร์ 8251 คือ เมื่อเพิ่มคล๊อคของระบบให้สูงขึ้น (เทอร์โบ) เช่น เพิ่มจาก 4.77 MHz เป็น 8 MHz จะไม่สามารถสื่อสารกันได้เลย

ข้อมูลต่างๆที่ป้อนให้กับคอมพิวเตอร์สามารถเก็บลงดิสค์และเรียกข้อมูลจากดิสค์เข้ามาใช้ได้ การใช้งานค่อนข้างง่ายเนื่องจากมีเมนู ผู้ใช้สามารถสั่งงานได้โดยการกดปุ่มเลือกการทำงานจากเมนูต่างๆ

ตัวอย่างการโปรแกรมเพลงเป็นดังนี้ (กรณีนี้ใช้เพียงแชนแนลเดียว)



ข้อมูลที่ผู้ใช้ต้องป้อนเข้าไปคือ (เฉพาะห้องแรก)

Time	Even	Data	Durr	Chan
0	E2	64	60	0
12	B2	64	12	0
24	G3	64	12	0
24	B3	64	12	0
36	A3	64	12	0
36	C4	64	12	0
48	A3	64	24	0
48	C4	64	24	0
72	G3	64	12	0
72	B3	64	12	0
84	B2	64	12	0

สำหรับโน๊ตในห้องที่ 2 สามารถใช้วิธีก๊อปปี้ข้อมูลของห้องแรกได้เลย (block copy) โดยเลื่อนเวลาให้ห้องที่สองไปตั้งต้นที่ time = 96 ทั้งหมดนี้คอมพิวเตอร์จะใส่คำสั่งให้เลิกเล่นโน๊ต (เป็นคำสั่งให้เล่นโน๊ตโดยที่ data = 0) ให้โดยอัตโนมัติ

บทที่ 5

บทวิจารณ์และสรุป

ปริญญานิพนธ์นี้เป็นเรื่องของการสื่อสารข้อมูลระหว่างคอมพิวเตอร์กับเครื่องดนตรีในระบบมิดี้ ซึ่งเป็นระบบที่กำลังได้รับความนิยมเป็นอย่างมาก กล่าวได้ว่าเครื่องดนตรีระบบดิจิตอลในสมัยนี้จะตอบสนองระบบมิดี้แทบทุกเครื่อง ซึ่งในคอมพิวเตอร์ Commodore Amiga ได้มีส่วนของมิดี้อินเตอร์เฟสให้มาด้วยเลย

อย่างไรก็ตามจากการทดลองทำทางด้านฮาร์ดแวร์พบว่าไอซีเบอร์ 8251 ไม่เหมาะนักสำหรับงานประเภทนี้ (แต่มันก็สามารถทำงานได้) เพราะมันไม่ได้ถูกออกแบบมาสำหรับการสื่อสารแบบอซิงโครนัสความเร็วสูงๆ ยังมีไอซีเบอร์อื่นที่น่าสนใจ เช่น MC 6850 ของ Motorola ซึ่งสามารถสื่อสารได้ด้วยความเร็วถึง 1 Mbps ในแบบอซิงโครนัส

ภาคผนวก

รายละเอียดของโปรแกรมและรหัสมีดี้

โปรแกรม MIDI.C

```
/*      M A I N   P R O G R A M     */
#include <stdio.h>
#include <dos.h>
#include <bvideo.h>
#include <string.h>
#include <dir.h>
#define control_port 0x2B1

extern struct  _address  {
    int  time;
    char even[5];
    int  data;
    int  dur;
    int  channel;
    struct  _address  *next;
    struct  _address  *prior;
    }  list_entry;

extern struct  _address *start;
extern struct  _address *last;
extern int  Vrow, Vcol, row, _rhythm, tempo, def_ch;
extern char drive, song_name[28];
char Mbuffer[80][25][2];
int  Mwas_off, Mcursor_row, Mcursor_col, Mhigh, Mlow;

main()
{
    Mwas_off = sccurst(&Mcursor_row,&Mcursor_col,&Mhigh,&Mlow);
    virdrect(0,0,24,79,&Mbuffer[0][0][0],CHAR_ATTR);
    screen();
    start = last = NULL;
    strcpy(song_name,"no_name");
```

```
    def_ch = 0;
    _rhythm = 1;  tempo = 88;  Vrow=0;   Vcol=0;
    drive='A'+getdisk();
    set_8253(88);
    init8251();
    for(;;)  {
        intro();
        switch(menu3())
    {
      case  5: file();   break;
      case 14: edit();   break;
      case 23: play();   break;
      case 32: error(" RECORDING not ready ! Press <ESC>");  break;
      case 43: channel();  break;
      case 55: error("INSTRUMENT not ready ! Press <ESC>");  break;
      case 66:  quit();
        }
    }
}

/* initialize 8251 USART */
init8251()
{
  /* mode word */                  /* asyn. 64 x Bd.rate factor */
  outportb(control_port,0xCF);
  /* 8 data bit, 2 stop bit, no parity check */

  /* command word */
  outportb(control_port,0x15);

  outportb(control_port,0x55); /* command word   TO RESET 8251*/

  /* mode word */                  /* asyn. 64 x Bd.rate factor */
```

```
  outportb(control_port,0xCF);
  /* 8 data bit, 2 stop bit, no parity check */

  /* command word */
  outportb(control_port,0x15);
}

screen()
{
   FILE *f1;
   char buffer[80][25][2];
   int col,row,ch,ds;
   if ((f1 = fopen("mdconfig.sys","rb")) != NULL )
   {
   for(col=0;col<=79;col++)
     {  for(row=0;row<=24;row++)
        {  for(ch=0;ch<=1;ch++)
 if((buffer[col][row][ch] = getc(f1)) == EOF) goto XXX;
        }
     }   XXX:
   fclose(f1);
   CURSOR_OFF;
   ds = disk_free();
   viwrrect(0,0,24,79,&buffer[0][0][0],0,0,CHAR_ATTR);
   sccurset(7,71); printf("%5dK",ds);
   } else
     {
         viwrrect(0,0,24,79,&Mbuffer[0][0][0],0,0,CHAR_ATTR);
sccurset(Mcursor_row,Mcursor_col);
scpgcur(Mwas_off,Mhigh,Mlow,CUR_ADJUST);
vidspmsg(Mcursor_row,Mcursor_col,BLACK,15,
"Not found MDCONFIG.SYS ...program abort !...");
         exit(0);
```

```
        }
}

quit()
{
     outportb(control_port,0x55); /* command word    TO RESET 8251*/
     viwrrect(0,0,24,79,&Mbuffer[0][0][0],0,0,CHAR_ATTR);
     sccurset(Mcursor_row,Mcursor_col);
     scpgcur(Mwas_off,Mhigh,Mlow,CUR_ADJUST);
     exit(0);
}
```

โปรแกรม FILE.C

```
#include <stdio.h>
#include <stdlib.h>
#include <alloc.h>
#include <string.h>
#include <bvideo.h>
#include <bmenu.h>
#include <bfiles.h>
#include <bstrings.h>
#include <dir.h>
#include <dos.h>

extern  struct  _address  {
    int    time;
    char   even[5];
    int    data;
    int    dur;
    int    channel;
    struct  _address  *next;
    struct  _address  *prior;
    }  list_entry;

extern  struct  _address *start;
extern  struct  _address *last;
extern  struct  _address *info;
extern  char  drive;
char temp_name[28];

BMENU  *Lmenu;

char     song_name[28];
char     din[30][13];
int      count, Dnumber;
```

```
struct  dfree *df;

file()
{
     for(;;) {  intro();
       switch(menu1()) {
       case 0: load();        return;
       case 1: save();        return;
       case 2: re_name();     return;
       case 3: ch_dir();      return;
       case 4:                 break;
       case 5: return;
       }
      }
}

save()
{
  FILE *fp;
  if(!strcmp(song_name,"no_name"))  re_name();
  if(!strcmp(song_name,temp_name))  {
     error("Can't Save file <no_name> Press ESC");
     return; }
  if((fp=fopen(song_name,"wb"))==NULL)  {
    error(" Can't open file. Press <ESC>");
    return;
  }
  bwindow(22,1," Saving Editor File ");
  info=start;
  while(info)  {
    fwrite(info,sizeof(struct _address),1,fp);
    info=info->next;
  }
```

```
  fclose(fp);
  sccurset(7,71); printf("%5dK",disk_free());
  wnout();
}

load()
{
  struct _address *temp = NULL;
  FILE *fp;
  start = last = NULL;
  CURSOR_OFF;
  if(get_din())  {
  if(load_menu() == 110) return;
  if((fp=fopen(din[count],"rb"))==NULL)  {
    error(" Cant' open file. Press <ESC>");
    return;
  }
  while(start)  {
    info=start->next;
    free(info);
    start=info;
  }
  bwindow(22,1," Loading");
  start=(struct _address *) malloc(sizeof(struct _address));
  if(!start)  {
    error(" Out of memory. Press <ESC>");
    return;
  }
  info = start;
  while(!feof(fp))  {
   if(1 != fread(info,sizeof(struct _address),1,fp))  break;
   info->next=(struct _address *) malloc(sizeof(struct _address));
   if(!info->next) {
```

```
      error(" Out of memory. Press <ESC>");
      return;
    }
    info->prior=temp;
    temp=info;
    info=info->next;
  }
  temp->next=NULL;
  last=temp;
  start->prior = NULL;
  fclose(fp);
  wnout();
  strcpy(song_name,din[count]);
  drive = 'A'+getdisk();
  }else  error("Not Found \\*.DIN   Press <ESC>");
}

re_name()
{
    BWINDOW *pwin;
    BORDER   bord;
    WHERE    location;
    int  mode, columns, act_page;
    int  scan;
    strcpy(temp_name,song_name);
    pwin = wncreate(1,30,CYAN);
    if(pwin == NIL)  return;
    bord.type     = BBRD_SSSS | BBRD_TCT;
    bord.attr     = MAGENTA;
    bord.ttattr   = REVERSE;
    bord.pttitle = " Enter song name ";
    location.dev    = scmode(&mode,&columns,&act_page);
    location.page = act_page;
```

```
    location.corner.row = 6;
    location.corner.col = 23;
    wndsplay(pwin,&location,&bord);
    sccurset(6,24);  eget(song_name);
    stpcvt(song_name,RWHITE | TOUP);
    strcat(song_name,".DIN");
    if(strlen(song_name) < 5 || strlen(song_name) > 12)
      { if(strlen(song_name)>12)
 error(" Invalid song name.  Press <ESC>");
          strcpy(song_name,temp_name); } else drive = 'A'+getdisk();
    wnremove(pwin);
    wndstroy(pwin);
}

clear()
{
  start = last = NULL;
    while(start)  {
      info=start->next;
      free(info);
      start=info;
    }
}

error(message)
char message[34];
{
    char buffer[80][1][2];
    virdrect(22,0,13,79,&buffer[0][0][0],CHAR_ATTR);
    CURSOR_OFF;
    sccurset(22,44);
    scattrib(BLACK,15,' ',35);
    sccurset(22,44); printf(message);
```

```
    do { } while(getch() != 27);
    viwrrect(22,0,13,79,&buffer[0][0][0],0,0,CHAR_ATTR);
}

int get_din()
{
   struct ffblk ffblk;
   int   done;
   Dnumber=0;
   done = findfirst("\*.din",&ffblk,0);
   if(!done) {
    while(!done)  {
     strcpy(din[Dnumber],ffblk.ff_name);
     done=findnext(&ffblk); Dnumber++;
     if(Dnumber>30) return Dnumber;
       }
    }  return Dnumber;
}

int  load_menu()
{
   BORDER  bord;
   WHERE   location;
   int     mode, columns, act_page;
   int     cursor_was_off, cursor_row, cursor_col, high, low;
   int     lrow, lcol, ercode, ch, keycode;
   /*   char    first;  */
   Lmenu = mncreate(8,60,REVERSE,CYAN,NORMAL,NORMAL);
   if (Lmenu == NIL) exit(0);
    lrow=0; lcol=0;
     for(count=0;count<Dnumber;count++)  {
     /*   first=din[count][0]; */
     if (NIL == mnitmkey(Lmenu,lrow,lcol,MN_NOPROTECT,din[count],
```

```
"~",MN_SELECT)) exit(0);
      lcol += 15;    if(lcol>=60) { lcol=0; lrow++; }
      }
    bord.type = BBRD_SSSS | BBRD_TCT;
    bord.attr = REVERSE;
    bord.ttattr = REVERSE;
      switch(getdisk())  {
      case 0: bord.pttitle = " A:\\*.DIN "; break;
      case 1: bord.pttitle = " B:\\*.DIN "; break;
      case 2: bord.pttitle = " C:\\*.DIN "; break;
      case 3: bord.pttitle = " D:\\*.DIN "; break;
      }
    location.dev          = scmode(&mode,&columns,&act_page);
    location.page         = act_page;
    location.corner.row   = 7;
    location.corner.col   = 10;
    cursor_was_off = sccurst(&cursor_row,&cursor_col,&high,&low);
       if (NIL == mndsplay(Lmenu,&location,&bord))
       exit(0);
ercode =  mnread(Lmenu,lrow,lcol,&lrow,&lcol,
          &ch,&keycode,MN_DESTROY);
    sccurset(cursor_row,cursor_col);
    scpgcur(cursor_was_off,high,low,CUR_NO_ADJUST);
    count = ((lrow*4) + (lcol/15));
    if(ercode == 110) count = 110;    return  count;
}

ch_dir()
{
    int Crow, Ccol;
    BMENU  *Cmenu;
    BORDER  bord;
    WHERE   location;
```

```
int      mode, columns, act_page;
int      cursor_was_off, cursor_row, cursor_col, high, low;
int      ch, keycode;
Cmenu = mncreate(1,18,CYAN,REVERSE,NORMAL,NORMAL);
if (Cmenu == NIL)
   return b_wnerr;
if (NIL == mnitmkey(Cmenu,0,0,MN_NOPROTECT,
   " A:\\ ","aA",MN_SELECT))     return b_wnerr;
if (NIL == mnitmkey(Cmenu,0,6,MN_NOPROTECT,
   " B:\\ ","bB",MN_SELECT))     return b_wnerr;
if (NIL == mnitmkey(Cmenu,0,12,MN_PROTECT,
   " C:\\ ","cC",MN_SELECT))     return b_wnerr;
bord.type = BBRD_SSSS | BBRD_TCT;
bord.attr = GREEN | INTENSITY;
Crow = 0;   Ccol = (6*getdisk());
switch(getdisk()) {
case 0:   bord.pttitle = " A:\\ ";   break;
case 1:   bord.pttitle = " B:\\ ";   break;
case 2:   bord.pttitle = " C:\\ ";   break;
}
bord.ttattr =  REVERSE;
location.dev           = scmode(&mode,&columns,&act_page);
location.page          = act_page;
location.corner.row    = 6;
location.corner.col    = 10;
cursor_was_off = sccurst(&cursor_row,&cursor_col,&high,&low);
if (NIL == mndsplay(Cmenu,&location,&bord))
   return b_wnerr;
mnread(Cmenu,Crow,Ccol,&Crow,&Ccol,&ch,&keycode,
MN_DESTROY | MN_UNKNOWN_BEEP);
setdisk(Ccol/6);  drive = 'A'+(Ccol/6);
sccurset(7,71); printf("%5dK",disk_free());
sccurset(cursor_row,cursor_col);
```

```
    scpgcur(cursor_was_off,high,low,CUR_NO_ADJUST);
}

int disk_free()
{
      unsigned  d_free;
      int   clustor, byte_per_sec, sec_per_clus;
      int   Kb_per_c;
      getdfree(0, df);
      clustor       = df->df_avail;
      byte_per_sec = df->df_bsec;
      sec_per_clus = df->df_sclus;
      Kb_per_c = ( byte_per_sec * sec_per_clus / 1024 );
      d_free = (Kb_per_c * clustor);
      return d_free;
}
```

โปรแกรม EDIT.C

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
#include <bstrings.h>
#include <bvideo.h>
#include <bkeybrd.h>
#include <bwindow.h>

extern struct  _address  {
    int    time;
    char   even[5];
    int    data;
    int    dur;
    int    channel;
    struct  _address  *next;
    struct  _address  *prior;
    } list_entry;

extern struct  _address  *start;
extern struct  _address  *last;
extern struct  _address  *info;
extern char song_name[28], data_response[30];
int  next_time;

edit()
{
     int i;
     CURSOR_OFF;
     for(;;) {  intro();
       switch(menu2()) {
  case 0:    enter();
```

```
        CURSOR_OFF;  vidspmsg(12,59,CYAN,BLACK," Edit ");
 for(i=13;i<=18;i++) scclrmsg(i,44,35);  break;
 case 1:   delete();      break;
 case 2:   view();        break;
 case 3:   search();      break;
 case 4:   clear();       break;
 case 5:   copy_block();
        CURSOR_OFF;  vidspmsg(12,59,CYAN,BLACK," Edit ");
 for(i=13;i<=18;i++) scclrmsg(i,44,35);  break;
 case 6:   return;
      }
    }
}

enter()
{
  struct _address  *dls_store();
  int i, off_time, temp_ch, temp_dur;
  char  temp_even[5];
  vidspmsg(12,59,BLACK,WHITE," Edit ");
  for(i=13;i<=17;i++) scclrmsg(i,44,34);
  for(;;) {
    for(i=13;i<=17;i++) scclrmsg(i,49,27);
    info=(struct _address *)malloc(sizeof(list_entry));
    if(!info) {
      error(" Out of Memory.   Press <ESC>");
      return;
    }
        if(!input_time())  return;
off_time = info->time;
        if(!input_even())  return;
 strcpy(temp_even,info->even);
        if(!input_data())  return;
```

```
  if( strcmp(info->even,"PRO") &&
  strcmp(info->even,"VOL") && strcmp(info->even,"SUS!"))
    {
    if(!input_dur())  return;
    temp_dur = info->dur;
    } else info->dur = 0;
  if(!input_ch())  return;
  temp_ch = info->channel;
    start = dls_store(info,start);
    if( strcmp(info->even,"PRO") &&
    strcmp(info->even,"VOL") && strcmp(info->even,"SUS!"))  {
     info=(struct _address *)malloc(sizeof(list_entry));
     if(!info) {
      error(" Out of Memory.  Press <ESC>");
      return;
     }
       next_time = (off_time + temp_dur);
       info->time = (next_time -1);
       strcpy(info->even,temp_even);
       info->data = 0;
       info->dur = 0;
       info->channel = temp_ch;
       start = dls_store(info,start);
    }  sccurset(18,44); printf("Next time -> %-6d ",next_time);
  }
}

copy_block()
{
  struct _address  *dls_store(), *find(), *block;
  int      i, time, new, begin, stop, target, data, dur, channel;
  char     even[5], bstart[6], bstop[6], btarget[6];
  vidspmsg(12,59,BLACK,WHITE," Edit ");
```

```
  for(i=13;i<=18;i++) scclrmsg(i,44,34);
  CURSOR_ON;
  sccurset(13,44); printf("Enter Block START: ");
  eget(bstart);  if(!bstart[0])  return;
  begin = atoi(bstart);
  sccurset(14,44); printf("Enter Block STOP : ");
  eget(bstop);    if(!bstop[0])  return;
  stop = atoi(bstop);
  if(begin > stop)   {
  error(" Invalid parameter.  Press<ESC>");  return; }
  sccurset(15,44); printf("Enter TARGET time: ");
  eget(btarget);   if(!btarget[0])  return;
  target = atoi(btarget);
  new = (target-begin);
info = find(begin);
for(i=begin;i<=stop;i++) {
while(info->time == i)  {
        block =  info;
        time = info->time;
        strcpy(even,info->even);
        data = info->data;
        dur  = info->dur;
        channel = info->channel;
      info=(struct _address *)malloc(sizeof(list_entry));
      if(!info) {
       error(" Out of Memory.   Press <ESC>");
       return; }
      info->time = (time + new);
      strcpy(info->even,even);
      info->data = data;
      info->dur  = dur;
      info->channel = channel;
      start = dls_store(info,start);
```

```
      info = block;
      info = info->next;
      }
   }
}

int data_test(number)
char number[10];
{
   int i;
   if(strlen(number) > 6) {
   error(" Invalid Data.  Press<ESC>"); return 1; }
   for(i=0;i<strlen(number);i++)  {
   if(!isdigit(number[i])) {
   error(" Invalid Data. Press<ESC>"); return 1; } }
   if(65000-atoi(number) < 0)  {
   error(" Data Over Limit.  Press<ESC>"); return 1; }
   return 0;
}

int command_test(even)
char even[10];
{
  if(!strcmp(even,"PRO"))  return 0;
  if(!strcmp(even,"VOL"))  return 0;
  if(!strcmp(even,"SUS"))  return 0;
  if(!strcmp(even,"SUS!")) return 0;
  if(!strcmp(even,"C1"))   return 0;
  if(!strcmp(even,"C2"))   return 0;
  if(!strcmp(even,"C3"))   return 0;
  if(!strcmp(even,"C4"))   return 0;
  if(!strcmp(even,"C5"))   return 0;
  if(!strcmp(even,"C6"))   return 0;
```

```
if(!strcmp(even,"D1"))    return 0;
if(!strcmp(even,"D2"))    return 0;
if(!strcmp(even,"D3"))    return 0;
if(!strcmp(even,"D4"))    return 0;
if(!strcmp(even,"D5"))    return 0;
if(!strcmp(even,"E1"))    return 0;
if(!strcmp(even,"E2"))    return 0;
if(!strcmp(even,"E3"))    return 0;
if(!strcmp(even,"E4"))    return 0;
if(!strcmp(even,"E5"))    return 0;
if(!strcmp(even,"F1"))    return 0;
if(!strcmp(even,"F2"))    return 0;
if(!strcmp(even,"F3"))    return 0;
if(!strcmp(even,"F4"))    return 0;
if(!strcmp(even,"F5"))    return 0;
if(!strcmp(even,"G1"))    return 0;
if(!strcmp(even,"G2"))    return 0;
if(!strcmp(even,"G3"))    return 0;
if(!strcmp(even,"G4"))    return 0;
if(!strcmp(even,"G5"))    return 0;
if(!strcmp(even,"A1"))    return 0;
if(!strcmp(even,"A2"))    return 0;
if(!strcmp(even,"A3"))    return 0;
if(!strcmp(even,"A4"))    return 0;
if(!strcmp(even,"A5"))    return 0;
if(!strcmp(even,"B1"))    return 0;
if(!strcmp(even,"B2"))    return 0;
if(!strcmp(even,"B3"))    return 0;
if(!strcmp(even,"B4"))    return 0;
if(!strcmp(even,"B5"))    return 0;
if(!strcmp(even,"C1#"))   return 0;
if(!strcmp(even,"C2#"))   return 0;
if(!strcmp(even,"C3#"))   return 0;
```

```
  if(!strcmp(even,"C4#"))  return 0;
  if(!strcmp(even,"C5#"))  return 0;
  if(!strcmp(even,"D1#"))  return 0;
  if(!strcmp(even,"D2#"))  return 0;
  if(!strcmp(even,"D3#"))  return 0;
  if(!strcmp(even,"D4#"))  return 0;
  if(!strcmp(even,"D5#"))  return 0;
  if(!strcmp(even,"F1#"))  return 0;
  if(!strcmp(even,"F2#"))  return 0;
  if(!strcmp(even,"F3#"))  return 0;
  if(!strcmp(even,"F4#"))  return 0;
  if(!strcmp(even,"F5#"))  return 0;
  if(!strcmp(even,"G1#"))  return 0;
  if(!strcmp(even,"G2#"))  return 0;
  if(!strcmp(even,"G3#"))  return 0;
  if(!strcmp(even,"G4#"))  return 0;
  if(!strcmp(even,"G5#"))  return 0;
  if(!strcmp(even,"A1#"))  return 0;
  if(!strcmp(even,"A2#"))  return 0;
  if(!strcmp(even,"A3#"))  return 0;
  if(!strcmp(even,"A4#"))  return 0;
  if(!strcmp(even,"A5#"))  return 0;
  error(" Invalid Command.  Press <ESC>");
  return 1;
}

int input_time()
{
   char time[10];
   do {    CURSOR_ON;
   scclrmsg(13,50,28);
   sccurset(13,44); printf("time -> ");
   if(eget(time) || !time[0])  return 0;
```

```
    stpcvt(time,RWHITE);
       } while( data_test(time) );
    info->time = atoi(time);
    return 1;
}

int input_data()
{
    char data[10];
    do {      CURSOR_ON;
    scclrmsg(15,50,28);
    sccurset(15,44); printf("data -> ");
    if(eget(data))  return 0;
    if(!data[0]) { info->data = 64;  return 1; }
    stpcvt(data,RWHITE);
    if(atoi(data) > 128) error(" Data Over Limit.  Press<ESC>");
       } while( data_test(data) || atoi(data) > 128);
    info->data = atoi(data);
    return 1;
}

int input_dur()
{
    char dur[10];
    do {     CURSOR_ON;
    scclrmsg(16,50,28);
    sccurset(16,44); printf("durr -> ");
    if(eget(dur))  return 0;
    stpcvt(dur,RWHITE);
       } while(data_test(dur) || (!dur[0]) );
    info->dur = atoi(dur);
    return 1;
}
```

```
int input_ch()
{
   char ch[10];
   do {     CURSOR_ON;
   scclrmsg(17,50,28);
   sccurset(17,44); printf("chan -> ");
   if(eget(ch)) return 0;
   if(!ch[0]) { info->channel = 0;  return 1; }
   if(ch[0]==27) return 0;
   stpcvt(ch,RWHITE);
   if(atoi(ch) > 15) error(" Invalid Channel.  Press<ESC>");
      } while(data_test(ch) || atoi(ch) > 15);
   info->channel = atoi(ch);
   return 1;
}

int input_even()
{
   char temp[10];
   do {
      CURSOR_ON;
      scclrmsg(14,44,35);
      sccurset(14,44);
      printf("Even -> ");
      if(eget(temp)) return 0;
      stpcvt(temp,RWHITE | TOUP);
      } while(command_test(temp));
   strcpy(info->even,temp);
   return 1;
}

int  eget(data)
char data[10];
```

```
{
   char temp;
   int  i=0, code;
   for(;;) {
   do {
   temp = getch();
   code = kbscanof(temp);
   if(code==255) temp = getch();
      } while(code == 255);
     if(temp == 27)  return 1;
     if(temp == '\r') { data[i] = NULL; return 0; }
     if( (temp == '\b') && i>0 ) { printf("\b \b"); i--; data[i]='
     if(i==8) printf("\a");
     else {
     if(isprint(temp)) {
       if(!(temp==',' || temp==';')) {
       printf("%c",temp);
       data[i] = temp;
       i++;
       }
     }
     }
   }
}

delete()
{
   struct _address  *find(), *find_dur();
   int  s, off;
   get_data(6,28," Enter time to delete ");
   if(!data_response[0]) return;
   s = atoi(data_response);
   info=find(s);
```

```
off = (info->time + info->dur -1);
if(info)  {
  if(start==info)  {
    start=info->next;
    if(start) start->prior=NULL;
    else last=NULL;
  }
  else  {
    info->prior->next = info->next;
    if(info != last)
info->next->prior = info->prior;
    else
last=info->prior;
  }
  free(info);
}
info=find_dur(off);
if(!info->dur) {
if(info)  {
  if(start==info)  {
    start=info->next;
    if(start) start->prior=NULL;
    else last=NULL;
  }
  else  {
    info->prior->next = info->next;
    if(info != last)
info->next->prior = info->prior;
    else
last=info->prior;
  }
  free(info);  }
}
```

```
}
struct _address *find(time)
int      time;
{
    info = start;
    while(info)  {
      if(info->time == time)  return info;
      info=info->next;
    }
    error(" Not found !  < Press ESC >");
    return NULL;
}

struct _address *find_dur(time)
int      time;
{
    info = start;
    while(info)  {
      if(info->time == time)  return info;
      info=info->next;
    }
    return NULL;
}

search()
{
    BWINDOW *pwin;
    BORDER   bord;
    WHERE    location;
    int  time, mode, columns, act_page, line;
    char Vbuffer[80][25][2];
    struct _address *find();
    virdrect(0,0,24,79,&Vbuffer[0][0][0],CHAR_ATTR);
```

```
 pwin = wncreate(19,40,CYAN);
 if(pwin == NIL)  return;
 bord.type     = BBRD_NO_BORDER;
 bord.attr     = MAGENTA;
 bord.ttattr   = REVERSE;
 location.dev    = scmode(&mode,&columns,&act_page);
 location.page   = act_page;
 location.corner.row = 4;
 location.corner.col = 1;
line = 1;
get_data(7,6," Enter time to Find ");
if(!data_response[0])   return;
time = atoi(data_response);
if(!(info=find(time)))  return;
else {
 wndsplay(pwin,&location,&bord);
 viwrrect(0,0,24,79,&Vbuffer[0][0][0],0,0,CHAR_ATTR);
 CURSOR_OFF;
 while(info->time == time)  {
    wnprintf("%6d %c ",info->time,179);
    wnprintf("%5s %c ",info->even,179);
    wnprintf("%5d %c ",info->data,179);
    wnprintf("%5d %c ",info->dur,179);
    wnprintf("%2d ",info->channel,179);
    line++;
    if(line == 20)  {
       line=1; error2("          press any key to continue.
          }
    info = info->next;
    if(info->time == time)   wnprintf("\n");
   }
  utspkr(262);  utsleep(3);  utspkr(0);
  error2("          End of Listing <press any key>");
```

```
    wnremove(pwin);
    wndstroy(pwin);
   }
}

display(info)
struct _address *info;
{
       printf("\ntime: %d  ",info->time);
       printf("even: %s  ",info->even);
       printf("data: %d  ",info->data);
       printf("durr: %d  ",info->dur);
       printf("chan: %d \n",info->channel);
}

view()
{
    BWINDOW *pwin;
    BORDER   bord;
    WHERE    location;
    int  mode, columns, act_page, line;
    char Vbuffer[80][25][2];
    virdrect(0,0,24,79,&Vbuffer[0][0][0],CHAR_ATTR);
    pwin = wncreate(19,40,CYAN);
    if(pwin == NIL)  return;
    bord.type     = BBRD_NO_BORDER;
    bord.attr     = MAGENTA;
    bord.ttattr   = REVERSE;
    location.dev    = scmode(&mode,&columns,&act_page);
    location.page   = act_page;
    location.corner.row = 4;
    location.corner.col = 1;
    line = 1;
```

```
   wndsplay(pwin,&location,&bord);
   viwrrect(0,0,24,79,&Vbuffer[0][0][0],0,0,CHAR_ATTR);
   info = start;
   CURSOR_OFF;
   while(info)
   {
      wnprintf("%6d %c ",info->time,179);
      wnprintf("%5s %c ",info->even,179);
      wnprintf("%5d %c ",info->data,179);
      wnprintf("%5d %c ",info->dur,179);
      wnprintf("%2d ",info->channel,179);
      line++;
      if(line == 20)  {
         line=1; error2("          press any key to continue.
            }
      if(!(info=info->next)); else
      wnprintf("\n");
     }
   utspkr(262);  utsleep(3);  utspkr(0);
   error2("       End of Listing <press any key>       ");
   wnremove(pwin);
   wndstroy(pwin);
}

error2(message)
char message[60];
{
char buffer[80][1][2];
     virdrect(23,0,1,79,&buffer[0][0][0],CHAR_ATTR);
     CURSOR_OFF;
     vidspmsg(23,0,BLACK,15,message);
     getch();
     viwrrect(23,0,1,79,&buffer[0][0][0],0,0,CHAR_ATTR);
```

```
}

struct _address *dls_store(i,top)
struct _address *i;
struct _address *top;
{
    struct _address *old, *p;
    if(last==NULL)  {
      i->next=NULL;
      i->prior=NULL;
      last=i;
      return i;
    }
    p = top;
    old = NULL;
    while(p)  {
        if(p->time < i->time)  {
 old = p;
 p = p->next;
        }
        else  {
if(p->prior)  {
   p->prior->next=i;
   i->next=p;
   i->prior = p->prior;
   p->prior = i;
   return top;
}
i->next = p;
i->prior = NULL;
p->prior = i;
return  i;
    }
```

```
    }
    old->next = i;
    i->next = NULL;
    i->prior = old;
    last = i;
    return start;
}
```

โปรแกรม PLAY.C

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <butil.h>
#include <bvideo.h>
#include <midi.h>

char        drive;
int         tempo, _rhythm, song_point, loop, def_ch;
char        data_response[30];
long        last_tick, tick;

struct  _address  {
    int   time;
    char  even[5];
    int   data;
    int   dur;
    int   channel;
    struct  _address  *next;
    struct  _address  *prior;
    }  list_entry;

struct  _address *start;
struct  _address *last;
struct  _address *info;

extern char song_name[28];
void   program_ch(), note_on();

play()
{
    CURSOR_OFF;
```

```
   vidspmsg(24,1,INTENSITY|GREEN,BLACK,
   "Play / Tempo / Voice / Channel / Rhythm / Quit ?");

     for(;;) {  intro();
         switch(toupper(getch())) {

case 'P':  bwindow(4,46," PLAYING");
                    to_midi();  stop();  break;

case 'T':  get_tempo(); break;

case 'R':  if(!_rhythm)  _rhythm = 1; else _rhythm = 0;  break

         case 'V':  voice();  break;

         case 'C':  channel();  break;

case 'Q':  CLS;  return;
      }
    }
}

to_midi()
{
  info  = start;
  song_point = last_tick = 0;
    if(_rhythm) { START; }
    while(info)
    {
do {
         utgetclk(&tick); } while(tick == last_tick);
         last_tick = tick;
POLL:    if(song_point == info->time) {
```

```
        table(info->data,def_ch);
        info = info->next;  goto POLL; }
      if(_rhythm) { CLOCK; }
      song_point++;
    }
}

void program_ch(number,channel)
int  number;
{
   check; outportb(data_port,_program_ch + channel);
   check; outportb(data_port,number);
}

void note_on(note, velocity ,channal)
int note, velocity, channal;
{
   check; outportb(data_port,_note_on + channal);
   check; outportb(data_port,note);
   check; outportb(data_port,velocity);
}

all_note_off(channel)
int channel;
{
   check; outportb(data_port,_all_note_off + channel);
   check; outportb(data_port,0);
}

volume(vol,channel)
int vol, channel;
{
   check; outportb(data_port, _control_ch + channel);
```

```
    check; outportb(data_port, 7);
    check; outportb(data_port,vol);
}

sustain(sus, channel)
int sus, channel;
{
    check; outportb(data_port, _control_ch + channel);
    check; outportb(data_port, 64);
    check; outportb(data_port,sus);
}

stop()
{
int i;
   if(_rhythm) { STOP; }
   for(i=0;i<=15;i++) all_note_off(i); wnout();
}

intro()
{
    sccurset(4,61); printf(" %c: %-13s",drive,song_name);
    sccurset(10,61); if(_rhythm) printf("Rhythm: ON ");
            else         printf("Rhythm: OFF");
    sccurset(10,52); printf("%3d",tempo);
    sccurset(7,53);  printf("%2d",def_ch);
}

int table(_data,_channel)
int _data, _channel;
{
char ch;
if(!strcmp(info->even,"PRO")) {program_ch(_data,_channel);return;}
```

```
f(!strcmp(info->even,"VOL")) { volume(_data, _channel); return; }
f(!strcmp(info->even,"SUS")) { sustain(_data, _channel); return; }
f(!strcmp(info->even,"SUS!")){ sustain(_data, _channel); return; }
f(!strcmp(info->even,"LOOP"))  return;
f(!strcmp(info->even,"C1")) { note_on(_C1,_data,_channel); return; }
f(!strcmp(info->even,"C2")) { note_on(_C2,_data,_channel); return; }
f(!strcmp(info->even,"C3")) { note_on(_C3,_data,_channel); return; }
f(!strcmp(info->even,"C4")) { note_on(_C4,_data,_channel); return; }
f(!strcmp(info->even,"C5")) { note_on(_C5,_data,_channel); return; }
f(!strcmp(info->even,"C6")) { note_on(_C6,_data,_channel); return; }
f(!strcmp(info->even,"D1")) { note_on(_D1,_data,_channel); return; }
f(!strcmp(info->even,"D2")) { note_on(_D2,_data,_channel); return; }
f(!strcmp(info->even,"D3")) { note_on(_D3,_data,_channel); return; }
f(!strcmp(info->even,"D4")) { note_on(_D4,_data,_channel); return; }
f(!strcmp(info->even,"D5")) { note_on(_D5,_data,_channel); return; }
f(!strcmp(info->even,"E1")) { note_on(_E1,_data,_channel); return; }
f(!strcmp(info->even,"E2")) { note_on(_E2,_data,_channel); return; }
f(!strcmp(info->even,"E3")) { note_on(_E3,_data,_channel); return; }
if(!strcmp(info->even,"E4")) { note_on(_E4,_data,_channel); return; }
if(!strcmp(info->even,"E5")) { note_on(_E5,_data,_channel); return; }
if(!strcmp(info->even,"F1")) { note_on(_F1,_data,_channel); return; }
if(!strcmp(info->even,"F2")) { note_on(_F2,_data,_channel); return; }
if(!strcmp(info->even,"F3")) { note_on(_F3,_data,_channel); return; }
if(!strcmp(info->even,"F4")) { note_on(_F4,_data,_channel); return; }
if(!strcmp(info->even,"F5")) { note_on(_F5,_data,_channel); return; }
if(!strcmp(info->even,"G1")) { note_on(_G1,_data,_channel); return; }
if(!strcmp(info->even,"G2")) { note_on(_G2,_data,_channel); return; }
if(!strcmp(info->even,"G3")) { note_on(_G3,_data,_channel); return; }
if(!strcmp(info->even,"G4")) { note_on(_G4,_data,_channel); return; }
if(!strcmp(info->even,"G5")) { note_on(_G5,_data,_channel); return; }
if(!strcmp(info->even,"A1")) { note_on(_A1,_data,_channel); return; }
if(!strcmp(info->even,"A2")) { note_on(_A2,_data,_channel); return; }
if(!strcmp(info->even,"A3")) { note_on(_A3,_data,_channel); return; }
```

```
f(!strcmp(info->even,"A4")) { note_on(_A4,_data,_channel); return; }
f(!strcmp(info->even,"A5")) { note_on(_A5,_data,_channel); return; }
f(!strcmp(info->even,"B1")) { note_on(_B1,_data,_channel); return; }
f(!strcmp(info->even,"B2")) { note_on(_B2,_data,_channel); return; }
f(!strcmp(info->even,"B3")) { note_on(_B3,_data,_channel); return; }
f(!strcmp(info->even,"B4")) { note_on(_B4,_data,_channel); return; }
f(!strcmp(info->even,"B5")) { note_on(_B5,_data,_channel); return; }
f(!strcmp(info->even,"C1#")){note_on(_C1o,_data,_channel); return; }
f(!strcmp(info->even,"C2#")){note_on(_C2o,_data,_channel); return; }
f(!strcmp(info->even,"C3#")){note_on(_C3o,_data,_channel); return; }
f(!strcmp(info->even,"C4#")){note_on(_C4o,_data,_channel); return; }
f(!strcmp(info->even,"C5#")){note_on(_C5o,_data,_channel); return; }
f(!strcmp(info->even,"D1#")){note_on(_D1o,_data,_channel); return; }
f(!strcmp(info->even,"D2#")){note_on(_D2o,_data,_channel); return; }
f(!strcmp(info->even,"D3#")){note_on(_D3o,_data,_channel); return; }
f(!strcmp(info->even,"D4#")){note_on(_D4o,_data,_channel); return; }
if(!strcmp(info->even,"D5#")){note_on(_D5o,_data,_channel); return; }
if(!strcmp(info->even,"F1#")){note_on(_F1o,_data,_channel); return; }
if(!strcmp(info->even,"F2#")){note_on(_F2o,_data,_channel); return; }
if(!strcmp(info->even,"F3#")){note_on(_F3o,_data,_channel); return; }
if(!strcmp(info->even,"F4#")){note_on(_F4o,_data,_channel); return; }
if(!strcmp(info->even,"F5#")){note_on(_F5o,_data,_channel); return; }
if(!strcmp(info->even,"G1#")){note_on(_G1o,_data,_channel); return; }
if(!strcmp(info->even,"G2#")){note_on(_G2o,_data,_channel); return; }
if(!strcmp(info->even,"G3#")){note_on(_G3o,_data,_channel); return; }
if(!strcmp(info->even,"G4#")){note_on(_G4o,_data,_channel); return; }
if(!strcmp(info->even,"G5#")){note_on(_G5o,_data,_channel); return; }
if(!strcmp(info->even,"A1#")){note_on(_A1o,_data,_channel); return; }
if(!strcmp(info->even,"A2#")){note_on(_A2o,_data,_channel); return; }
if(!strcmp(info->even,"A3#")){note_on(_A3o,_data,_channel); return; }
if(!strcmp(info->even,"A4#")){note_on(_A4o,_data,_channel); return; }
if(!strcmp(info->even,"A5#")){note_on(_A5o,_data,_channel); return; }
stop();      scpclr();
```

```
sccurset(4,0); puts("Unknow Command or Even...     program abort...")
     display(info);
  printf("%s Not save or Save ?  [y/n] ",song_name);
  do {  ch = toupper(getch());
  switch(ch) {
  case 'Y': save();  break;
  case 'N': break; }
     } while(ch != 'Y' && ch != 'N');
        quit();
}

get_tempo()
{
     int x;
     get_data(6,10," Enter new Tempo ");
     if(!data_response[0]) return;
     x = atoi(data_response);
     if(x < 20 || x > 200)
       error(" Out of range 20-200. Press <ESC>");
     else tempo = x;
     set_8253(tempo);
}

set_8253(speed)
int speed;
{
    unsigned time;
    time = (50000/speed)*24 + 5530;
    outportb(0x43,0x36);
    outportb(0x40,time % 256);
    outportb(0x40,time / 256);
}
```

```
channel()
{
     int x;
     get_data(6,10," Enter working Channel ");
     if(!data_response[0]) return;
     x = atoi(data_response);
     if(x < 0 || x > 15)
     error(" Out of range 0-15.    Press <ESC>");
     else def_ch = x;
}
```

โปรแกรม VOICE.C

```
#include <dos.h>
#include <midi.h>
#include <stdio.h>
#include <butil.h>
#include <bscreens.h>
#include <bmenu.h>

extern int def_ch;
int Vrow, Vcol;

voice()
{
    BMENU  *Vmenu;
    BORDER  bord;
    WHERE   location;
    int     mode, columns, act_page;
    int     cursor_was_off, cursor_row, cursor_col, high, low;
    int     ercode, ch, keycode;
    Vmenu = mncreate(8,29,REVERSE,CYAN,NORMAL,NORMAL);
    if (Vmenu == NIL)
       return b_wnerr;
    if (NIL == mnitmkey(Vmenu,0,0,MN_NOPROTECT,
      "JAZZ ORGAN  ","jJ",MN_SELECT ))  return b_wnerr;
    if (NIL == mnitmkey(Vmenu,1,0,MN_NOPROTECT,
      "PIPE ORGAN  ","Pp",MN_SELECT ))  return b_wnerr;
    if (NIL == mnitmkey(Vmenu,2,0,MN_NOPROTECT,
      "STRINGS     ","sS",MN_SELECT))   return b_wnerr;
    if (NIL == mnitmkey(Vmenu,3,0,MN_NOPROTECT,
      "BRASS 1     ","bB",MN_SELECT))   return b_wnerr;
    if (NIL == mnitmkey(Vmenu,4,0,MN_NOPROTECT,
      "BRASS 2     ","bB",MN_SELECT))   return b_wnerr;
    if (NIL == mnitmkey(Vmenu,5,0,MN_NOPROTECT,
```

```
  "BRASS&CHIMES","bB",MN_SELECT))   return b_wnerr;
if (NIL == mnitmkey(Vmenu,6,0,MN_NOPROTECT,
  "CLARINET    ","cC",MN_SELECT))   return b_wnerr;
if (NIL == mnitmkey(Vmenu,7,0,MN_NOPROTECT,
  "CALLIOP     ","cC",MN_SELECT))   return b_wnerr;
if (NIL == mnitmkey(Vmenu,0,14,MN_NOPROTECT,
  "PIANO       ","pP",MN_SELECT))   return b_wnerr;
if (NIL == mnitmkey(Vmenu,1,14,MN_NOPROTECT,
  "ELEC. PIANO ","eE",MN_SELECT))   return b_wnerr;
if (NIL == mnitmkey(Vmenu,2,14,MN_NOPROTECT,
  "HARPSICHORD ","hH",MN_SELECT))   return b_wnerr;
if (NIL == mnitmkey(Vmenu,3,14,MN_NOPROTECT,
  "VIBES       ","vV",MN_SELECT))   return b_wnerr;
if (NIL == mnitmkey(Vmenu,4,14,MN_NOPROTECT,
  "JAZZ GUITAR ","jJ",MN_SELECT))   return b_wnerr;
if (NIL == mnitmkey(Vmenu,5,14,MN_NOPROTECT,
  "HAWAI GUITAR","hH",MN_SELECT))   return b_wnerr;
if (NIL == mnitmkey(Vmenu,6,14,MN_NOPROTECT,
  "MUSIC BOX   ","mM",MN_SELECT))   return b_wnerr;
if (NIL == mnitmkey(Vmenu,7,14,MN_NOPROTECT,
   "COSMIC      ","cC",MN_SELECT))   return b_wnerr;
bord.type = BBRD_SSSS | BBRD_TCT | BBRD_BCT;
bord.attr = REVERSE;
bord.pttitle = " Voice Libary.";
bord.ttattr =  REVERSE;
bord.pbtitle = " Press ESC to quit ";
bord.btattr = REVERSE;
location.dev          = scmode(&mode,&columns,&act_page);
location.page         = act_page;
location.corner.row   = 6;
location.corner.col   = 18;
cursor_was_off = sccurst(&cursor_row,&cursor_col,&high,&low);
if (NIL == mndsplay(Vmenu,&location,&bord))
```

```
      return b_wnerr;
do {
ercode =  mnread(Vmenu,Vrow,Vcol,&Vrow,&Vcol,&ch,&keycode,
          MN_ALL_TRANSMIT);
   if(Vcol==0) {
     switch(Vrow) {
     case 0:   program_ch(0,def_ch); break;
     case 1:   program_ch(1,def_ch); break;
     case 2:   program_ch(2,def_ch); break;
     case 3:   program_ch(3,def_ch); break;
     case 4:   program_ch(4,def_ch); break;
     case 5:   program_ch(5,def_ch); break;
     case 6:   program_ch(6,def_ch); break;
     case 7:   program_ch(7,def_ch); break;   }
   }
   else  {
   if(Vcol==14) {
      switch(Vrow) {
     case 0:   program_ch(8,def_ch); break;
     case 1:   program_ch(9,def_ch); break;
     case 2:   program_ch(10,def_ch); break;
     case 3:   program_ch(11,def_ch); break;
     case 4:   program_ch(12,def_ch); break;
     case 5:   program_ch(13,def_ch); break;
     case 6:   program_ch(14,def_ch); break;
     case 7:   program_ch(15,def_ch); break; }
         }
  }
    }while(ercode != 110);
   mndstroy(Vmenu);
   sccurset(cursor_row,cursor_col);
   scpgcur(cursor_was_off,high,low,CUR_NO_ADJUST);
}
```

โปรแกรม MENU.C

```
#include <stdio.h>
#include <butil.h>
#include <bscreens.h>
#include <bmenu.h>

int  row, col, row3, col3, row2, col2;;

/*      M A I N     M E N U     */
extern int menu3()
{
    BMENU  *pmenu;
    BORDER  bord;
    WHERE   location;
    int     mode, columns, act_page;
    int     ch, keycode;
    pmenu = mncreate(1,78,CYAN,REVERSE,NORMAL,NORMAL);
    if (pmenu == NIL)
       return b_wnerr;
    if (NIL == mnitmkey(pmenu,0,5,MN_NOPROTECT,"File",
      "fF",MN_SELECT)) return b_wnerr;
    if (NIL == mnitmkey(pmenu,0,14,MN_NOPROTECT,"Edit",
      "eE",MN_SELECT)) return b_wnerr;
    if (NIL == mnitmkey(pmenu,0,23,MN_NOPROTECT,"Play",
      "pP",MN_SELECT)) return b_wnerr;
    if (NIL == mnitmkey(pmenu,0,32,MN_NOPROTECT,"Record",
      "rR",MN_SELECT)) return b_wnerr;
    if (NIL == mnitmkey(pmenu,0,43,MN_NOPROTECT,"Channel",
      "cC",MN_SELECT)) return b_wnerr;
    if (NIL == mnitmkey(pmenu,0,55,MN_NOPROTECT,"Instru",
      "iI",MN_SELECT)) return b_wnerr;
    if (NIL == mnitmkey(pmenu,0,66,MN_NOPROTECT,"Quit",
```

```
    "qQ",MN_SELECT)) return b_wnerr;
  bord.type = BBRD_SSSS;
  bord.attr = GREEN;
  location.dev          = scmode(&mode,&columns,&act_page);
  location.page         = act_page;
  location.corner.row   = 1;
  location.corner.col   = 1;
  if (NIL == mndsplay(pmenu,&location,&bord))
     return b_wnerr;
  mnread(pmenu,row3,col3,&row3,&col3,&ch,&keycode,MN_TRANSMIT);
  return(col3);
}

/*      F I L E      M E N U      */
extern int menu1()
{
  BMENU  *Fmenu;
  BORDER  Fbord;
  WHERE   Flocation;
  int     ercode, ch, keycode, Fmode, Fcolumns, Fact_page;
  Fmenu = mncreate(6,14,REVERSE,CYAN,NORMAL,NORMAL);
  if (Fmenu == NIL)
     return b_wnerr;
  if (NIL == mnitmkey(Fmenu,0,0,MN_NOPROTECT," Load         ",
    "lL",MN_SELECT)) return b_wnerr;
  if (NIL == mnitmkey(Fmenu,1,0,MN_NOPROTECT," Save         ",
    "sS",MN_SELECT)) return b_wnerr;
  if (NIL == mnitmkey(Fmenu,2,0,MN_NOPROTECT," Rename       ",
    "rR",MN_SELECT)) return b_wnerr;
  if (NIL == mnitmkey(Fmenu,3,0,MN_NOPROTECT," Change dir   ",
    "cC",MN_SELECT)) return b_wnerr;
  if (NIL == mnitmkey(Fmenu,4,0,MN_NOPROTECT," OS shell     ",
    "oO",MN_SELECT)) return b_wnerr;
```

```
   if (NIL == mnitmkey(Fmenu,5,0,MN_NOPROTECT," Quit          ",
     "qQ",MN_SELECT)) return b_wnerr;
   Fbord.type = BBRD_SSSS | BBRD_TCT;
   Fbord.attr = REVERSE;
   Fbord.ttattr = REVERSE;
   Fbord.pttitle = " File";
   Flocation.dev           = scmode(&Fmode,&Fcolumns,&Fact_page);
   Flocation.page          = Fact_page;
   Flocation.corner.row    = 1;
   Flocation.corner.col    = 1;
   if (NIL == mndsplay(Fmenu,&Flocation,&Fbord))
      return b_wnerr;
   ercode =   mnread(Fmenu,row,col,&row,&col,&ch,&keycode,
    MN_DESTROY);
   if(ercode == 110) row=5;
   return(row);
}

/*      E D I T     M E N U      */
extern int menu2()
{
   BMENU  *pmenu;
   BORDER  bord;
   WHERE   location;
   int     mode, columns, act_page;
   int     cursor_was_off, cursor_row, cursor_col, high, low;
   int     ercode, ch, keycode;
   pmenu = mncreate(7,13,REVERSE,CYAN,NORMAL,NORMAL);
   if (pmenu == NIL)
      return b_wnerr;
   if (NIL == mnitmkey(pmenu,0,0,MN_NOPROTECT," Edit          ",
     "eE",MN_SELECT)) return b_wnerr;
   if (NIL == mnitmkey(pmenu,1,0,MN_NOPROTECT," Delete        ",
```

```
    "dD",MN_SELECT)) return b_wnerr;
  if (NIL == mnitmkey(pmenu,2,0,MN_NOPROTECT," View       ",
    "Vv",MN_SELECT)) return b_wnerr;
  if (NIL == mnitmkey(pmenu,3,0,MN_NOPROTECT," Find       ",
    "fF",MN_SELECT)) return b_wnerr;
  if (NIL == mnitmkey(pmenu,4,0,MN_NOPROTECT," Clear      ",
    "cC",MN_SELECT)) return b_wnerr;
  if (NIL == mnitmkey(pmenu,5,0,MN_NOPROTECT," Block Copy ",
    "bB",MN_SELECT)) return b_wnerr;
  if (NIL == mnitmkey(pmenu,6,0,MN_NOPROTECT," Quit       ",
    "qQ",MN_SELECT)) return b_wnerr;
  bord.type = BBRD_SSSS | BBRD_TCT;
  bord.attr = REVERSE;
  bord.ttattr = REVERSE;
  bord.pttitle = " Edit ";
  location.dev         = scmode(&mode,&columns,&act_page);
  location.page        = act_page;
  location.corner.row  = 1;
  location.corner.col  = 10;
  cursor_was_off = sccurst(&cursor_row,&cursor_col,&high,&low);
  if (NIL == mndsplay(pmenu,&location,&bord))
     return b_wnerr;
ercode =  mnread(pmenu,row2,col2,&row2,&col2,&ch,&keycode,
          MN_DESTROY);
  sccurset(cursor_row,cursor_col);
  scpgcur(cursor_was_off,high,low,CUR_NO_ADJUST);
  if(ercode == 110) row2 = 6;
  return(row2);
}
```

โปรแกรม WINDOW.C

```
#include <bscreens.h>
#include <butil.h>
#include <bwindow.h>
#include <string.h>
#include <bstrings.h>

BWINDOW *Win;
BORDER   Wbord;
WHERE    Wlocation;
int  Wmode, Wcolumns, Wact_page;
int  Wcursor_was_off, Wrow, Wcol, Whigh, Wlow;
extern  char  data_response[40];

bwindow(locate_y, locate_x, message)
int   locate_y, locate_x;
char *message;
{
    CURSOR_OFF;
    Win = wncreate(1,1+strlen(message),REVERSE);
    if(Win == NIL)  return;
    Wbord.type     = BBRD_SSSS;
    Wbord.attr     = CYAN | INTENSITY;
    Wlocation.dev    = scmode(&Wmode,&Wcolumns,&Wact_page);
    Wlocation.page   = Wact_page;
    Wlocation.corner.row = locate_y;
    Wlocation.corner.col = locate_x;
    Wcursor_was_off = sccurst(&Wrow,&Wcol,&Whigh,&Wlow);
    wndsplay(Win,&Wlocation,&Wbord);
    wnprintf(message);
}

wnout()
```

```
{
    wnremove(Win);
    sccurset(Wrow,Wcol);
    scpgcur(Wcursor_was_off,Whigh,Wlow,CUR_NO_ADJUST);
    wndstroy(Win);
}

char  *get_data(G_row, G_col, message)
int G_row,G_col;
char message[30];
{
    BWINDOW *win;
    BORDER   bord;
    WHERE    location;
    int  mode, columns, act_page;
    int  scan;
    win = wncreate(1,30,CYAN);
    bord.type    = BBRD_SSSS | BBRD_TCT;
    bord.attr    = MAGENTA;
    bord.ttattr  = REVERSE;
    bord.pttitle = message;
    location.dev    = scmode(&mode,&columns,&act_page);
    location.page   = act_page;
    location.corner.row = G_row;
    location.corner.col = G_col;
    wndsplay(win,&location,&bord);
    wnquery(data_response,sizeof(data_response),&scan);
    stpcvt(data_response,RWHITE | TOUP);
    wnremove(win);
    wndstroy(win);
    return data_response;
}
```

โปรแกรม MIDI.H (เป็น include file เก็บค่าคงที่ต่างๆ)

```
/* ------------------------------------------------------------ */
/*                      M I D I     C O D E S                   */
/* ------------------------------------------------------------ */

 /* ------------------   PORT DEFINITIONS  ------------------- */
#define  control_port 0x2B1
#define  data_port    0x2B0
/* ------------------------------------------------------------ */

/* ---------------------   MIDI STATUS  ----------------------- */
#define  _note_off         0x80    /* * * * * * * ** * * * * * * ** */
#define  _note_on          0x90    /*                              */
#define  _control_ch       0xB0    /*  status byte for channel 0   */
#define  _program_ch       0xC0    /*                              */
#define  _pitch_wheel_ch   0xE0    /* * ** * * * * * * * * * * **/
/* ------------------------------------------------------------ */

/* ---------------  SYSTEM REAL TIME MESSAGES  ---------------- */
#define  _timing_clock     0xF8
#define  _start            0xFA
#define  _continue         0xFB
#define  _stop             0xFC
#define  _active_sensing   0xFE
#define  _system_reset     0xFF
/* ------------------------------------------------------------ */

/* -------------------- CHANNEL MODE MESSAGES ------------------ */
#define  _all_note_off   123
/* ------------------------------------------------------------ */

/* ---------------------   NOTE NUMBER  ----------------------- */
#define  _C1   36
```

```
#define  _C2  48
#define  _C3  60
#define  _C4  72
#define  _C5  84
#define  _C6  96
#define  _C1o 37
#define  _C2o 49
#define  _C3o 61
#define  _C4o 73
#define  _C5o 85
#define  _D1  38
#define  _D2  50
#define  _D3  62
#define  _D4  74
#define  _D5  86
#define  _D1o 39
#define  _D2o 51
#define  _D3o 63
#define  _D4o 75
#define  _D5o 87
#define  _E1  40
#define  _E2  52
#define  _E3  64
#define  _E4  76
#define  _E5  88
#define  _F1  41
#define  _F2  53
#define  _F3  65
#define  _F4  77
#define  _F5  89
#define  _F1o 42
#define  _F2o 54
#define  _F3o 66
```

```
#define  _F4o 78
#define  _F5o 90
#define  _G1  43
#define  _G2  55
#define  _G3  67
#define  _G4  79
#define  _G5  91
#define  _G1o 44
#define  _G2o 56
#define  _G3o 68
#define  _G4o 80
#define  _G5o 92
#define  _A1  45
#define  _A2  57
#define  _A3  69
#define  _A4  81
#define  _A5  93
#define  _A1o 46
#define  _A2o 58
#define  _A3o 70
#define  _A4o 82
#define  _A5o 94
#define  _B1  47
#define  _B2  59
#define  _B3  71
#define  _B4  83
#define  _B5  95
/* ------------------------------------------------------------ */
#define  check   while(!(inportb(control_port) & 1))
                 /* test DO=1 ? if not, loop until DO=1 */
#define   START   check;  outportb(data_port,_start)
#define   STOP    check;  outportb(data_port,_stop)
#define   CLOCK   check;  outportb(data_port,_timing_clock)
```

TABLE I

SUMMARY OF STATUS BYTES

STATUS D7—D0	# OF DATA BYTES	DESCRIPTION
Channel Voice Messages		
1000nnnn	2	Note Off event
1001nnnn	2	Note On event (velocity=0 Note Off)
1010nnnn	2	Polyphonic key pressure/aftertouch
1011nnnn	2	Control change
1100nnnn	1	Program change
1101nnnn	1	Channel Pressure (Aftertouch)
1110nnnn	2	Pitch wheel change
CHANNEL MODE MESSAGES		
1011nnnn	2	Selects Channel Mode
SYSTEM MESSAGES		
11110000	******	System Exclusive
11110sss	0 to 2	System Common
11111ttt	0	System Real Time

NOTES:

nnnn: N−1, where N = Channel #,
i.e. 0000 is Channel 1.
0001 is Channel 2.
.
.
.
1111 is Channel 16.

*****: Oiiiiiii, data, . . . , EOX

iiiiiii: Identification

sss: 1 to 7

ttt: 0 to 7

TABLE II

CHANNEL VOICE MESSAGES

STATUS	DATA BYTES	DESCRIPTION
1000nnnn	0kkkkkkk	Note Off (see notes 1-4)
	0vvvvvvv	vvvvvvv: note off velocity
1001nnnn	0kkkkkkk	Note On (see Notes 1-4)
	0vvvvvvv	vvvvvvv≠0: velocity
		vvvvvvv=0: note off
1010nnnn	0kkkkkkk	Polyphonic Key Pressure (After-Touch)
	0vvvvvvv	vvvvvvv: pressure value
1011nnnn	0ccccccc	Control Change
	0vvvvvvv	ccccccc: control # (0-121) (see notes 5-8)
		vvvvvvv: control value
		ccccccc=122 thru 127: Reserved. See Table III.
1100nnnn	0ppppppp	Program Change
		ppppppp: program number (0-127)
1101nnnn	0vvvvvvv	Channel Pressure (After-Touch)
		vvvvvvv: pressure value
1110nnnn	0vvvvvvv	Pitch Wheel Change LSB (see note 10)
	0vvvvvvv	Pitch Wheel Change MSB

Notes for Table II on the following page.

NOTES:

1. nnnn: Voice Channel # (1-16, coded as defined in Table I notes)
2. kkkkkkk: note # (0-127)
   kkkkkkk=60: Middle C of keyboard

0	12	24	36	48	60	72	84	96	108	120	127
		c	c	c	c	c	c	c	c		
		piano range									

3. vvvvvvv: key velocity
   A logarithmic scale would be advisable.

0	1			64				127
off	ppp	pp	p	mp	mf	f	f	fff

   vvvvvvv=64: in case of no velocity sensors
   vvvvvvv=0: Note Off, with velocity=64
4. Any Note On message sent should be balanced by sending a Note Off message for that note in that channel at some later time.
5. ccccccc: control number

ccccccc	Description
0	Continuous Controller 0 MSB
1	Continuous Controller 1 MSB (MODULATION WHEEL)
2	Continuous Controller 2 MSB
3	Continuous Controller 3 MSB
4-31	Continuous Controller 4-31 MSB
32	Continuous Controller 0 LSB
33	Continuous Controller 1 LSB (MODULATION WHEEL)
34	Continuous Controller 2 LSB
35	Continuous Controller 3 LSB
36-63	Continuous Controller 4-31 LSB
64-95	Switches (On/Off)
96-121	Undefined
122-127	Reserved for Channel Mod message (see Table III).

6. The controllers are not specifically defined. A manufacturer can assign the logical controllers to physical ones as necessary. The controller allocation table must be provided in the user's operation manual.
7. Continuous controllers are divided into Most Significant and Least Significant Bytes. If only seven bits of resolution are needed for any particular controllers, only the MSB is sent. It is not necessary to send the LSB. If more resolution is needed, then both are sent, first the MSB, then the LSB. If only the LSB has changed in value, the LSB may be sent without re-sending the MSB.
8. vvvvvvv: control value (MSB)
   (for controllers)

0	127
min	max

   (for switches)

0	127
off	on

   Numbers 1 through 126, inclusive, are ignored.
9. Any messages (e.g. Note On), which are sent successively under the same status, can be sent without a Status byte until a different Status byte is needed.
10. Sensitivity of the pitch bender is selected in the receiver. Center position value (no pitch change) is 2000H, which would be transmitted EnH-00H-40H.

TABLE III
CHANNEL MODE MESSAGES

STATUS	DATA BYTES	DESCRIPTION
1011nnnn	0ccccccc 0vvvvvvv	Mode Messages ccccccc=122: Local Control vvvvvvv=0, Local Control Off vvvvvvv=127, Local Control On ccccccc=123: All Notes Off vvvvvvv=0 ccccccc=124: Omni Mode Off (All Notes Off) vvvvvvv=0 ccccccc=125: Omni Mode On (All Notes Off) vvvvvvv=0 ccccccc=126: Mono Mode On (Poly Mode Off) (All Notes Off) vvvvvvv=M, where M is the number of channels vvvvvvv=0, the number of channels equals the number of voices in the receiver. ccccccc=127: Poly Mode On (Mono Mode Off) vvvvvvv=0 (All Notes Off)

S:

n: Basic Channel # (1-16, coded as defined in Table I)
ssages 123 thru 127 function as All Notes Off mes-
es. They will turn off all voices controlled by the
gned Basic Channel. Except for message 123, All
es Off, they should not be sent periodically, but only
a specific purpose. In no case should they be used in
of Note Off commands to turn off notes which have
n previously turned on. Therefore any All Notes Off
mand (123-127) may be ignored by receiver with no
sibility of notes staying on, since any Note On com-
nd must have a corresponding specific Note Off com-
nd.

ntrol Change #122, Local Control, is optionally used to
rrupt the internal control path between the keyboard, for example, and the sound-generating circuitry. If 0 (Local Off message) is received, the path is disconnected: the keyboard data goes only to MIDI and the sound-generating circuitry is controlled only by incoming MIDI data. If a 7FH (Local On message) is received, normal operation is restored.

4. The third byte of 'Mono' specifies the number of channels in which Monophonic Voice messages are to be sent. This number, "M," is a number between 1 and 16. The channel(s) being used, then, will be the current Basic Channel (=N) thru N+M−1 up to a maximum of 16. If M=0, this is a special case directing the receiver to assign all its voices, one per channel, from the Basic Channel N through 16.

TABLE IV
SYSTEM COMMON MESSAGES

STATUS	DATA BYTES	DESCRIPTION
11110001		Undefined
11110010		Song Position Pointer
	01111111	1111111: (Least significant) hhhhhhh: (Most significant)
11110011	0sssssss	Song Select sssssss: Song #
11110100		Undefined
11110101		Undefined
11110110	none	Tune Request
11110111	none	EOX: "End of System Exclusive" flag

S:

g Position Pointer: Is an internal register which holds
number of MIDI beats (1 beat = 6 MIDI clocks)
e the start of the song. Normally it is set to 0 when
START switch is pressed, which starts sequence play-
k. It then increments with every sixth MIDI clock
ipt, until STOP is pressed. If CONTINUE is pressed,
ontinues to increment. It can be arbitrarily preset (to a
olution of 1 beat) by the SONG POSITION
INTER message.

2. Song Select: Specifies which song or sequence is to be played upon receipt of a Start (Real-Time) message.
3. Tune Request: Used with analog synthesizers to request them to tune their oscillators.
4. EOX: Used as a flag to indicate the end of a System Exclusive transmission (see Table VI).

TABLE V

SYSTEM REAL TIME MESSAGES

US	DATA BYTES	DESCRIPTION
00		Timing Clock
01		Undefined
10		Start
11		Continue
00		Stop
01		Undefined
10		Active Sensing
11		System Reset

S:

System Real Time messages are for synchronizing all he system in real time.

System Real Time messages can be sent at any time. messages which consist of two or more bytes may be to insert Real Time messages.

ing Clock (F8H)

system is synchronized with this clock, which is sent rate of 24 clocks/quarter note.

rt (from beginning of song) (FAH)

s byte is immediately sent when the PLAY switch on master (e.g. sequencer or rhythm unit) is pressed.

ntinue (FBH)

s is sent when the CONTINUE switch is hit. A uence will continue at the time of the next clock.

p (FCH)

s byte is immediately sent when the STOP switch is It will stop the sequence.

ive Sensing (FEH)

e of this message is optional, for either receivers or nsmitters. This is a "dummy" Status byte that is sent ry 300 ms (max), whenever there is no other activity MIDI. The receiver will operate normally if it never eives FEH. Otherwise, if FEH is ever received, it will pect to receive FEH or a transmission of any type every ms (max). If a period of 300 ms passes with no activ- the receiver will turn off the voices and return to rmal operation.

stem Reset (FFH)

is message initializes all of the system to the condition just having turned on power. The System Reset mes- e should be used sparingly, preferably under manual nmand only. In particular, it should not be sent auto- tically on power up.

TABLE VI

SYSTEM EXCLUSIVE MESSAGES

STATUS	DATA BYTES	DESCRIPTION
11110000	0iiiiiii	Bulk dump etc iiiiiii: identification
	(0*******)	Any number of bytes may be sent here, for any purpose, as long as they all have a zero in the most isgnificant bit.
	11110111	EOX: "End of System Exclusive"

NOTES:

1. iiiiiii: identification ID (0-127)
2. All bytes between the System Exclusive Status byte and EOX of the next Status byte must have zeroes in the MSB.
3. The ID number can be obtained from the MIDI committee. See Table VII.
4. In no case should other Status or Data bytes (except Real-Time) be interleaved with System Exclusive, regardless of whether or not the ID code is recognized.
5. EOX or any other Status byte, except Real-Time, will terminate a System Exclusive message, and should be sent immediately at its conclusion.

TABLE VII

MANUFACTURERS' ID NUMBERS

Sequential Circuits, Inc.	01H
Big Briar	02H
Octave/Plateau	03H
Moog Music	04H
Passport Designs	05H
Lexicon	06H
Oberheim	10H
Bon Tempi	20H
S.I.E.L.	21H
Kawai	40H
Roland	41H
Korg	42H
Yamaha	43H

กิตติกรรมประกาศ

ปริญญานิพนธ์ฉบับนี้สำเร็จลงได้ด้วยความร่วมมือเป็นอย่างดีจาก อาจารย์ สมยศ จุณณะบิยะ ซึ่งเป็นอาจารย์ที่ปรึกษาตลอดการทำปริญญานิพนธ์นี้ ตลอดจนอาจารย์ในภาควิชาวิศวกรรมโทรคมนาคม ที่ได้ให้คำแนะนำอันมีค่าและเป็นประโยชน์อย่างมาก ข้าพเจ้าขอขอบพระคุณคณาจารย์ทุกท่านที่ได้ช่วยเหลือสนับสนุนการทำปริญญานิพนธ์นี้ ซึ่งไม่สามารถกล่าวรายนามได้ครบถ้วนทุกท่านเป็นอย่างสูงมาไว้ ณ ที่นี้

เอกสารอ้างอิง

ก. เอกสารอ้างอิงที่เป็นภาษาไทย

1. ชูชัย ธนสารตั้งเจริญ, "การใช้งาน Z80", พิสิกส์เซ็นเตอร์, 284 หน้า
2. บุญเลิศ เอี่ยมทัศนา, "โปรแกรมคอมพิวเตอร์ภาษาซี", ซีเอ็ดยูเคชั่น, 301 หน้า, 2529

ข. เอกสารอ้างอิงที่เป็นภาษาอังกฤษ

1. James W. Coffron, "The IBM PC Connection"
2. Murray Sargent III and Richard L. Shoemaker, "The IBM Personal Computer From the Inside Out", Addison-Wesley Publishing Company, Inc., 483 p., 1984
3. Borland International, Inc., "Turbo C", 300 p., 1987
4. Blaise Computing Inc., "Turbo C Tools", 319 p., 1987
5. Herbert Schildt, "Advance TURBO C", Osborne McGraw-Hill, 397 p., 1987
6. Craig Anderton, "MIDI for Musicians.", Amsco Publications, 105 p., 1986