คอมพิวเตอร์ในงานดนตรี

(Computer & Music)

โดย

นาย จิรพัฒน์ จันทร์เจิดศักดิ์
นาย วีระ นพนิราพาธ
Jirapat Janjerdsak
Wera Nopnirapath

อาจารย์ที่ปรึกษา
ผู้ช่วยศาสตราจารย์ ดรรชิต ไมตรี
Advisor

ปริญญานิพนธ์สำหรับปริญญาวิศวกรรมศาสตร์บัณฑิต
สาขาวิศวกรรมคอมพิวเตอร์
คณะวิศวกรรมศาสตร์
สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปริญญานิพนธ์ปีการศึกษา 2532

ภาควิชา วิศวกรรมคอมพิวเตอร์

คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

เรื่อง คอมพิวเตอร์ในงานดนตรี

ผู้จัดทำ

1.นาย จิรนันท์ จันทร์เจิดศักดิ์ 291025

2.นาย วีระ นพนิราพาธ 291186

..... อาจารย์ที่ปรึกษา

(ผศ. ครรชิต ไมตรี)

คอมพิวเตอร์ในงานดนตรี

จิรพันธ์ จันทร์เจิดศักดิ์
วีระ นพนิราพาช
ผศ.ดรรชิต ไมตรี อาจารย์ที่ปรึกษา
ปีการศึกษา 2532

บทคัดย่อ

วิทยานิพนธ์ฉบับนี้เสนอการนำเครื่องคอมพิวเตอร์มาช่วยในงานศิลปทางด้านการดนตรี ซึ่งเป็นงานปราณีตและต้องใช้เครื่องมือหรืออุปกรณ์ที่มีความละเอียดอ่อนสูง โดยมีจุดประสงค์ให้เป็นประโยชน์กับผู้ที่ต้องการศึกษาทางด้านดนตรี หรือช่วยงานผู้ที่มีความชำนาญในงานด้านนี้อยู่แล้ว ในลักษณะที่ผู้ใช้สามารถเขียนโน้ตดนตรีสากลผ่านทางเครื่องคอมพิวเตอร์ แล้วส่งข้อมูลไปยังเครื่องดนตรีดิจิตอลต่างๆเพื่อทำการสร้างเสียงจากโน้ตเหล่านั้น ในส่วนที่เป็นโปรแกรมการรับโน้ตจากผู้ใช้เข้ามานั้นก็ได้สอดแทรกทฤษฎีดนตรีที่จำเป็นเพื่อเพิ่มความฉลาดให้กับโปรแกรมทำให้สามารถอำนวยความสะดวกให้กับผู้ใช้ได้อย่างมาก และยังใช้การเขียนโปรแกรมภายใต้ไมโครซอฟต์วินโดว์ที่ได้ชื่อว่ามีส่วนเชื่อมต่อกับผู้ใช้ได้ดีที่สุด ทำให้การใช้งานโปรแกรมสะดวกยิ่งขึ้น ในด้านการเชื่อมต่อก็ได้ทำการศึกษามาตรฐานการเชื่อมต่อระหว่างเครื่องดนตรีดิจิตอลที่เรียกว่า "มิดิ"(MIDI - Musical Instrument Digital Interface) และได้สร้างการ์ดที่สามารถเปลี่ยนข้อมูลที่เป็นข้อมูลของเครื่องคอมพิวเตอร์ให้เป็นข้อมูลที่เป็นมาตรฐานของมิดิส่งออกไปทางสายสัญญาณมิดิ พร้อมทั้งสามารถแปลงข้อมูลในทางกลับกันได้ รายละเอียดต่างๆของทั้งส่วนที่เป็นฮาร์ดแวร์และซอฟต์แวร์ได้นำเสนออย่างละเอียดแล้วในวิทยานิพนธ์ฉบับนี้

Computer & Music

Jirapat Janjeardsak

Wera Nopnirapath

Professor Assistant Kanchit Maitri Advisor

Abstract

This thesis present an application of computer in musical field,the skillful field,that have to accompany with precise and delicate instruments.Intend to be utilized by novice musician in the new progerssion of learning or even the professional musician can use this application to alliviate their works.This applicaton was designed to allow user "select and place" notes in autometic staffs and dispatch its to digital instruments.In the program part,we run it under Microsoft Windows ,the eminent user interface method ,and we use the necessary musical rules to comfort the user in placing notes. In the interface part, our card use the MIDI standard in communication with the instruments.Another detail of this project already proposed in the following chapter.

สารบาญ

บทที่ 1
ความสำคัญและที่มา

เครื่องดนตรีเป็นเครื่องมือทางศิลปที่มีความละเอียดอ่อน เป็นสิ่งที่แสดงถึงความเจริญทางด้านจิตใจของมนุษย์ วัสดุที่มีนำมาทำเครื่องดนตรีจึงถูกเลือกสรรแล้วว่าดีที่สุด ไม่ว่าจะเป็นไม้ที่ผ่านกระบวนการเอาส่วนประกอบของน้ำออกให้ได้มากที่สุดเพื่อที่จะนำมาทำไวโอลินที่มีเสียงที่ดีที่สุด หรือจะเป็นการนำหนังแพะอย่างดีมาทำเป็นซอ ดังนั้นเมื่อเริ่มเข้ายุคของไฟฟ้าอิเลคทรอนิคส์ก็ได้เริ่มมีการประดิษฐ์เครื่องดนตรีที่ประกอบขึ้นมาด้วยวงจรอิเลคทรอนิคส์ต่างๆ เช่น วงจรประเภทวงจรประเภทออสซิเลเตอร์ (Oscillator) ฟิลเตอร์ (Filter) โดย ดร.โรเบิร์ด มอก และดร.แมกซ์ แมธิว จากห้องทดลองทรูแมนเบิร์กและห้องทดลองเบลในสหรัฐอเมริกา [1] จึงอาจถือว่าเป็นจุดเริ่มแห่งเครื่องดนตรีเครื่องดนตรีที่เป็นเครื่องอิเลคทรอนิคส์จึงเริ่มถือกำเนิดตั้งแต่นั้นมา

จากนั้นการพัฒนาของเครื่องดนตรีก็มีมาอย่างต่อเนื่อง จนกระทั่งเริ่มมีการนำระบบดิจิตอลเข้ามาใช้งานในเครื่องดนตรี โดยการใช้วงจรที่เรียกว่า ดีทูเอ (D/A - Digital to Analog Convertion) และ เอทูดี A/D (Analog to Digital Convertion) เป็นตัวช่วยแปลงสัญญาณไปกลับระหว่างสัญญาณดิจิตอลและสัญญาณอนาลอก ทำให้มีรูปแบบของข้อมูลทางด้านเสียงที่เป็นตัวตน สามารถมีการเก็บในลักษณะของสัญญาณดิจิตอลและเล่นกลับในลักษณะของสัญญาณอนาลอกได้ ความคิดที่จะมีทำการสื่อสารระหว่างเครื่องดนตรีที่มีระบบดิจิตอลด้วยกัน จึงเริ่มมีขึ้น ด้วยการตกลงของบริษัทเครื่องดนตรีชั้นนำหลายบริษัทก็ได้ออกข้อกำหนดมาตรฐานของการสื่อสารระหว่างเครื่องดนตรีดิจิตอลขึ้น โดยให้ชื่อมาตรฐานนี้ว่า "มิดิ" (MIDI - Musical Instrument Digital Inerface) ซึ่งมีความสามารถที่จะทำให้เครื่องดนตรีที่เป็นดิจิตอลด้วยกันสามารถติดต่อสื่อสารกันได้ โดยผ่านทางมิดิพอร์ท (มิดิอิน มิดิเอาท์ มิดิทรู) ลักษณะของการสื่อสารก็เช่น เครื่องดนตรีชิ้นที่ 1 สั่งให้เครื่องดนตรีชิ้นที่ 2 เล่นโน๊ตตัว "C" ด้วยความดัง 80 เปอร์เซ็นต์ ซึ่งความสามารถของมาตรฐานนี้สามารถทำให้มีการสื่อสานของเครื่องดนตรีได้ถึง 16 ชิ้นพร้อมๆกัน โดยใช้สายสัญญาณเพียงเส้นเดียว

ในทางด้านเทคโนโลยีคอมพิวเตอร์ โดยธรรมดาคนส่วนมากมักจะมองงานทางด้านคอมพิวเตอร์ว่าเป็นงานที่มีแต่ตัวเลข โปรแกรม งานที่น่าเบื่อยุ่งยากซับซ้อน ไม่น่ารื่นรมย์ แต่นับตั้งแต่มีการพัฒนาเครื่องดนตรีดิจิตอลและมาตรฐานมิดิขึ้นมา คอมพิวเตอร์ก็เริ่มเข้ามามีบทบาทในช่วยงานด้านศิลปการดนตรี โดยอาจทำหน้าที่เป็นตัวจัดลำดับโน๊ต (Sequencer) หรือเป็นตัวควบคุมการเล่นของเครื่องดนตรีพร้อมกันหลายๆชิ้น และในขณะนี้คอมพิวเตอร์ในงานด้านดนตรีก็มีแนวโน้มที่จะทำหน้าที่เป็นเครื่องช่วยในการประพันธ์เพลง (Composer) มากขึ้นซึ่งบริษัท

คอมพิวเตอร์ที่มีชื่อเสียงในงานด้านนี้ก็เช่นบริษัทอะตาริ บริษัทอะมิกา ลักษณะของการนำคอมพิวเตอร์ไปช่วยในงานประพันธ์เพลงก็คือ การทำหน้าที่เป็นโน๊ตอิดิเตอร์ (Note Editor) ให้ผู้ใช้สามารถที่จะเขียนตัวโน๊ตลงไปบนจอคอมพิวเตอร์ แล้วทดลองฟังเสียงจากโน๊ตที่เขียนเข้าไปทางเครื่องดนตรีดิจิตอลต่างๆจากนั้นจึงทำการแก้ไข แล้วทดลองฟังใหม่จนกว่าจะได้ผลเป็นที่พอใจ ทำให้ประหยัดเวลากว่าการที่จะนำไปให้นักดนตรีจริงๆทดลองเล่นแล้วจึงหาที่ผิดและแก้ไข การที่จะเขียนโปรแกรมทางด้านนี้จำเป็นต้องอาศัยทฤษฎีพื้นฐานทางด้านดนตรีเข้ามาประกอบด้วย โดยเฉพาะในส่วนของโน๊ตอิดิเตอร์

การที่จะทำให้คอมพิวเตอร์สามารถสื่อสารกับเครื่องดนตรีดิจิตอลได้ จำเป็นต้องมีการแปลงสัญญาณจากเครื่องคอมพิวเตอร์ไปเป็นสัญญาณที่เป็นมาตรฐานมิดิ โดยอาศัยการทำงานของฮาร์ดแวร์ที่เพิ่มเข้าไป ซึ่งอาจเป็นฮาร์ดแวร์เพิ่มเติมที่มีขายในท้องตลาด เช่น MPU-401 ของบริษัทโรแลนด์ ก็เป็นฮาร์ดแวร์ที่การออกแบบซอฟต์แวร์ส่วนใหญ่ยึดเป็นมาตรฐาน หรือ อาจจะเป็นฮาร์แวร์ที่สร้างขึ้นมาด้วยรูปแบบของตัวเอง เพียงแต่จะต้องสามารถสร้างและรับข้อมูลมิดิได้ เช่นในส่วนฮาร์ดแวร์ของโครงงานนี้

งานหลักที่ทำมาในโครงงานในเทอมนี้ เริ่มจากการศึกษารวบรวมข้อมูลและแนวความคิดของนักดนตรีในเรื่องความต้องการการปรับปรุงความสามารถของเครื่องดนตรีต่างๆ จากนักดนตรีในจุดที่สามารถนำคอมพิวเตอร์เข้ามาช่วยได้ โดยจากเฉพาะผู้ที่อยู่ในวงการดนตรีดิจิตอล และศึกษาจากซอฟต์แวร์ทางด้านนี้ที่เป็นผลิตภัณฑ์ในท้องตลาด เพื่อนำมาใช้ในการออกแบบทิศทางของชิ้นงานที่ต้องการพัฒนาขึ้นมาจากนั้นได้ทำการศึกษาการเชื่อมต่อแบบ MIDI, การศึกษาการเขียนโน๊ต และทฤษฎีดนตรีพื้นฐาน จากการศึกษาพบว่านักดนตรีส่วนใหญ่ยังไม่สามารถมองภาพดนตรีให้เป็นรูปร่างตัวโน๊ตได้ จึงมีแนวความคิดที่ว่าน่าจะใช้คอมพิวเตอร์เชื่อมโยงรูปภาพที่เป็นโน๊ตดนตรีกับเสียงดนตรีให้ได้ใกล้เคียงกันที่สุด ในขณะที่นักดนตรีเห็นภาพที่เป็นโน๊ตดนตรี ก็สามารถได้ยินเสียงเพลงไปได้พร้อมกัน โดยมีความสามารถต่างๆที่นักดนตรีต้องการ เช่น ความสามารถในการเล่นเครื่องดนตรีหลายๆชิ้นพร้อมๆกัน สามารถเปลี่ยนเครื่องดนตรีที่ใช้บรรเลงตามใจชอบได้ และง่ายต่อการใช้งาน

โครงงานนี้จึงจัดทำขึ้นมาเพื่อเป็นแนวทางในการพัฒนาโปรแกรมที่สามารถใช้ประโยชน์ที่กล่าวมาข้างต้นได้อย่างมีประสิทธิภาพและเป็นประโยชน์แก่นักดนตรีและผู้ที่สนใจให้มากที่สุด

บทที่ 2
ทฤษฎีและแนวความคิด

2.1 การสื่อสารแบบมิดิ

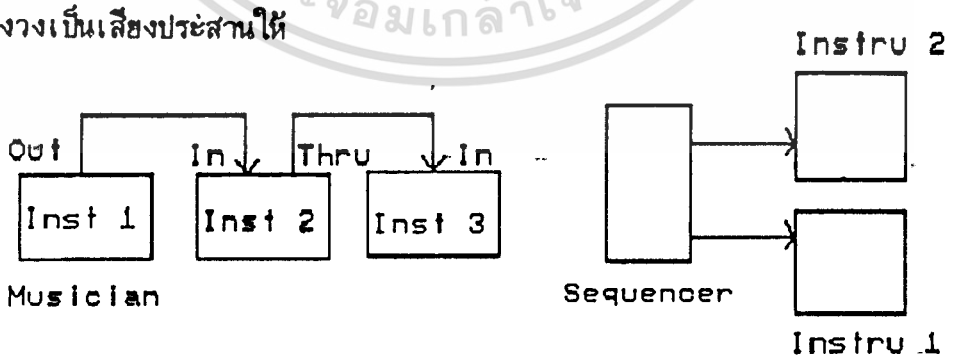
2.1 ความสำคัญและความหมาย

การสื่อสารแบบมิดินี้มีความสัมพันธ์กับเครื่องดนตรีอิเลคทรอนิคส์มาก เพราะเครื่องดนตรีที่สามารถทำการสื่อสารทางมิดิได้ต้องเป็นเครื่องดนตรีอิเลคทรอนิคส์เท่านั้น ลักษณะที่สำคัญของเครื่องดนตรีประเภทนี้คือ จะใช้วงจรอิเลคทรอนิคส์ประเภทวงจรสร้างความถี่ (Oscillator) ,ตัวกรองสัญญาณ (Filter) หรือ อุปกรณ์ที่ทำงานเกี่ยวกับคลื่น (Wave Shaping) มาทำงานร่วมกัน โดยอาจมีอุปกรณ์ที่เกี่ยวกับหน่วยความจำมาเก็บเสียงที่สร้าง (Synthesis) หรือเก็บตัวอย่าง (Sampling) มาทำงานร่วมด้วยก็ได้ เมื่อข้อมูลทางเสียงดนตรีอยู่ในรูปของข้อมูลทางด้านดิจิตอลแล้วก็สามารถนำไปใช้งานได้หลายอย่าง เช่น การดัดแปลงแก้ไข การส่งข้อมูลไปให้เครื่องดนตรีชิ้นอื่นเล่น ได้ง่ายขึ้น มาตรฐานที่ใช้ในการติดต่อหรือภาษาของเครื่องดนตรีจึงต้องถูกกำหนดขึ้น และถูกเรียกว่า "มิดิ" (MIDI - Musical Instrument Digital Interface)

2.2 ประโยชน์ของมิดิ

ประโยชน์ของการสื่อสารแบบมิดินี้สามารถแบ่งเป็นหัวข้อย่อยๆได้ 4 หัวข้อ คือ

2.2.1 สามารถทำให้เครื่องดนตรีหลายชิ้นเล่นพร้อมกันได้ ทำให้นักดนตรีคนเดียวสามารถเล่นเครื่องดนตรีได้หลายชิ้นพร้อมกัน โดยอาจตั้งให้เครื่องดนตรีเครื่องหนึ่งเป็นเสียงหลักและอีกหลายๆเครื่องเป็นเสียงประสาน เช่น นักดนตรีอาจนั่งเล่นเปียโนไฟฟ้าในขณะที่มีออเคสตร้าทั้งวงเป็นเสียงประสานให้

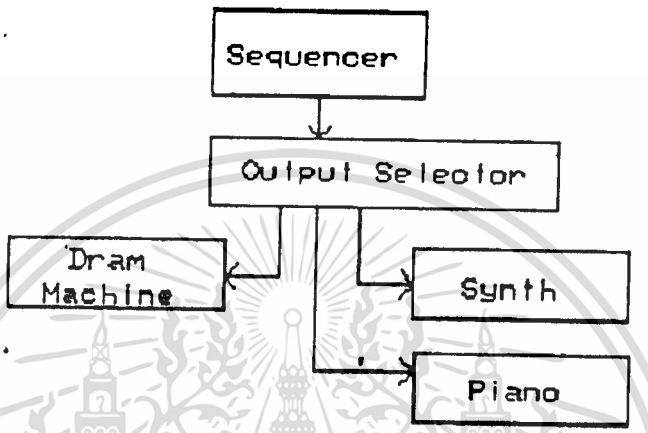


รูป 2.1 และ 2.2 การใช้งานเครื่องดนตรีหลายเครื่องพร้อมกัน

2.2.2 ทำให้เครื่องดนตรีต่างชนิดกันสามารถติดต่อใช้งานร่วมกันได้ เช่น เมื่อนำเปียโนไฟฟ้าเป็นเครื่องแม่และให้เครื่องสร้างเสียงกลอง (Drum Machine) เป็นเครื่องลูก ก็อาจจะทำการตั้งให้ว่า เมื่อกดที่เปียโนไฟฟ้าเป็นเสียง 'C' ก็ให้เครื่องสร้างเสียงกลองเล่นเสียง

กลองใหญ่ (Drum Bass) ออกมา

2.2.3 การจัดลำดับการเล่นโน๊ตของการเล่นแบบประสานเสียง โดยใช้ตัวจัดลำดับโน๊ต (Sequencer) จากรูป 2.2 จะเห็นว่าเครื่องดนตรีหลายชิ้นสามารถเล่นเพลงเดียวกันได้โดยใช้ตัวจัดลำดับโน๊ตเพียงตัวเดียว โดยที่ตัวจัดลำดับโน๊ตอาจสั่งให้เปียโนเล่นคอร์ด ในขณะที่ซินธีไซเซอร์เล่นเมโลดี้ เครื่องสร้างเสียงกลองเล่นกลอง



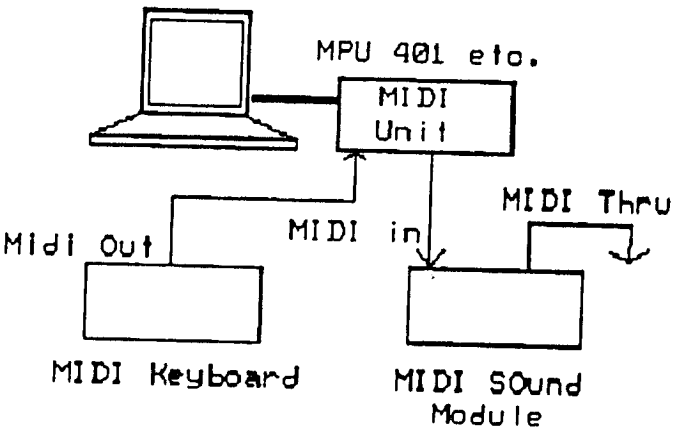
รูปที่ 2.3 การใช้ตัวจัดลำดับโน๊ตควบคุมเครื่องดนตรีชิ้นอื่น

2.2.4 การนำคอมพิวเตอร์เข้ามาควบคุมเครื่องดนตรี เป็นการนำเครื่องคอมพิวเตอร์มาทำหน้าที่เป็นตัวจัดลำดับโน๊ต (Sequencer) หรือเป็นตัวสร้างเพลง (Composer) แทนอุปกรณ์ที่สร้างขึ้นมาเฉพาะหน้าที่ การใช้คอมพิวเตอร์มาสื่อสารในระบบมิดินี้จำเป็นต้องมีการแปลงข้อมูลไปมาระหว่างข้อมูลที่เป็นมาตรฐานมิดิ กับข้อมูลที่เป็นของคอมพิวเตอร์ ซึ่งฮาร์ดแวร์ในส่วนนี้ได้ทำเป็นมาตรฐานวางขายอยู่ทั่วไป เช่น MPU 401 ของบริษัทโรแลนด์ เป็นตัวอินเตอร์เฟสที่ได้รับความนิยมกันเป็นอย่างมาก ข้อดีที่สามารถนำคอมพิวเตอร์เข้ามามีบทบาทในทางดนตรีได้ก็คือความยืดหยุ่นของตัวโปรแกรมที่มีสามารถนำไปทำงานบนเครื่องมีมาก (สามารถทำให้คอมพิวเตอร์เป็นอุปกรณ์ทางดนตรีได้หลายชนิด) และหน่วยความจำที่มีอยู่ก็มีมากพอที่จะบรรจุตัวโน๊ตเข้าไปได้มาก ควบคุมเครื่องดนตรีได้มากชิ้นและมีประสิทธิภาพ

2.3 การสื่อสารแบบมิดิ

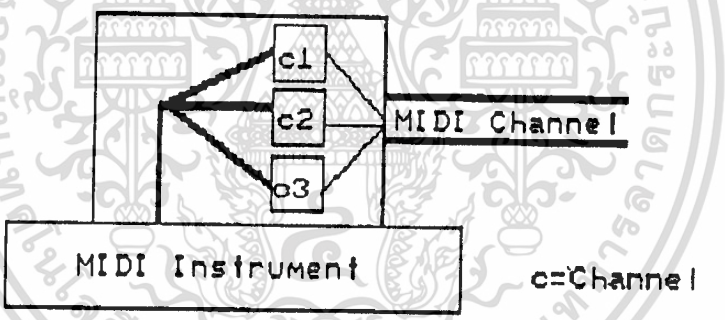
ข้อมูลที่ใช้สื่อสารในระบบมิดิมีหลายประเภทแต่ก่อนที่จะกล่าวถึงข้อมูลเหล่านั้นมีคำศัพท์ที่จำเป็นต้องทำความเข้าใจก่อน คือ

แชนแนล (Channal) ในมาตรฐานมิดิสามารถส่งข้อมูลได้พร้อมกันถึง 16 แชนแนล นั่นหมายถึง สามารถส่งข้อมูลไปให้เครื่องดนตรี 16 ชิ้นเล่นได้พร้อมๆกัน ข้อมูลที่เป็นข้อมูลแชนแนล



รูปที่ 2.4 การใช้คอมพิวเตอร์สื่อสารกับเครื่องดนตรี

แต่ละตัวสามารถกำหนดได้ว่าเป็นข้อมูลของแชนแนลใด ส่วนทางเครื่องดนตรีที่เป็นตัวรับก็สามารถที่จะกำหนดได้ว่าจะฟังข้อมูลจากแชนแนลใดบ้าง หรืออาจฟังพร้อมกันทุกแชนแนลก็ได้ หลักการนี้คล้ายกับเครื่องรับโทรทัศน์ที่สามารถเลือกที่จะให้สัญญาณของสถานีใดที่มีอยู่หลายๆสัญญาณปนกันออกมาได้



รูปที่ 2.5 การสื่อสารแบบหลายแชนแนล

โหมด (Mode) โหมดต่างๆของเครื่องดนตรีแบ่งออกได้เป็น

1. โหมด 1 (Omni - On,Poly) เครื่องดนตรีที่ถูกตั้งเป็นโหมดนี้จะรับฟังข่าวสารจากทุกแชนแนล(Omni-On) และสามารถเล่นได้ครั้งละหลายโน๊ตพร้อมกัน (Polyphony)

2. โหมด 2 (Omni - On,Mono) เครื่องดนตรีที่ถูกตั้งเป็นโหมดนี้จะรับฟังข่าวสารจากทุกแชนแนล(Omni-On) และสามารถเล่นได้ครั้งโน๊ตเท่านั้น (Monophony)

3. โหมด 3 (Omni - Off,Poly) เครื่องดนตรีที่ถูกตั้งเป็นโหมดนี้จะรับฟังข่าวสารจากแชนแนลที่ได้ตั้งไว้เพียงแชนแนลเดียวเท่านั้น แต่สามารถเล่นได้ครั้งละหลายโน๊ตพร้อมกัน

4. โหมด 4 (Omni - Off,Mono) เครื่องดนตรีที่ถูกตั้งเป็นโหมดนี้จะรับฟังข่าวสารจากแชนแนลที่ได้ตั้งไว้เพียงแชนแนลเดียวเท่านั้น และยังสามารถเล่นได้ทีละโน๊ตเดียวเท่านั้น

ทั้ง 4 โหมดสามารถสรุปได้เป็นตารางดังนี้

	Poly	Mono
Omni On	Mode 1	Mode 2
Omni Off	Mode 3	Mode 4

รูปที่ 2.6 ตารางโหมด

2.4 รูปแบบของข้อมูลมิดิ

ข้อมูลมิดิ หมายถึงข้อมูลต่างๆที่เครื่องดนตรีใช้ติดต่อสื่อสารกัน แบ่งออกได้เป็น 2 ส่วนใหญ่ๆ คือ

2.4.1 ข้อมูลที่มีการกำหนดแชนแนล (Channel Message) เป็นข้อมูลที่ส่งออกไปพร้อมๆกันกับหมายเลขแชนแนล ยังสามารถแบ่งย่อยออกไปได้อีก คือ

2.4.1.1 ข้อมูลเสียง (Voice Message)

- ข้อมูลที่เกี่ยวกับโน๊ต จะเป็นข้อมูลที่บอกว่าเป็นโน๊ตตัวใด
- ข้อมูลเปลี่ยนโปรแกรม (Program Change) เป็นข้อมูลที่บอกให้เครื่องดนตรีที่เป็นตัวรับว่าให้เล่นเป็นเสียงอะไร เช่น ถ้าตัวรับเป็นซินธีไซเซอร์ก็จะสามารถใช้ข้อมูลนี้สั่งให้ซินธีไซเซอร์เปลี่ยนเป็นเสียงที่ซินธีไซเซอร์สามารถสร้างเสียงใดก็ได้
- ข้อมูลเปลี่ยนการควบคุม (Control Change) เป็นข้อมูลที่ทำให้เสียงที่ออกมาแตกต่างๆไปจากธรรมดาเล็กน้อย อาจจะเป็นเสียงสั่น เสียงห้วน ซึ่งจะไม่มีในเครื่องดนตรีทุกชนิด ส่วนมากจะเป็นเครื่องดนตรีประเภทเปียโนไฟฟ้า
- ข้อมูลหลังการกดคีย์ (After Touch) เป็นข้อมูลที่เกิดขึ้นหลังจากการกดคีย์ของเครื่องดนตรีประเภทคีย์บอร์ด ข้อมูลเหล่านี้ก็เช่น แรงที่กดคีย์ ความเร็วที่กด
- ข้อมูลการเปลี่ยนแปลงความถี่เสียง (Pitch Bender) เป็นข้อมูลที่เกิดขึ้นเนื่องจากใช้ ตัว Pitch Bender ที่อยู่บนเครื่องดนตรีประเภทคีย์บอร์ด จะทำให้ความถี่ของโน๊ตที่กดอยู่ตอนนั้นค่อยๆสูงขึ้นหรือต่ำลง

2.4.1.2 ข้อมูลโหมด (Mode Message)

เป็นการส่งข้อมูลในการเปลี่ยนโหมดเพื่อทำการเปลี่ยนโหมดของตัวรับ

2.4.2 ข้อมูลที่ไม่มีการกำหนดแชนแนล (System Message) เป็นการส่งข้อมูลไปยังอุปกรณ์ทุกชิ้นที่ต่ออยู่ในข่ายการสื่อสารของมิดิ ข้อมูลเหล่านี้ก็คือ

2.4.2.1 ข้อมูลในการซิงโครไนซ์และควบคุมเวลาของระบบ (System Synchronization) เรียกข้อมูลนี้ว่า ข่าวสารระบบควบคุมเวลา จริง (System Real Time message) สามารถแบ่งย่อยออกได้ เป็น

- ข้อมูลสัญญาณนาฬิกา (Timing Clock Message) ใช้ในการซิงโครไนซ์อุปกรณ์ต่างๆในระบบ เช่น เครื่องสร้างเสียงกลอง เครื่องจัดลำดับตัวโน๊ต ซึ่งข้อมูลนี้จะถูกส่งออกไป 24 ครั้งใน 1 ควอเตอร์โน๊ตทุก
- ข้อมูลบอกการเริ่มต้น (Start Message) เป็นข้อมูลที่บอกให้อุปกรณ์ทุกชิ้นในเครือข่ายมิดิเริ่มทำงาน
- ข้อมูลต่อเนื่อง (Continuous Message) เป็นข้อมูลที่บอกให้อุปกรณ์ทำงานที่สัญญาณนาฬิกาถัดไปหลังจากที่หยุดทำงาน
- ข้อมูลบอกการสิ้นสุด (Stop Message) เป็นข้อมูลที่บอกให้อุปกรณ์ทุกชิ้นในระบบหยุดทำงาน
- ข้อมูลบอกการเริ่มต้นใหม่ (System Reset Message) เป็นการเริ่มต้นทำงานใหม่ทั้งหมดเหมือนการเปิดเครื่องใหม่

2.4.2.2 ข้อมูลระบบปกติ (System Common Message) ใช้กำหนดการทำงานและหน้าที่ของอุปกรณ์ต่างๆในระบบ แบ่งย่อยได้ดังนี้

- ข้อมูลบอกตำแหน่งเพลง (Song Position Pointer Messsge) เป็นการกำหนดการนับจังหวะ (Beat Counter) ของอุปกรณ์ให้มีค่าตามที่ต้องการ
- ข้อมูลเลือกเพลง (Song Select Message) เป็นข้อมูลที่ใช้ในการกำหนดเพลงที่จะเล่น สามารถเลือกได้ถึง 128 เพลง
- ข้อมูลการขอปรับแต่ง (Tune Request Message) ใช้กับซินธีไซเซอร์แบบอนาลอก เป็นการขอปรับแต่งวงจรสร้างความถี่

2.4.2.3 ข้อมูลระบบพิเศษ (System Executive Message) เป็นการกำหนดเลขหมายที่นอนให้กับเครื่องดนตรี (Manufacturer' ID Number) เพื่อให้เครื่องดนตรีหลายๆยี่ห้อสามารถร่วมเล่นกันได้

2.2 <u>ทฤษฎีพื้นฐานทางด้านดนตรี</u>

ในการเขียนโปรแกรมที่เกี่ยวกับงานทางด้านดนตรี ไม่ว่าจะเป็นส่วนของโน๊ตอิดิเตอร์

หรือส่วนของแคสเซทเพลย์เยอร์นั้น จำเป็นที่จะต้องมีความรู้พื้นฐานเกี่ยวกับทฤษฎีทางดนตรีทั้งในส่วนที่เป็นทฤษฎีดนตรีจริงๆและในส่วนที่เป็นหลักการที่เกี่ยวข้องกับเครื่องดนตรีอิเลคทรอนิคส์ ในส่วนนี้จึงจะขอกล่าวถึงความรู้พื้นฐานทางด้านดนตรีที่มีความจำเป็นในการทำโครงงานนี้

2.2.1 โน๊ตและตัวหยุด

ตัวโน๊ตเป็นสัญญลักษณ์ที่ใช้แทนเสียงที่ความยาวของเสียงต่างกัน ซึ่งในระบบสากลจะมีตัวโน๊ตอยู่ 7 ตัว เมื่อเทียบกับคีย์ของเปียโนจะมีการวางตัวดังนี้

คีย์สีดำที่อยู่ตรงกลางเป็นคีย์ครึ่งเสียง หมายถึงจะมีเสียงที่สูงขึ้นกว่าคีย์สีขาวที่อยู่ก่อนหน้ามันครึ่งเสียง และมีเสียงต่ำกว่าคีย์สีขาวที่อยู่ข้างหลังมันครึ่งเสียง จะเห็นได้ว่าระหว่าง E กับ F และ B กับ C จะไม่มีคีย์สีดำคั่น หมายถึงมีความแตกต่างของเสียงเพียงครึ่งเดียวเมื่อเทียบกับโน๊ตตัวอื่น ตั้งแต่ C ตัวหนึ่งถึง C อีกตัวหนึ่งเรียกว่า 1 ออคเทฟ (Octave) ตัวโน๊ตแต่ละตัวสามารถแทนด้วยสัญญลักษณ์ต่างๆดังนี้

ชื่อโน๊ต	ลักษณะตัวโน๊ต	ความยาว
โน๊ตเขบ็ต 3 ชั้น		1/8
โน๊ตเขบ็ต 2 ชั้น		1/4
โน๊ตเขบ็ต 1 ชั้น		1/2
โน๊ตตัวดำ		1
โน๊ตตัวขาว		2
โน๊ตตัวกลม		4

สามารถแสดงความสัมพันธ์ของความยาวของต่างๆออกเป็นแผนภาพต้นไม้ได้ดังรูปที่ 2.7 นอกจากจะมีสัญญลักษณ์ของโน๊ตแล้วยังมีสัญญลักษณ์ของตัวหยุดเสียงต่างๆดังนี้

ลักษณะตัวหยุด	ความยาว
𝄾	1/8
𝄿	1/4
𝅀	1/2
𝄻	1
𝄼	2
𝄼	4

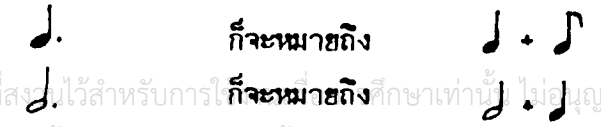
รูปที่ 2.7 แผนภาพต้นไม้ของตัวโน๊ต

ลักษณะการเชื่อมโน๊ต

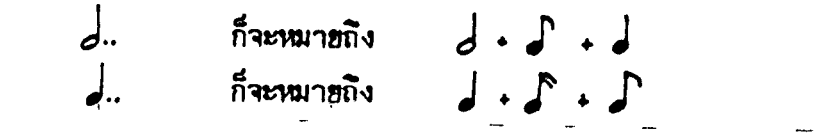
เมื่อมีโน๊ตชเบ็จมาอยู่ติดกันเกิน 1 ตัวมักนิยมมีการรวบหาง (Stem) ของตัวโน๊ตเข้าด้วยกัน เช่น

การประจุดตัวโน๊ต

เมื่อตัวโน๊ตที่แสดงบนบรรทัด 5 เส้นมีจุดเล็กๆอยู่ที่ข้างหลังตัวโน๊ต หมายถึง โน๊ตตัวนั้นจะมีความยาวเพิ่มขึ้นครึ่งเสียงของตัวมันเอง เช่น



ถ้ามีการประ 2 จุดก็จะเพิ่มความยาวด้วยครึ่งหนึ่งของตัวมันก่อน จากนั้นก็เพิ่มขึ้นอีกครึ่งหนึ่งของโน๊ตที่มีการแบ่งครึ่งแล้ว ตัวอย่างเช่น



2.2.2 บรรทัด 5 เส้นและส่วนประกอบ

การที่จะบอกว่าโน๊ตตัวนั้นมีค่า (เสียงสูง ต่ำ) เป็นอย่างไร จำเป็นต้องมีสเกลบอก ซึ่งสเกลที่ใช้บอกความค่าของโน๊ตต่างๆนี้เรียกว่า บรรทัด 5 เส้น (Staff)

รูปที่ 2.8 ลักษณะของบรรทัด 5 เส้น

การวางโน๊ตลงบนบรรทัด 5 เส้นจะวางระหว่างเส้นหรือบนเส้นโดยสามารถมีเส้นเล็กๆที่อยู่บนหรือล่างบรรทัด 5 เส้นได้อีกสำหรับโน๊ตที่มีระดับเสียงสูงหรือต่ำกว่าบรรทัด 5 เส้นไป เรียกว่า เส้นน้อย โน๊ตตัวแรกที่เห็นเรียกว่า ซีกลาง (Middle C) และไล่ขึ้นไปเรื่อยๆคือเป็น D E F G A B และกลับมาเป็น C อีก

ส่วนเส้นที่กั้นอยู่ภายในบรรทัดนั้นเรียกว่า เส้นกั้นห้อง (Bar line) เป็นเส้นที่ทำหน้าที่แยกโน๊ตของแต่ละห้องเสียง (Bar) ออกจากกัน โดยที่จำนวนจังหวะในแต่ละห้องจะถูกกำหนดด้วยสัญญลักษณ์ที่เรียกว่า เครื่องหมายกำหนดจังหวะ (Time Signature) ความหมายของเครื่องหมายกำหนดจังหวะต่างๆมีดังนี้

ตัวเลขข้างบนหมายถึง จำนวนจังหวะที่มีในแต่ละห้อง

ตัวเลขข้างล่างหมายถึง ลักษณะโน๊ตที่เป็นตัวกำหนดความเป็นหนึ่งจังหวะ

เช่น หมายถึง ในหนึ่งห้องจะมี 4 จังหวะและโน๊ตที่เป็นโน๊ตที่นับหนึ่งจังหวะคือโน๊ตตัวดำ ในลักษณะที่เป็น เช่นนี้เราอาจแทนด้วยตัวอักษร C (Common time) ได้ หากเป็น ก็หมายความว่าในหนึ่งห้องจะมีอยู่ 3 จังหวะโดยมี โน๊ตขเบ็จ 1 ชั้นเป็นโน๊ต 1 จังหวะ

สัญญลักษณ์ที่อยู่หน้าสุดของบรรทัด 5 เส้น เรียกว่า กุญแจประจำหลัก (Clef) มีทั้งหมด 6 อย่าง คือ

กุญแจทรีเบิ้ล (Treble Clef) หรือ กุญแจซอล

กุญแจเบส (Bass Clef) หรือ กุญแจฟา

กุญแจอัลโต (Alto Clef)

กุญแจเทเนอร์ (Tenor Clef)

กุญแจทรีเบิ้ลเอจวา (Treble 8 Va Clef)

กุญแจเบสเอจวา (Bass 8 Va Clef)

กุญแจเหล่านี้มีหน้าที่บอกว่าโน๊ตที่เส้นบนบรรทัด 5 เส้นใดเป็นเสียงใด เช่น โน๊ตตัวที่อยู่บนเส้นที่ 2 ของบรรทัด 5 เส้นที่ถูกกำกับด้วยกุญแจทรีเบิ้ลจะเป็นโน๊ต G (ซอล) โน๊ตตัวที่อยู่ในช่องที่ 1 ของบรรทัด 5 เส้นที่ถูกกำกับด้วยกุญแจเบสจะเป็นโน๊ต F (ฟา)

2.2.3 เครื่องหมายแปลงเสียง การที่จะทำให้ตัวโน๊ตมีเสียงที่สูงขึ้นหรือต่ำลงครึ่งเสียงนั้นจะต้องมีการเติมเครื่องหมายแปลงเสียงไปข้างหน้าโน๊ตตัวนั้น เครื่องหมายแปลงเสียงจะมี

Sharp [#] เป็นการทำให้เสียงของโน๊ตตัวนั้นสูงขึ้นครึ่งเสียง และโน๊ตตัวเดียวกันที่ตามมาทั้งหมดจะต้องมีเสียงสูงขึ้นไปครึ่งเสียงด้วย จนกระทั่งพบเครื่องหมายเนเชอรัล

Flat [♭] เป็นการทำให้เสียงของโน๊ตตัวนั้นต่ำลงครึ่งเสียง และโน๊ตตัวเดียวกันที่ตามมาทั้งหมดจะต้องมีเสียงต่ำลงไปครึ่งเสียงด้วย จนกระทั่งพบเครื่องหมายเนเชอรัล

Natural [♮] สำหรับทำให้โน๊ตที่ถูกยกหรือลดเสียงนั้นคืนสู่สภาพปกติ

2.2.4 เทมโป (Tempo) หมายถึง อัตราในการเคาะจังหวะในทางดนตรีต่อหนึ่งนาที โดยปกติจะเขียนในรูปของ[♩= 72] ก็จะมีความหมายว่าในหนึ่งนาทีจะสามารถเล่นโน๊ตตัวดำได้ 72 ตัว

นอกจากที่กล่าวมาแล้วรายละเอียดและสัญญลักษณ์เพิ่มเติมสามารถอ่านได้ในภาคผนวก ฉ.

2.3 ไมโครซอฟต์วินโดว์

เนื่องจากงานชิ้นนี้ เป็นโปรแกรมสำหรับผู้ใช้ที่ มักจะไม่ค่อยจะข้องแวะกับคอมพิวเตอร์เท่าใดนักส่วน ส่วนการติดต่อกับผู้ใช้ (User Interface) จึงเป็นเรื่องสำคัญเรื่องหนึ่ง ทั้งในเวลานี้ยังมีโปรแกรมแบบรวม (Integrated Enviroment) ซึ่งมี ส่วนติดต่อกับผู้ใช้ ให้ใช้จึงได้เลือกไมโครซอฟต์วินโดว์มาเป็นส่วนติดต่อกับผู้ใช้ ด้วยสาเหตุดังนี้

1. มี ส่วนติดต่อกับผู้ใช้ ที่นิยมมากพอที่จะเป็นมาตรฐาน น่าจะได้ผลดีกว่าการพัฒนาส่วนติดต่อกับผู้ใช้เองหรือใช้ตัวอื่นๆเอง

2. เสียเวลาพัฒนาเร็วกว่า และจากการที่ไม่ต้องพัฒนาส่วนติดต่อกับผู้ใช้เองโดยใช้ของสำเร็จที่มีอยู่แล้วทำให้ใช้เวลาน้อยลง

3. ได้ผลงานเป็นชิ้นเป็นอันและสามารถใช้งานได้ เนื่องจากความสามารถอื่นๆของไมโครซอฟต์ วินโดว์ เองซึ่งจะขออธิบายในส่วน ไมโครซอฟต์ วินโดว์ ซึ่งไม่ต้องมาพัฒนาเองทำให้นำเวลาไปพัฒนางานให้ดีขึ้นกว่าเดิม จนถึงขั้นใช้งานได้

2.3.1 ไมโครซอฟต์วินโดว์

การใช้ไมโครซอฟต์วินโดว์จะมีการทำงานพิเศษหลายๆอย่างดีและมากกว่าระบบทั่วๆไปใน เหตุนี้การเขียนโปรแกรมเพื่อใช้กับ ไมโครซอฟต์ วินโดว์ จึงดูค่อนข้างจะซับซ้อนกว่าเนื่องจาก การทำงานพิเศษเหล่านั้นของตัวไมโครซอฟต์วินโดว์เอง ซึ่งจะกล่าวได้ดังนี้

- ส่วนติดต่อกับผู้ใช้ (User Interface) แบบเสดงผลเป็นรูปภาพ อันมีช่องหน้าต่าง, เมนู, กรอบข้อความ (Dialog Boxes), การควบคุมพิเศษต่างๆ
- ระบบคิว(Queued)ของข้อมูลเข้า/ออก
- ระบบภาพที่ไม่ขึ้นต่ออุปกรณ์(Device Independent Graphics)
- จัดระบบงานแบบหลายงาน(Multitasking)
- การแลกเปลี่ยนข้อมูลกันเองระหว่างโปรแกรม (Data Interchange Between Application)

ความแตกต่างของการเขียนโปรแกรมระหว่างบนไมโครซอฟต์วินโดว์กับโปรแกรมธรรมดามีมากพอสมควร เช่น การทำงานต่างๆในโปรแกรมธรรมดา เช่น I/O, การจัดการแฟ้มข้อมูล, การจัดการหน่วยความจำ จะทำงานโดยโปรแกรมในไลบรารี่ (Libaries) และดอสซึ่งจะส่งและรับข้อมูลเป็นแบบส่งและรับตัวอักษร การเข้าถึงความจำก็ยังเป็นแบบธรรมดาอยู่ ส่วนในไมโครซอฟต์วินโดว์การทำงานจะต่างออกไปโปรแกรมใช้งานต่างๆนั้นจำเป็นจะต้องปันส่วนการใช้งานทรัพยากรภายในคอมพิวเตอร์กันรวมทั้งหน่วยประมวลผลกลางด้วย ซึ่งจะติดต่อผู้ใช้โดยผ่านอุปกรณ์หลายแบบอันมีจอรูปภาพ (Graphic Display) แป้นพิมพ์, เมาส์(Mouse) ซึ่งจะขออธิบายไว้เป็นส่วนๆตามหัวข้อเปรียบเทียบการทำงานพิเศษของไมโครซอฟต์ วินโดว์ดังต่อไปนี้

ส่วนติดต่อกับผู้ใช้ (User Interface)

ในระบบวินโดว์ (ในที่นี้หมายถึง ไมโครซอฟต์ วินโดว์) มีจุดประสงค์ที่จะให้ผู้ใช้มองเห็นการทำงานแต่ละส่วนให้มากที่สุดในเวลาเดียวกัน ในระบบงานแบบหลายงานแต่ละโปรแกรมก็จะมีส่วนหนึ่งของจอภาพเป็นของตัวเองโดยที่จะสามารถติดต่อกับจอภาพได้ตลอดเวลา แต่โปรแกรมจะปันส่วนจอภาพซึ่งกันและกันโดยใช้ช่องหน้าต่าง ซึ่งการใช้งานช่องหน้าต่างเหล่านี้ก็จะทำให้สามารถแทนสิ่งต่างๆได้มากมาย

ในการเขียนโปรแกรมในแบบธรรมดา โปรแกรมย่อยในไลบรารี่จะจัดการตระเตรียมจอภาพให้โดยส่งค่า Handle ซึ่งใช้สำหรับส่งข้อมูลออกจอภาพโดยโปรแกรมาย่อยในไลบารี่หรือดอสแต่ในระบบ ไมโครซอฟต์วินโดว์ จำเป็นต้องเขียนโปรแกรมที่สร้างช่องหน้าต่างของตนเองสำหรับแสดงข้อมูลหรือรับข้อมูลจากผู้ใช้ แต่ถ้าได้สร้างขึ้นครั้งหนึ่งแล้วระบบ วินโดว์จะจัดการเกี่ยวกับความต้องการของผู้ใช้ได้หลายอย่างเช่น เปลี่ยนขนาดของช่องหน้าต่างหรือเปลี่ยน

ตำแหน่งของช่องหน้าต่าง

ข้อดีอีกอย่างคือ ระบบวินโดว์ คือ โปรแกรมหนึ่งๆสามารถสร้างช่องหน้าต่างได้หลายช่อง และสามารถใช้ช่องหน้าต่างช่องนั้นๆได้เต็มที่

ระบบคิว (Queuel) ของข้อมูลเข้า-ออก

ส่วนที่แตกต่างกันมากที่สุดระหว่าง โปรแกรมธรรมดากับโปรแกรมบนระบบวินโดว์ ก็คือ ส่วนรับข้อมูลจากผู้ใช้ในระบบธรรมดานั้นจะรับจากหลายๆทาง เช่น อินพุทมาตรฐาน สแกนโค๊ด ของดอส โดยไบออส ข้อมูลจากเมาส์ไดรฟเวอร์ แต่ในระบบวินโดว์จะมีข้อมูลเหล่านี้โดยอัตโนมัติทันทีที่ช่องหน้าต่างนั้นถูกสร้างขึ้นมา ซึ่งรวบรวมมาจากหลายๆ อุปกรณ์ในแบบต่างๆกันให้มาอยู่ในรูป "ข่าวสารเข้า" (Input Message) ไม่ว่าจะเป็นรหัสตัวหนังสือ, สแกนโค๊ด, ตำแหน่งของเมาส์หรือข้อมูลรูปแบบอื่น จะมาในรูปแบบเดียวกันหมด คือ ข่าวสารเข้า

ระบบภาพที่ไม่ขึ้นกับอุปกรณ์

ในระบบวินโดว์จะสามารถทำงานกับระบบภาพที่ไม่ขึ้นต่ออุปกรณ์ได้ นั่นหมายถึงวาดภาพออกอุปกรณ์อื่นๆ เช่น เครื่องพิมพ์กราฟฟิก ได้เช่นเดียวกับวาดรูปบนจอ ในการนี้ระบบวินโดว์จำเป็นจะต้องอาศัยโปรแกรมควบคุมอุปกรณ์พิเศษสำหรับอุปกรณ์แต่ละชนิดซึ่งจะติดต่อกับอุปกรณ์อีกทีหนึ่ง

การจัดระบบงานแบบหลายงาน (Multitasking)

ระบบวินโดว์เป็นระบบงานแบบหลายงาน ซึ่งหมายถึงสามารถนำโปรแกรมหลายๆ โปรแกรมมาทำงานพร้อมกันได้ ซึ่งในระบบธรรมดานั้นจะไม่สามารถทำได้ เนื่องจากในระบบงานแบบธรรมดานั้นทรัพยากรทุกอย่าง ได้แก่ อุปกรณ์ I/O, จอภาพ, หน่วยความจำ หรือแม้กระทั่งหน่วยประมวลจะต้องให้เพียงโปรแกรมงานเพียงงานเดียวใช้ได้เท่านั้นแต่ในระบบวินโดว์ โปรแกรมงานแต่ละชิ้นจะต้องปันส่วนทรัพยากรแก่โปรแกรมอื่นๆ ที่กำลังทำงานอยู่ ด้วยเหตุนี้ระบบวินโดว์ จะต้องควบคุมทรัพยากรเหล่านี้โดยระมัดระวังโปรแกรมที่ทำงานบนระบบวินโดว์จึงจะต้องมีลักษณะพิเศษเพื่อให้ระบบสามารถควบคุมทรัพยากรได้ เช่น ในระบบธรรมดา โปรแกรมจะสามารถเข้าถึงหน่วยความจำ ตำแหน่งใดๆก็ได้ไม่ว่าจะได้ทำการขอไว้จากDosหรือไม่

แต่ในระบบวินโดว์ นั้นหน่วยความจำเป็นทรัพยากรปันส่วนจึงจำเป็นจะต้องใช้อย่างระมัดระวังไม่ให้กระทบต่อหน่วยความจำส่วนอื่นๆ ที่ไม่ได้ทำการจองเอาไว้ และแต่ละครั้งในการทำงานโปรแกรมก็จะจองหน่วยความจำเท่าที่ต้องการเคลื่อนย้ายหรือแม้กระทั่งจำกัดหน่วยความจำบางส่วน ซึ่งหมายถึงข้อมูลต่างๆจะอยู่ในตำแหน่งที่ไม่แน่นอน หากมีโปรแกรมหลายๆ โปรแกรมทำงานพร้อมๆกัน หน่วยความจำจะถูกย้ายที่บ่อยมาก

2.3.2 รูปแบบของโปรแกรมที่ทำงานแบบ ไมโครซอฟต์ วินโดว์

โปรแกรมที่จะทำงานบนระบบวินโดว์จะต้องมีลักษณะเฉพาะเพื่อให้สามารถใช้งานได้ในคุณสมบัติที่กล่าวมาแล้วข้างต้น จะประกอบไปด้วยส่วนต่างๆ พื้นฐานดังนี้

- ช่องหน้าต่าง (Windows)
- เมนู (Menu)
- กรอบข้อความ (Dialog Box)
- ส่วนจัดการตัวสาร (Massage Loop)

ช่องหน้าต่าง (Windows)

ช่องหน้าต่างเป็นอุปกรณ์แสดงข้อมูลพื้นฐานของโปรแกรม ประกอบด้วยส่วนต่างๆ คือ Title Bar ,Menu Bar,Scroll Bar, Borders และอื่นๆที่อยู่ภายในสี่เหลี่ยมบนจอภาพดังรูป 2.9 ถึงแม้ตัวโปรแกรมจะเป็นเจ้าของช่องหน้าต่างที่สร้างขึ้นมาก็ตาม แต่การทำงานเป็นการร่วมกันทำงานระหว่างโปรแกรมและระบบ วินโดว์ ระบบ วินโดว์ จะจัดการหน้าที่มาตรฐานต่างๆ ของช่องหน้าต่าง เช่น ขนาด, ตำแหน่งบนจอภาพ, การเลือกเมนู, ชื่อช่องหน้าต่าง

ระบบเมนู (Menu)

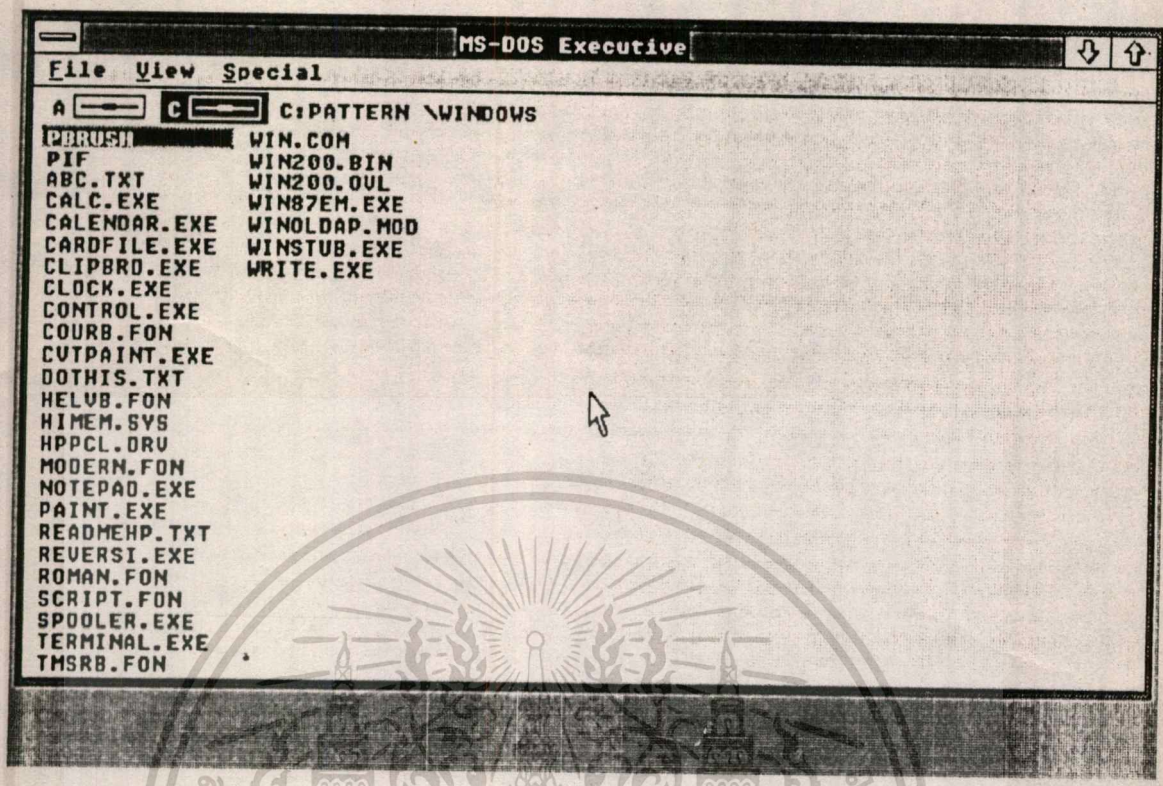
เมนูเป็นวิธีหลักที่โปรแกรมจะรับข้อมูลจากผู้ใช้ เมนูจะเป็นรายชื่อของคำสั่งที่ผู้ใช้สามารถเรียกดูและเลือกใช้ เพื่อโปรแกรมทำงานจะต้องส่งข้อมูลของเมนูเพื่อให้ระบบวินโดว์ สร้างเมนูให้ จากนั้นเพื่อผู้ใช้เลือกเมนู ก็จะมี ข้อความเข้า (Input Massage) เข้ามาทางระบบคิว

กรอบความ (Dialog Boxes)

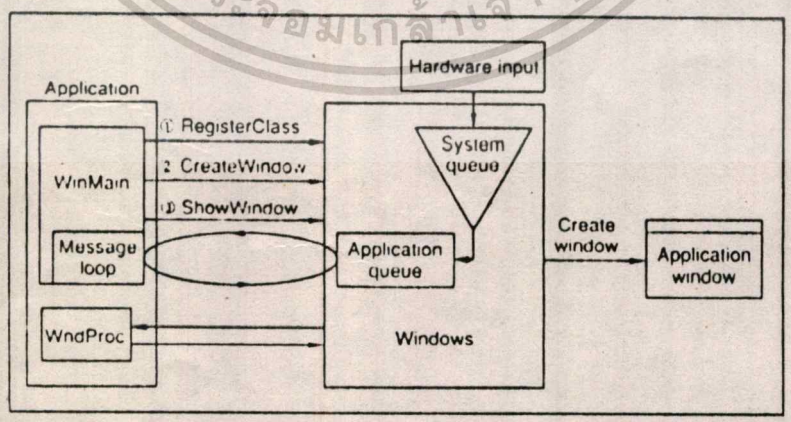
กรอบความ คือช่องหน้าต่างชั่วคราวซึ่งเปิดโอกาสให้ผู้ใช้ได้ทราบรายละเอียดของคำสั่งได้มากขึ้นภายในกรอบจะประกอบไปด้วยข้อความสั้นๆ และควบคุมง่ายๆ เพื่อให้ผู้ใช้เลือกทำหรือใส่ข้อความเพียงเล็กน้อยเท่านั้น

ส่วนจัดการตัวสาร (Message Loop)

เนื่องจาก โปรแกรมจะรับข้อมูลผ่านทางคิว ส่วนสำคัญของโปรแกรมก็จะเป็นส่วนจัดการข่าวสารซึ่งจะทำหน้าที่รับข้อมูลจากคิวแล้วแจกจ่ายไปให้ช่องหน้าต่างตามความเหมาะสม ดังรูป 2.10 ระบบ วินโดว์ จะรับข้อมูลจากอุปกรณ์ทุกๆชิ้น แล้วนำมาจัดสรรลงในคิวระบบแล้วจะทำการก๊อปปี้ข้อมูลนั้น ไปยังคิวของโปรแกรม จากนั้นจะแจกจ่ายไปยังช่องหน้าต่างตามความเหมาะสม โดยผ่านทางระบบ วินโดว์ จากนั้นโปรแกรมย่อยที่จะตอบสนองต่อข้อมูลนั้นก็จะทำงาน

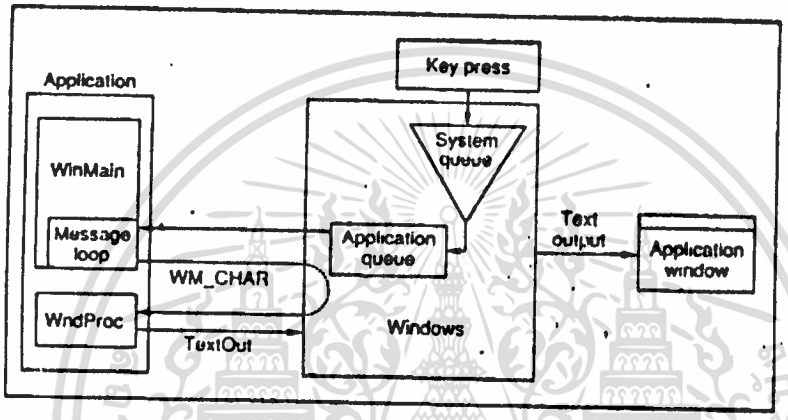


รูป 2.9 วินโดว์



รูป 2.10 บล็อกไดอะแกรมของวินโดว์

จากรูป 2.11 เป็นตัวอย่างการทำงานร่วมกันของโปรแกรมใช้งานกับระบบ วินโดว์ ในการรับข้อมูลตัวอักษร เริ่มต้นระบบจะรับข้อมูลจากแป้นพิมพ์เมื่อผู้ใช้ได้กดหรือปล่อยปุ่มบนแป้นพิมพ์ ระบบจะก๊อปปี้ข่าวสารจากตัวระบบไปยังคิวใช้งาน ส่วนจัดการข่าวสารจะรับข่าวสารจากคิวใช้งาน มาแปลงเป็น อักขระ ANSI(WM_CHAR) และส่งข่าวสาร WM_CHAR นี้ไปยัง ฟังก์ชั่นของวินโดว์ที่เหมาะสม



รูป 2.11 Loop การรับส่งสาร

จากนั้นฟังก์ชั่นของวินโดว์จะทำการเรียกระบบ วินโดว์ให้แสดงผลเป็นอักขระออกมาทางช่องหน้าต่าง

บทที่ 3
การออกแบบและการสร้าง

ในโครงงานนี้ได้แบ่งงานที่ต้องทำออกเป็น 2 ส่วนใหญ่ คือ งานด้านฮาร์ดแวร์เชื่อมต่อ และงานด้านซอฟต์แวร์ 4 โปรแกรม โดยจะขอกล่าวเป็นส่วนๆดังนี้ คือ

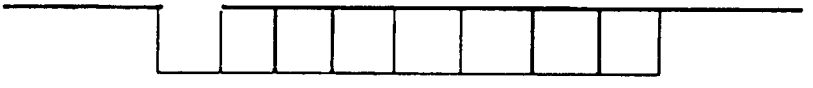
3.1 ส่วนการเชื่อมต่อ (Hardware Interface)

งานด้านฮาร์ดแวร์เป็นการพัฒนาส่วนเชื่อมต่อกับมิดิ (MIDI Interface) เพื่อให้เครื่องคอมพิวเตอร์สามารถสื่อสารกับอุปกรณ์มิดิได้ โดยมีขั้นตอนในการทำงาน คือ การเรียนรู้มาตรฐานของการสื่อสารแบบมิดิ การศึกษาวงจรการสื่อสารที่มีอยู่เดิม จากนั้นทำการออกแบบปรับปรุงแก้ไข เพื่อให้เข้ากับจุดประสงค์ของงานที่ต้องการ และสุดท้ายคือการสร้างลงบนการ์ดจริง พร้อมทั้งทดลองใช้งาน หัวข้อที่จะกล่าวต่อไปแบ่งออกเป็นหัวข้อย่อย ดังนี้ มาตรฐานการเชื่อมต่อแบบมิดิ โครงสร้างของวงจร การทำงานและการเริ่มต้นของ Z-80 DART การทำงานและการเริ่มต้นของ 8253

3.1.1 มาตรฐานการเชื่อมต่อแบบมิดิ

จากในบทที่แล้วได้กล่าวถึงหลักการในการเชื่อมต่ออุปกรณ์ที่ใช้มาตรฐานมิดิในการสื่อสารเข้าด้วยกัน ในส่วนนี้จึงจะขอกล่าวเพิ่มเติมในรายละเอียดของการสื่อสารแบบมิดิ

การส่งของมูลแบบมิดินั้นเป็นการส่งแบบอนุกรมอะซิงโครนัส (Serial Asynchronous) โดยมี บิทเริ่มต้น 1 บิท (1 Start bit), บิทสิ้นสุด 1 บิท (1 Stop bit) มีบิทข้อมูล 8 บิท (8 Data Bits) โดยที่ไม่มีบิทตรวจสอบความผิดพลาดใดๆ (Parity bit) สายที่ใช้สื่อสารเป็นสายแบบ 1 เส้นสัญญาณ 1 สายดิน เพราะฉะนั้นในสายแต่ละเส้นจึงเป็นการสื่อสารทางเดียว สิ่งที่สำคัญที่ต้องคำนึงถึงมากที่สุดก็คือ ความเร็วในการส่งจะต้องส่งด้วยความถี่ 31250 Hz และต้องมีความผิดพลาดได้ไม่เกิน 1 % ส่วนแจ็คที่ใช้จะต้องเป็นแจ็คดิน (Din) 5 ขา

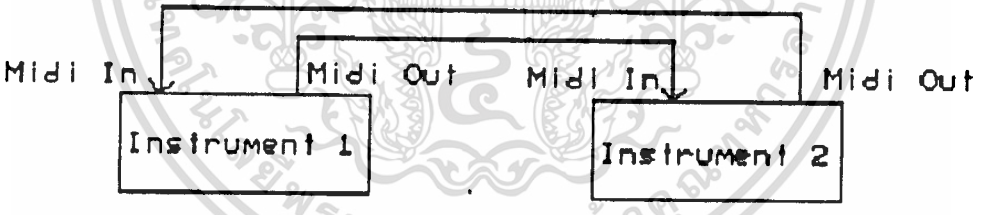


รูปที่ 3.1 ลักษณะสัญญาณมิดิ

เนื่องจากมาตรฐานในการส่งกำหนดให้การส่งเป็นแบบ Current Loop (หมายถึง การที่สัญญาณที่ได้รับจะเป็น 0 หรือเป็น 1 พิจารณาได้จากทิศทางของกระแสที่ไหลในสาย) โดยที่กำหนดให้กระแสที่ไหลในสายมีค่าไม่เกิน 5 มิลลิแอมป์ และสายเชื่อมต่อควรมีความยาวไม่เกิน 50 ฟุต

รูปที่ 3.2 ลักษณะการต่อสายสัญญาณกับแจ็คดิน

3.1.2 โครงสร้างวงจร

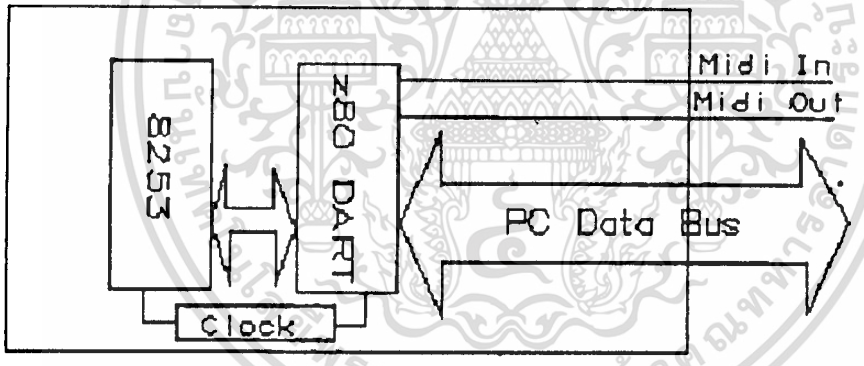


รูปที่ 3.3 ลักษณะการต่อเชื่อมของข่ายการสื่อสารแบบมิดิ

พิจารณาจากรูปการเชื่อมต่อ ส่วนหลักในเครือข่ายการสื่อสารที่ใช้ในโครงงานนี้ คือ การ์ดที่ทำหน้าที่ในการส่งสัญญาณมิดิ และเครื่องดนตรีที่รับสัญญาณมิดิ โดยพอร์ทที่รับสัญญาณมิดิเข้าไปของอุปกรณ์เรียกว่า **พอร์ทมิดิอิน** (MIDI in Port) และพอร์ทที่ทำหน้าที่ส่งสัญญาณมิดิออกมานั้นเรียกว่า **พอร์ทมิดิเอาท์** (MIDI out Port) ส่วนการต่อเครื่องดนตรีในเครือข่ายมากกว่า 1 ชิ้นจะต้องผ่านทาง **พอร์ทมิดิทรู** (MIDI thru Port) ซึ่งพอร์ทนี้จะส่งสัญญาณที่เข้ามาทางพอร์ทมิดิอินออกไปทางพอร์ทมิดิทรูอีกต่อหนึ่ง เนื่องจากข้อมูลมิดิสามารถกำหนดแชนแนลในการรับส่งได้ถึง 16 แชนแนลจึงสามารถใช้สายเส้นเดียวในการติดต่อกับเครื่องดนตรีหลายๆชิ้น (โดยที่เครื่องดนตรีแต่ละชิ้นจะถูกตั้งให้รับสัญญาณมิดิไม่ซ้ำแชนแนลกัน)

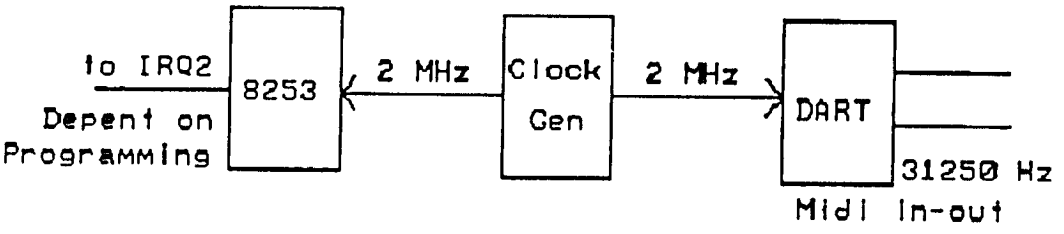
การ์ดมิดิที่สร้างขึ้นจะถือเป็นเสมือนเครื่องดนตรีชิ้นหนึ่ง คือ สามารถรับส่งสัญญาณมิดิผ่านทางพอร์ทมิดิอินและมิดิเอาท์ได้เช่นเดียวกัน แต่ในโครงงานนี้ความสามารถของพอร์ทมิดิอินของการ์ดไม่ได้ถูกนำมาใช้ ดังนั้นการ์ดจึงทำตัวเสมือนเป็นแหล่งกำเนิดสัญญาณมิดิเพียงอย่างเดียว การทำงานของการ์ดมิดิที่สร้างขึ้นจะมีส่วนที่สำคัญอยู่ 3 ส่วน (รูปที่ 3.4) คือ

ส่วนที่ทำการเปลี่ยนข้อมูลที่เป็นแบบขนานใน Data bus ของเครื่อง PC มาเป็นข้อมูลแบบอนุกรมให้ได้ความเร็วและลักษณะของข้อมูลตามมาตรฐานของมิดิ (และในทางกลับกันคือเปลี่ยนจากอนุกรมมาเป็นขนานด้วย) ซึ่งในโครงงานนี้ใช้ชิพของไซลอก (Zilog) เบอร์ Z-80 DART มาทำหน้าที่แปลงลักษณะของข้อมูล เนื่องจากเป็นชิพที่ออกแบบมาเพื่อการสื่อสารแบบ Asynchronous โดยตรง ขา RxDA เป็นขาที่รับข้อมูลจากสายสัญญาณมิดิ และขา TxDA ทำหน้าที่ตรงกันข้าม คือรับสัญญาณมิดิเข้ามา โดยผ่าน Opto coupler เพื่อป้องกันสัญญาณรบกวนและขา D0-D7 ก็ต่อโดยตรงกับ Data bus ของเครื่อง PC เพื่อทำการรับ ส่งข้อมูลกับเครื่องซึ่งชิพตัวนี้จะได้สัญญาณนาฬิกามาจากส่วนผลิตสัญญาณนาฬิกา 2 MHz โดยจะนำมาหาร 64 เพื่อให้ได้ความถี่ 31250 Hz ตามมาตรฐานมิดิ



รูปที่ 3.4 บล็อกไดอะแกรมของการ์ดมิดิ

ส่วนเคาน์เตอร์ (Counter) ในโครงงานนี้เลือกใช้เบอร์ 8253 เป็นส่วนที่ใช้ในการสร้างความเร็วช้า (Tempo) ทั้งหมดในโปรแกรมส่วนที่ควบคุมการส่งสัญญาณมิดิไปยังเครื่องดนตรี ซึ่งตัว 8253 นี้จะถูกป้อนด้วยสัญญาณนาฬิกาความถี่ 2 Mhz ให้กับเคาน์เตอร์ 0 เป็นตัวหารความถี่ในชั้นแรก จากนั้นสัญญาณนาฬิกาที่ออกมาจากเคาน์เตอร์ 0 ก็จะถูกส่งไปให้เคาน์เตอร์ 1 ทำการหารความถี่ในชั้นที่ 2 และสัญญาณนาฬิกาที่ออกมาจากเคาน์เตอร์ 1 ก็จะถูกผ่านชุดเลือกทางเดินอินเตอร์รัพท์ แล้วจึงส่งไปให้ขา IRQ2 ของสล็อตของเครื่องพีซี เพื่อใช้เป็นสัญญาณในการสร้าง ความเร็วช้า (Tempo) ต่างๆกันไปในแต่ละเพลง ซึ่งค่าของเคาน์เตอร์ 0 และเคาน์เตอร์ 1 สามารถโปรแกรมได้ด้วยซอฟต์แวร์



รูปที่ 3.5 สัญญาณนาฬิกาที่จุดต่างๆบนการ์ด

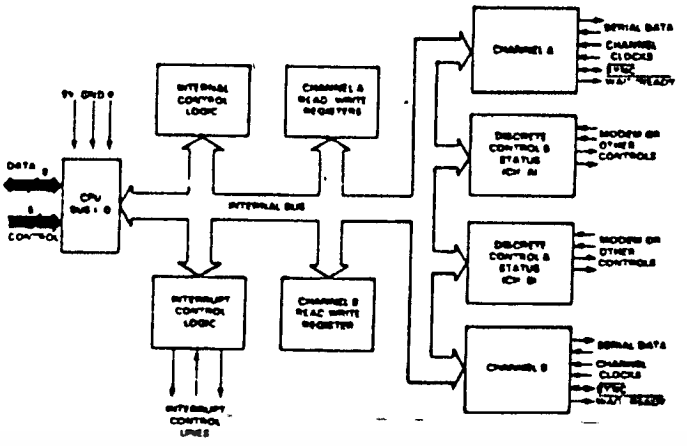
และส่วนสุดท้ายเป็นส่วนที่ใช้ในการเลือกพอร์ท (Decode Port) ที่ใช้ในการติดต่อของเครื่องกับชิพต่างๆบนการ์ด โดยใช้ 74138 มาเป็นตัวเลือก ซึ่งในโครงงานนี้จะทำการเลือกพอร์ทต่างๆดังนี้

พอร์ท FFA0 เป็นพอร์ทข้อมูล (ทั้งอินและเอาท์) ของ Z-80 DART
พอร์ท FFA2 เป็นพอร์ทควบคุมของแชนแนล A ของ Z-80 DART
พอร์ท FFA4 เป็นพอร์ทข้อมูลของเคาน์เตอร์ 0 ของ 8253
พอร์ท FFA5 เป็นพอร์ทข้อมูลของเคาน์เตอร์ 1 ของ 8253
พอร์ท FFA7 เป็นพอร์ทควบคุมของ 8253
พอร์ท FFA8 เป็นพอร์ทสลับ (Toggle) การเลือกทางเดินของสัญญาณอินเตอร์รัพท์ระหว่าง 8253 กับ Z-80 DART
พอร์ท FFAC เป็นพอร์ทเลือกทางเดินของสัญญาณอินเตอร์รัพท์จาก 8253

ในส่วนของ 2 พอร์ทหลังเป็นส่วนที่ช่วยในการเลือกทางเดินของสัญญาณอินเตอร์รัพท์ระหว่าง 8253 กับ Z-80 DART (พิจารณาจากรูป 3.6 ประกอบ) โดยจะใช้ฟลิปฟลอปเบอร์ 7473 เป็นตัวเลือกเพื่อให้ชิพทั้งสองตัวสามารถที่จะส่งสัญญาณมาอินเตอร์รัพท์ซีพียูได้ เมื่อทำการอ้างถึงพอร์ท FFAC จะเป็นการกระตุ้นขาเคลียร์ (Clear) ของ 7473 ให้ขา Q เป็น 1 ทำให้สัญญาณจากขา Out1 ของ 8253 สามารถผ่านแอนด์เกทไปยังขา IRQ2 ของสล็อตบนเครื่องพีซีได้ จากนั้นเมื่อทำการอ้างพอร์ท FFA8 จะเป็นการสลับ (Toggle) ฟลิปฟลอบให้ขา Q เป็น 1 สัญญาณที่มาจาก Z-80 DART ก็จะสามารถผ่านแอนด์เกทเข้าไปยังของ IRQ2 ได้ เมื่อทำการอ้างถึงอีกครั้งก็จะสลับกลับไปอีก ทำให้เลือกสัญญาณอินเตอร์รัพท์จากชิพตัวใดก็ได้ (สัญญาณอินเตอร์รัพท์จาก Z-80 DART จะมีประโยชน์ตอนรับข้อมูลจากเครื่องดนตรีต่างๆเข้ามา)

3.1.3 การทำงานและการเริ่มต้น (Initialize) ของ Z-80 DART

Z-80 DART เป็นชิพที่ออกแบบมาเพื่อการสื่อสารแบบอนุกรมอะซิงโครนัสโดยเฉพาะ โดยมีบล็อกไดอะแกรมดังนี้



รูปที่ 3.7 บล็อกไดอะแกรมของ Z-80 DART

จากรูปจะเห็นว่า Z-80 DART มีแชนแนลในการรับส่งถึง 2 แชนแนล คือ แชนแนล A และแชนแนล B แต่ในโครงงานนี้จะใช้แชนแนล A เพียงแชนแนลเดียว สามารถควบคุมการทำงานและอ่านค่าสถานะด้วยรีจิสเตอร์ (Register) ต่างๆ จึงทำให้สามารถควบคุมการทำงานทั้งหมดได้ด้วยซอฟต์แวร์ ซึ่งรีจิสเตอร์ต่างๆเหล่านี้ประกอบด้วยรีจิสเตอร์ในการเขียน (internal write register) ของแชนแนล A 7 ตัว และของแชนแนล B 8 ตัว โดยที่แชนแนล B มีรีจิสเตอร์ที่ใช้เก็บอินเตอร์รัพท์เวคเตอร์อีก 1 ตัว นอกจากนั้นยังมีรีจิสเตอร์ที่ใช้ในการอ่านค่าสถานะต่างๆอีก 3 ตัว(ข้อมูลของรีจิสเตอร์สามารถอ่านเพิ่มเติมได้จากภาคผนวก) การทำงานของ Z-80 DART สามารถทำได้ทั้ง แบบ อินเตอร์รัพท์ (Interrupt) , โพลลิ่ง (Polling) และ DMA

การตั้งค่าเริ่มต้นของ Z-80 DART

เริ่มต้นจะต้องทำการตั้งค่าให้ Z-80 DART มีการรับส่งข้อมูลที่เป็นมาตรฐานของมิดี คือมีข้อมูล 8 บิท,มีบิทเริ่มต้นเป็น 0 มีบิทสิ้นสุดเป็น 1 และมีการหารความถี่ที่เข้ามาด้วย 64 เพื่อนำไปใช้ในการรับส่งข้อมูล (2 MHz / 64 = 31250 Hz) ซึ่งสัญญาณนี้จะมีผลต่อขา RxA,RxB,TxA,TxB โดยมีขั้นตอนในการตั้งค่าดังนี้

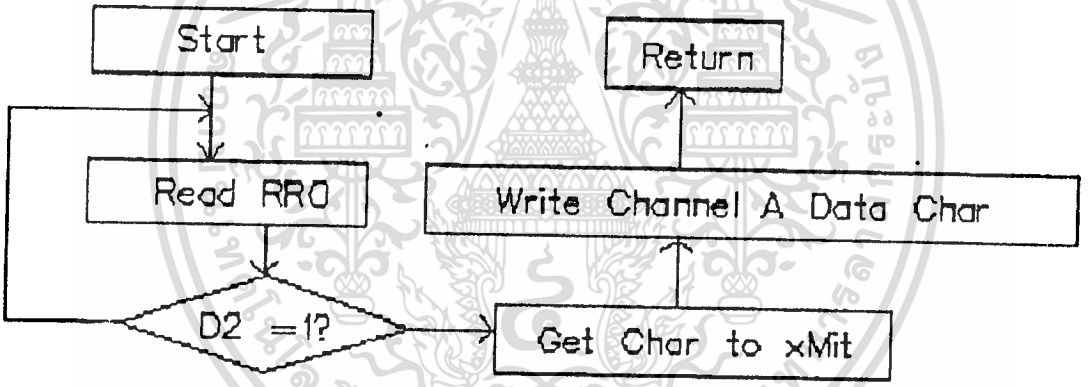
1. เขียนข้อมูลให้ รีจิสเตอร์เขียน 0 (WR0) เพื่อเริ่มการทำงาน
2. เขียนข้อมูลให้ รีจิสเตอร์เขียน 1 (WR1) เพื่อเป็นการกำหนดค่าที่จะนำมาใช้หารความถี่ของสัญญาณนาฬิกาที่เข้ามา โดยในโครงงานนี้ต้องกำหนดให้เป็นค่า 64 และกำหนดขนาดของบิทสิ้นสุดเท่ากับ 1
3. เขียนข้อมูลให้ รีจิสเตอร์เขียน 3 (WR3) เพื่อเป็นการกำหนดจำนวนบิทของข้อมูลที่ทำการรับให้เป็น 8 บิท และทำการอีนาเบิ้ลตัวรับ (Receiver Enable) กับ เซ็ทให้เป็นอีนาเบิ้ลแบบอัตโนมัติ (Auto Enable)
4. เขียนข้อมูลให้ รีจิสเตอร์เขียน 5 (WR5) เพื่อเป็นการกำหนดจำนวนบิทของข้อมูลที่ทำ

การส่งให้เป็น 8 บิทและทำการอีนาเบิ้ลตัวส่ง (Transmitter Enable)

5. เขียนข้อมูลให้ รีจิสเตอร์เขียน 1 (WR1) เพื่อทำการอีนาเบิ้ลหรือดิสเอเบิ้ล การขอ อินเตอร์รัพต์จาก Z-80 DART

ขั้นตอนในการส่งข้อมูลอนุกรม

ในขั้นตอนแรกสุดจะทำการตรวจสอบสถานะของบัฟเฟอร์ตัวส่งว่าว่างที่จะรับข้อมูลหรือไม่ โดยอ่านจากรีจิสเตอร์สถานะตัวที่ 0 (RR0) และตรวจสอบบิทที่ 3 ถ้ามีค่าเป็น 1 แสดงว่าบัฟเฟอร์ว่างสามารถที่จะส่งข้อมูลได้ แต่ถ้ามีค่าเป็น 0 จะต้องทำการรอแล้วตรวจสอบใหม่จนมีค่าเป็น 1 จึงจะส่งข้อมูลออกไปได้

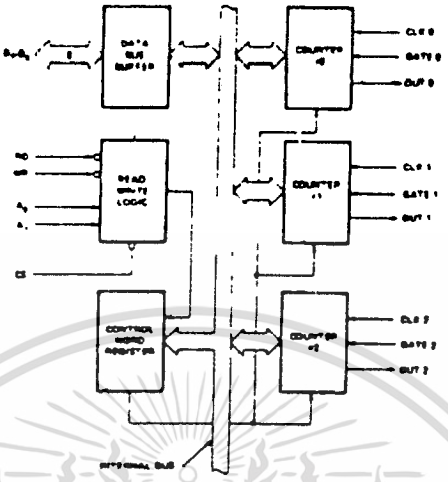


รูปที่ 3.8 ขั้นตอนในการส่งข้อมูล

ส่วนขั้นตอนในการรับข้อมูลนั้นในที่นี้จะไม่ขอกล่าวถึง เนื่องจากยังไม่ได้ทำในโครงงานนี้

3.1.4 การทำงานและการเริ่มต้นของ 8253

8253 เป็นไอซีที่สามารถทำหน้าที่เป็น ไทมเมอร์ (Timer) หรือ เคาเตอร์ (Counter) ก็ได้ โดยจะมีตัวนับเวลาอยู่ 3 ชุด (ดังรูป 3.9) ซึ่งเป็นรีจิสเตอร์ขนาด 16 บิท โดยที่แต่ละตัวสามารถทำงานได้เป็นอิสระต่อกัน มีขา Clk เป็นขาที่รับสัญญาณเข้ามา ขา Out เป็นขาที่ส่งสัญญาณเอาท์พุทออกไป การทำงานของตัวนับแต่ละตัวสามารถทำงานได้หลายโหมด โดยสามารถควบคุมได้โดยซอฟต์แวร์ผ่านทางรีจิสเตอร์ควบคุมของ 8253



รูปที่ 3.9 บล็อกไดอะแกรมของ 8253

โหมดต่างๆของ 8253

1. โหมด 0 Event Counter
2. โหมด 1 Programmable one-shot
3. โหมด 2 Rate Generator
4. โหมด 3 Square Wave Generator
5. โหมด 4 Software Triggered Strobe
6. โหมด 5 Hardware Triggered Strobe

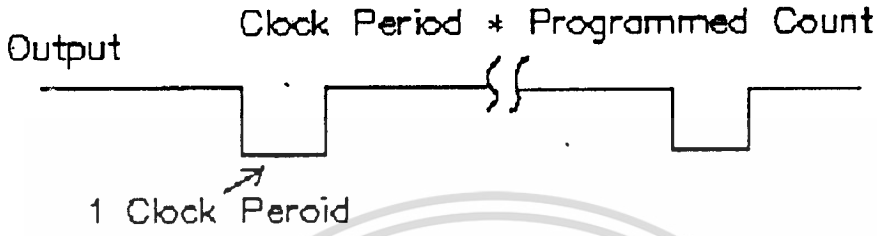
โหมดที่เลือกใช้ในการโครงงานนี้ คือ ให้เคาเตอร์ทั้ง 2 ตัวทำงานที่โหมด 2 ซึ่งมีลักษณะเป็นการหารความถี่ที่เข้ามาทางขา Clk ด้วยค่าที่โปรแกรมไว้ นั่นคือเมื่อมีสัญญาณนาฬิกาเข้ามา 1 ลูกก็จะลดค่าที่ตั้งไว้ในรีจิสเตอร์ เมื่อนับลงถึง 0 ก็จะให้เอาท์พุทออกมาหนึ่งลูก จากนั้นเริ่มนับถอยหลังลงมาใหม่เมื่อถึง 0 ก็จะให้เอาท์พุทอีกหนึ่งลูกเป็นอย่างนี้เรื่อยไปสามารถเขียนเป็นขั้นตอนการตั้งค่าได้ดังนี้

1. ตั้งค่ารีจิสเตอร์ควบคุมตั้งเคาเตอร์ 0 ให้ทำงานในโหมด 2 นับแบบฐาน 2 และให้ใส่ไบท์ต่ำก่อนแล้วตามด้วยไบท์สูง
2. ใส่ค่าไบท์ต่ำในเคาเตอร์ 0 ตามด้วยค่าไบท์สูง
3. ตั้งค่ารีจิสเตอร์ควบคุมตั้งเคาเตอร์ 1 ให้ทำงานในโหมด 2 นับแบบฐาน 2 และให้

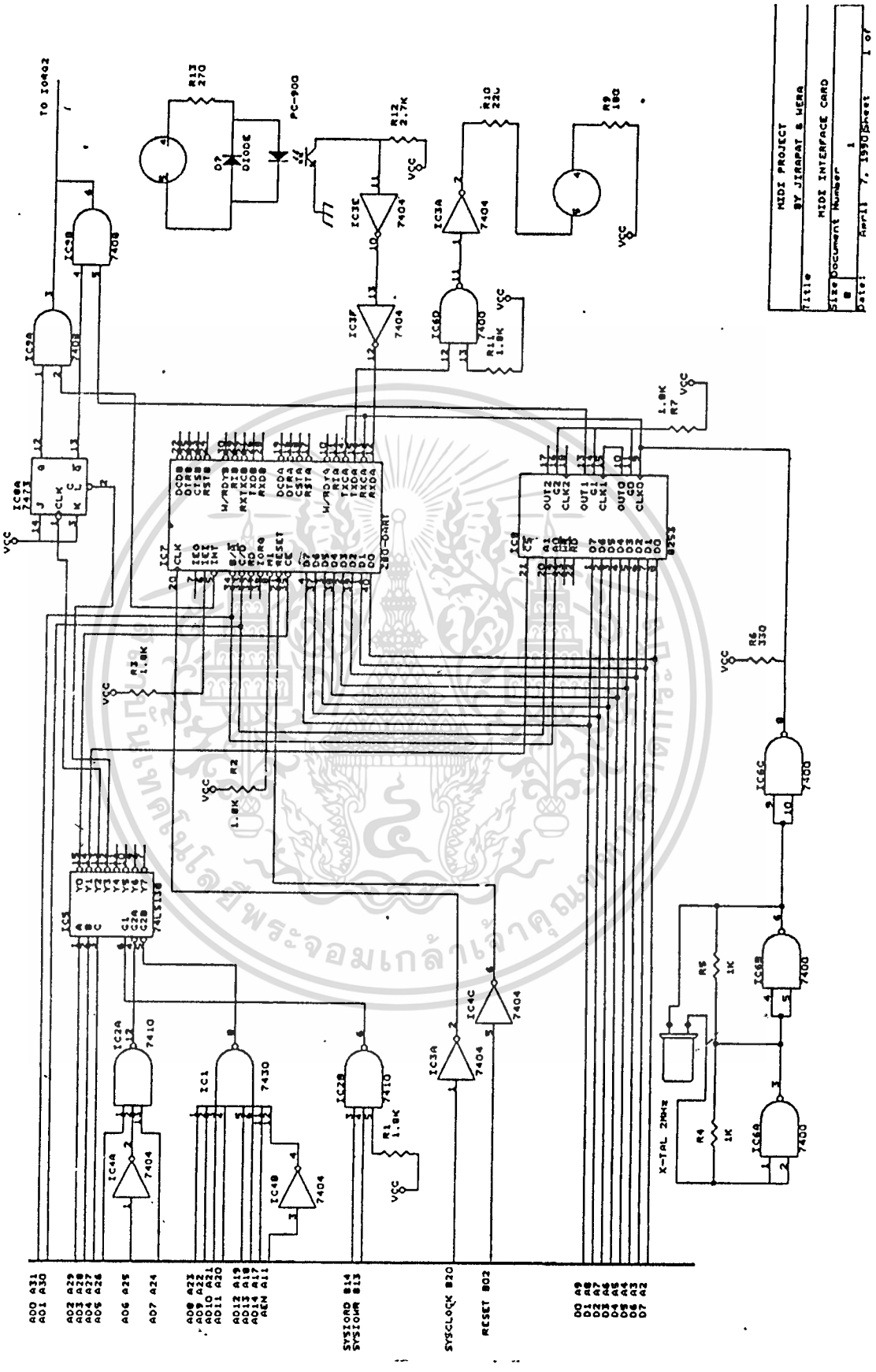
ใส่ไบท์ต่ำก่อนแล้วตามด้วยไบท์สูง

4. ใส่ค่าไบท์ต่ำในเคาเตอร์ 1 ตามด้วยค่าไบท์สูง

ส่วนการทำงานในโหมดอื่นนั้นสามารถอ่านเพิ่มเติมได้จากภาคผนวก ข.



รูป 3.10 ลักษณะการหารความถี่ของ 8253 โหมด 2



รูป 3.11 วงจรมิดิอินเตอร์เฟส

3.2 ส่วนโปรแกรม

3.2.1 แนวความคิดและหลักการออกแบบ

ความต้องการ จุดประสงค์หลักที่เป็นแนวคิดในการออกแบบซอฟต์แวร์ในโครงงานนี้ ก็คือ ต้องการให้นักแต่งเพลง (Composer) สามารถสร้างงานดนตรีที่มีประสิทธิภาพด้วยเครื่องคอมพิวเตอร์ส่วนตัวที่มีไม่ต้องมีการออกแบบมาเพื่องานด้านนี้เป็นพิเศษ เมื่อนำเครื่องดนตรีดิจิตอลที่ตนถนัดมาต่อเข้ากับส่วนเชื่อมต่อ (MIDI Interface) ก็สามารถใช้งานโปรแกรมนี้ได้อย่างสะดวก ทำให้พัฒนาการทางด้านดนตรีก้าวหน้าไปเร็วยิ่งขึ้น

ปัญหาของเครื่องพีซี เนื่องจากเป็นเครื่องได้รับความนิยมมากที่สุดในเครื่องระดับส่วนตัว การที่พัฒนาโปรแกรมจึงพัฒนาบนเครื่องนี้ (ไอบีเอ็ม พีซี/เอ็กซ์ที เอที) แต่เนื่องจากตัวเครื่องเองมีข้อจำกัดอยู่บ้าง การพัฒนาโปรแกรมจึงต้องพยายามหลบเลี่ยงข้อเสียเหล่านี้ เช่น การแสดงตัวโน๊ตจำเป็นอย่างยิ่งที่ต้องแสดงในกราฟฟิกโหมด เพราะความละเอียดของเท็กซ์โหมดมีไม่พอ นอกจากนั้นจอภาพของเครื่องพีซีที่มีขายอยู่ในท้องตลาดก็มีมากมายหลายมาตรฐาน การที่จะเขียนโปรแกรมให้สามารถใช้ได้กับจอทุกจอก็ดูจะเป็นเรื่องยาก จึงเป็นเหตุผลหนึ่งที่เลือกการเขียนโปรแกรมภายใต้ไมโครซอฟต์วินโดว์ ที่สามารถทำงานได้กับจอแทบทุกมาตรฐาน ทำให้ประหยัดเวลา นำเวลาไปพัฒนางานด้านอื่นได้อีก

หลักการออกแบบงานด้านซอฟต์แวร์ จากการที่ได้ทำการศึกษาโปรแกรมที่เกี่ยวข้องกับทางด้านดนตรีที่มีขายอยู่ในท้องตลาด ทำให้สามารถแบ่งชนิดของโปรแกรมประเภทนี้ออกได้เป็น โปรแกรมที่ทำงานแบบทีละขั้นตอน (Step Time) หมายถึงเป็นโปรแกรมที่มีช่วงเวลาระหว่างโน๊ตต่างๆแน่นอน ซึ่งส่วนมากจะทำงานกับโน๊ตอิดิเตอร์ โดยใช้ตัวโน๊ตอิดิเตอร์เป็นตัวรับข้อมูลแบบสเต็ปไทม์เข้ามาจากนั้นจึงค่อยส่งโน๊ตที่ได้รับมาไปให้เครื่องดนตรีต่างๆในลักษณะที่มีช่วงเวลาที่แน่นอน ส่วนอีกประเภทหนึ่งคือ โปรแกรมที่ทำงานเป็นแบบรีลไทม์ (Real Time) โดยโปรแกรมจะทำการบันทึกสัญญาณที่มาจากเครื่องดนตรีดิจิตอล จากนั้นจึงค่อยเล่นกลับ (Play back) ความห่างของแต่ละโน๊ตก็จะไม่คงที่ขึ้นกับการเล่นตอนเริ่มบันทึก โปรแกรมที่ได้รับการออกแบบในเทอมนี้จะเป็นการทำงานเป็นแบบสเตปไทม์ โดยมีโปรแกรมโน๊ตอิดิเตอร์เป็นส่วนที่รับโน๊ตแบบสเตปไทม์มาจากผู้ใช้ แล้วนำมาเล่นกลับโดยโปรแกรมแคสเซ็ทเพลเยอร์ (Cassette Player) ในลักษณะของรีลไทม์ เพื่อการพัฒนาการรับข้อมูลแบบรีลไทม์ในอนาคตด้วย

นอกจากโปรแกรมทั้งสองตัวที่ได้เกริ่นมาแล้วนั้น ระบบซอฟต์แวร์ในโครงงานนี้ยังประกอบด้วยโปรแกรมอีกส่วนมีชื่อว่า ซองแมเนเจอร์ (Song Manager) ซึ่งคอยทำหน้าที่เป็นตัวแปลงระหว่างข้อมูลเสตปไทม์และรีลไทม์จากโปรแกรมหลักทั้งสอง นอกจากนั้นยังมีหน้าที่ในการติดตั้ง (Installation) เครื่องดนตรีและอุปกรณ์มิดิต่างๆเข้ากับโปรแกรม

ประโยชน์อีกประการของไมโครซอฟต์วินโดว์ที่ปัจจัยในการออกแบบระบบซอฟต์แวร์ของ

โครงงานนี้ คือ การทำงานแบบมัลติทาสกิ้ง (Multitasking) จึงสามารถแยกส่วนต่างๆออกไปเป็นโปรแกรมย่อยๆได้อย่างเป็นอิสระไม่ขึ้นต่อกัน เป็นประโยชน์ในการเขียนและออกแบบโปรแกรม ซึ่งสามารถนำโปรแกรมเหล่านั้นมาทำงานพร้อมกัน หรือเรียกโปรแกรมเดิมหลายครั้ง เช่น โปรแกรมโน๊ตอิดิเตอร์ก็จะสามารถทำงาน (เปียนโน๊ต) ได้ทีละหลายๆเครื่องดนตรีพร้อมกัน ซึ่งตัวโปรแกรมนั้นเขียนขึ้นมาเพื่อทำงานทีละเครื่องดนตรี หรือเมื่อเรียกโปรแกรมโน๊ตอิดิเตอร์ออกมาพร้อมกับแคสเซทเพลเยอร์ผู้ใช้ก็จะสามารถเขียนโน๊ตไปพร้อมกับเล่นออกทางเครื่องดนตรีได้ โดยการถ่ายเทข้อมูลระหว่างโปรแกรมจะใช้ระบบดีดีอี (DDE-Dynamic Data Exchange) แต่จากการที่เลือกใช้ไมโครซอฟต์วินโดว์มาเป็นส่วนติดต่อกับผู้ใช้ทำให้เกิดข้อเสียบางประการ นั่นคือ ตัวไมโครซอฟต์วินโดว์เองจะเป็นตัวที่คอยควบคุมการใช้ทรัพยากรทั้งหมดของเครื่อง ทำให้การติดต่อโดยตรงกับส่วนเชื่อมต่อมิดิทำไม่ได้ จึงจำเป็นที่ต้องเขียนโปรแกรมในส่วนที่ 4 ขึ้นมา นั่นคือ โปรแกรมมิดิไดรฟเวอร์ (MIDI Driver) และเนื่องจากวินโดว์ก็คลุมอยู่บนเอ็มเอสดอส ไดรเวอร์ที่เขียนบนเอ็มเอสดอสก็สามารถใช้บนวินโดว์ได้ด้วย จึงได้สร้างโปรแกรม MIDIDRV.COM เพื่อเป็นไดรเวอร์ และ โปรแกรม MIDICONC.LIC และ MIDICONP.LIB เป็นไลบรารี่ที่ช่วยในการติดต่อกับภาษาสูง (ขึ้นละปาสคาลตามลำดับ) โดยมีโปรโตไทป์ชื่อ MIDICON.H

ส่วนแนวความคิดทางด้านโน๊ตจะใช้หลักการของ อีเวนต์ (Event) นั่นหมายถึง การแบ่งห้องเพลงออกเป็นอีเวนต์ ซึ่งจะเป็นตัวแสดงความละเอียดของโปรแกรม โดยใน 1 อีเวนต์ อาจจะมีโน๊ตเกิดขึ้น หรือเป็นอีเวนต์ว่างๆก็ได้

รูปที่ 3.12 ลักษณะการพิจารณาช่วงเวลาเป็นอีเวนต์

ต่อจากนี้จะขอกล่าวถึงรายละเอียดและแนวความคิดของโปรแกรมในแต่ละส่วน

3.2.2 โปรแกรมโน๊ตอิดิเตอร์ (Note Editor)

จุดประสงค์

เป็นโปรแกรมที่เขียนขึ้นมาเพื่อให้ผู้ใช้สามารถใส่ฝึกฝนการเขียนโน๊ตสากล ทั้งผู้ที่ไม่มีความรู้มากนัก และที่เป็นเป็นนักดนตรีที่มีประสบการณ์แล้ว สามารถใช้ในการช่วยแต่งเพลงได้ เนื่องจากเมื่อเขียนโน๊ตลงไปบนจอแล้วสามารถที่จะฟังเสียงโน๊ตเหล่านั้นได้ทางเครื่องดนตรีที่มีการสื่อสารแบบมิดิได้ทันที และเพื่อการทำงานที่รวดเร็ว การใช้โปรแกรมนี้จึงต้องใช้เมาส์ประกอบด้วยเสมอ

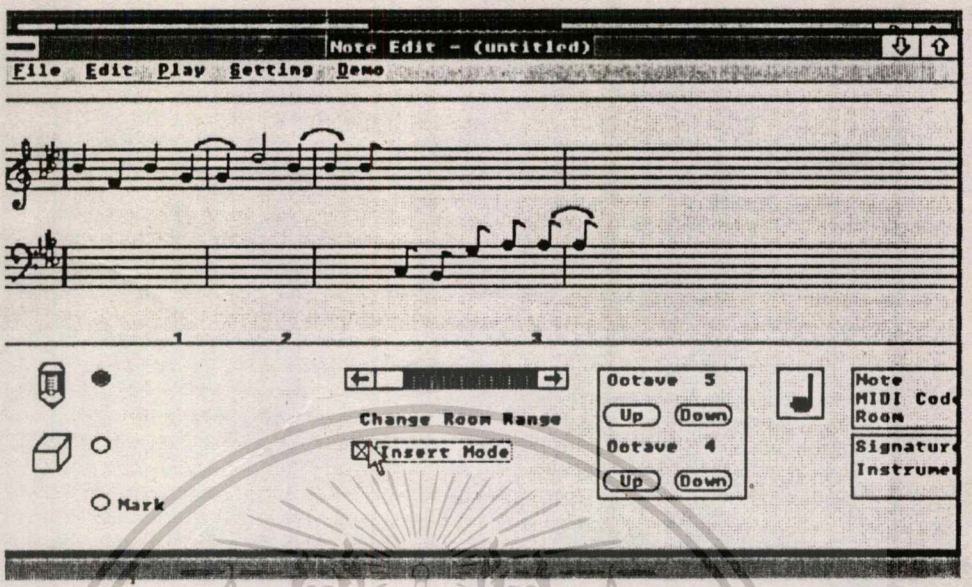
ความสามารถ

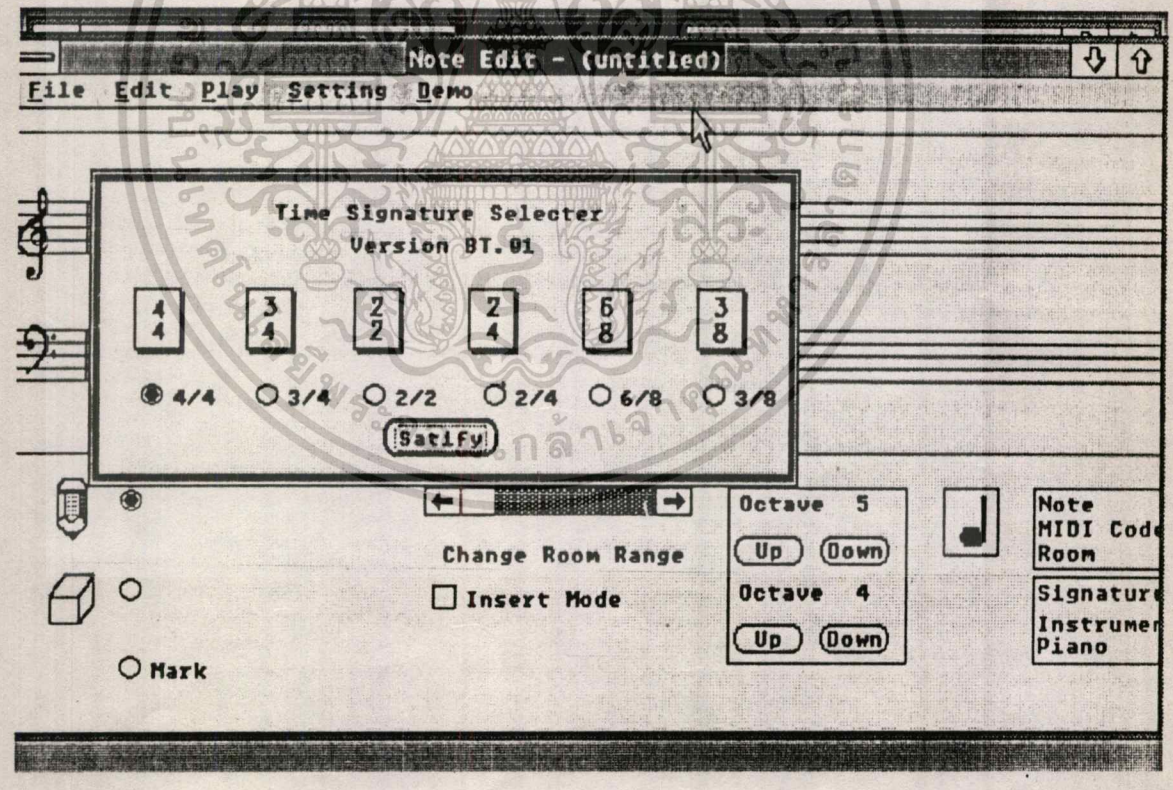
ความสามารถในด้านดนตรี : มีบรรทัด 10 เส้น (5 เส้นบนเป็นกุญแจซอล 5 เส้นล่างเป็นกุญแจฟา) ให้ สามารถให้ผู้ใช้เลือกโน๊ตสากลลงไปวางบนบรรทัด 10 เส้นนี้ได้ทุกตัว พร้อมทั้งตัวหยุดโดยสามารถใส่ให้โดยอัตโนมัติหรือผู้ใช้ใส่เอง มีความกว้างของช่วงโน๊ตถึง 11 ออกเทฟ (สูงสุดถึง 128 โน๊ต ครบตามมาตรฐานของมิดิ) เส้นกั้นห้องก็สามารถใส่ให้โดยอัตโนมัติมีเครื่องหมายกำหนดจังหวะให้ผู้ใช้เลือก 6 แบบ และสามารถเลือกเครื่องดนตรีที่ต้องการได้โดยที่เครื่องดนตรีที่มีอยู่ในระบบจะถูกรายงานมาโดยส่วน ชอง แมเนเจอร์ สำหรับการเปลี่ยนเสียงก็มีตัวเปลี่ยนเสียงให้ผู้ใช้เลือกได้ครบทั้ง 3 ตัว คือ ชาป (Sharp) แฟลท (Flat) และเนเจอรัล (Natural) โดยที่เมื่อโน๊ตในเสียงนั้นถูกเปลี่ยนเสียงด้วยเครื่องหมายเหล่านี้แล้วโน๊ตในเสียงนั้นที่ตามมาทั้งหมดก็จะถูกเปลี่ยนตามไปในลักษณะเดียวกันด้วย นอกจากนั้นยังสามารถเลือกคีย์มาตรฐานได้ทั้งหมด ในการเลือกคีย์นี้ ถ้าต้องการการใส่เครื่องหมาย ชาปหรือแฟลทที่บรรทัดใดโปรแกรมก็จะแสดงเครื่องหมายมาให้เห็นโดยอัตโนมัติ

ความสามารถในทางโปรแกรม : ตัวโปรแกรมถูกออกแบบให้มีความฉลาดในด้านดนตรีด้วยทฤษฎีดนตรีพื้นฐานที่ใส่ให้ เช่น การตัดโน๊ตที่เกินห้องออกแล้วแทนที่ด้วยโน๊ต 2 ตัวที่มีความยาวเสียงเท่ากันแล้วใช้เครื่องหมายโยงเสียงให้รู้ ดังรูป 3.12 หรือการประจุดให้กับตัวโน๊ตที่ไม่ได้มีความยาวตามโน๊ตปกติ เช่น 3 จังหวะ ก็จะสามารถแทนโน๊ต ด้วย แทน ความสามารถในการตัดโน๊ตเพื่อเก็บลงบัฟเฟอร์ชั่วคราวเพื่อนำไปแปะที่อื่นๆได้ สามารถเก็บ(Save)เพลงหรือ เรียกเพลงขึ้นมาใหม่ (Load) ได้ สามารถแก้ไขเพลงที่มีตัวโน๊ตได้ถึง 2500 ตัว โดยไม่จำกัดจำนวนห้อง ทั้งยังสามารถที่จะแก้ไขได้ทีละหลายเพลงพร้อมกัน โดยใช้ความสามารถที่จะเรียกโปรแกรมเดียวกันซ้ำๆออกมาได้ของไมโครซอฟตวินโดว์

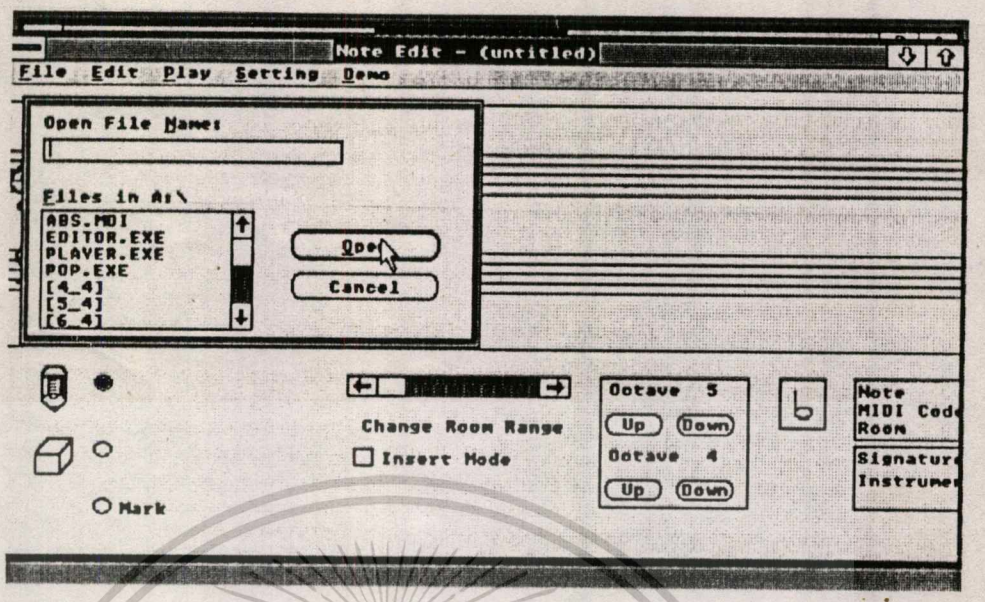
รูปที่ 3.13 การเปลี่ยนโน๊ตที่เกินห้อง

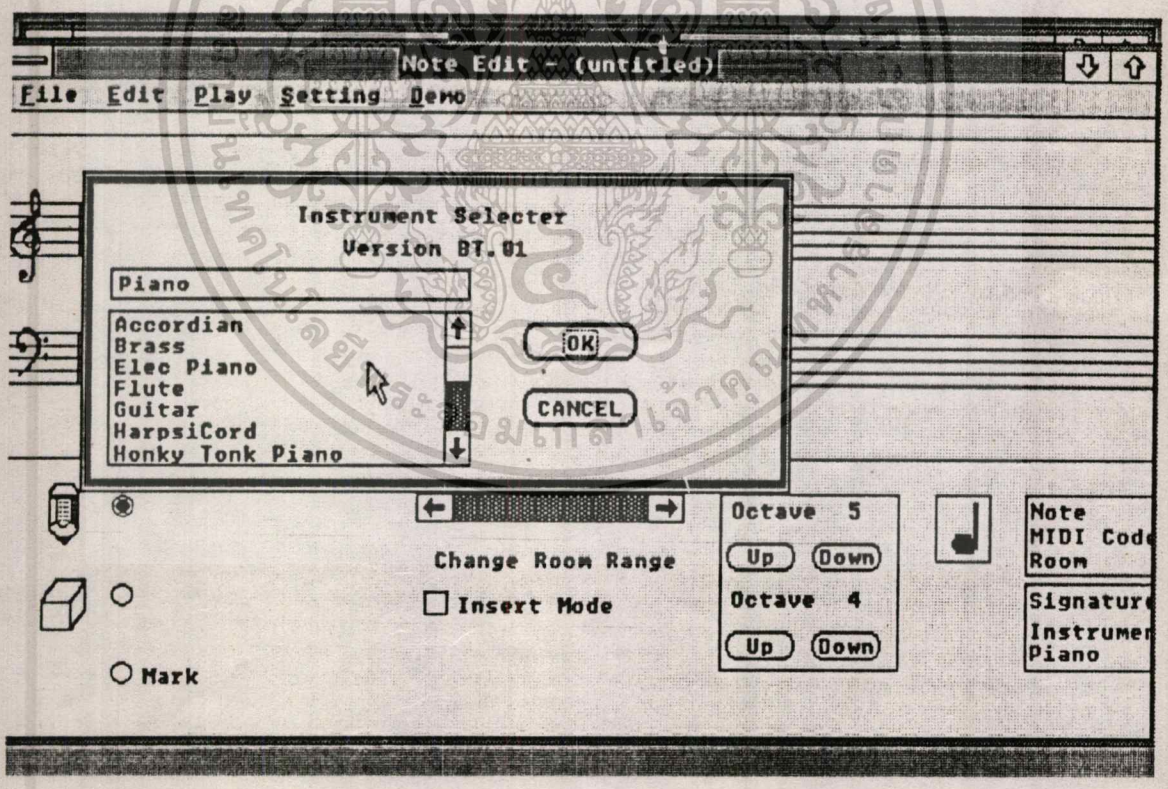
ข้อจำกัด ความสามารถของตัวอิดิเตอร์ยังมีได้ไม่เต็มที่เพราะทฤษฎีดนตรีที่เป็นส่วนที่สำคัญในการเพิ่มความฉลาดให้โปรแกรมยังต้องมีการศึกษาอีกมาก





รูป 3.14 โน๊ตอิดิเตอร์กับความสามารถต่างๆ





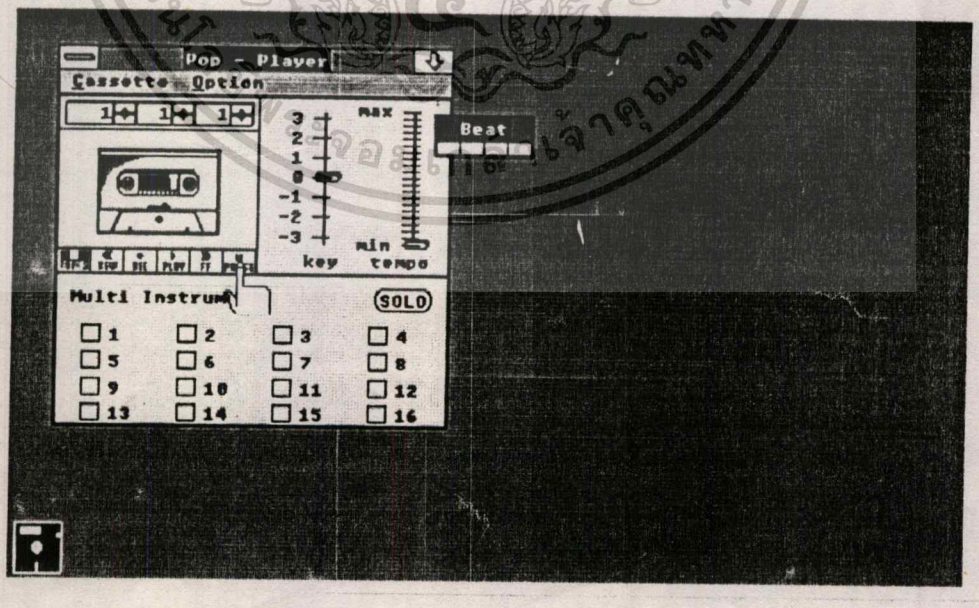
รูป 3.14 โน๊ตอิดิเตอร์กับความสามารถต่าง(ต่อ)

3.2.3 โปรแกรมแคสเซทเพลย์เยอร์ (Cassette Player)

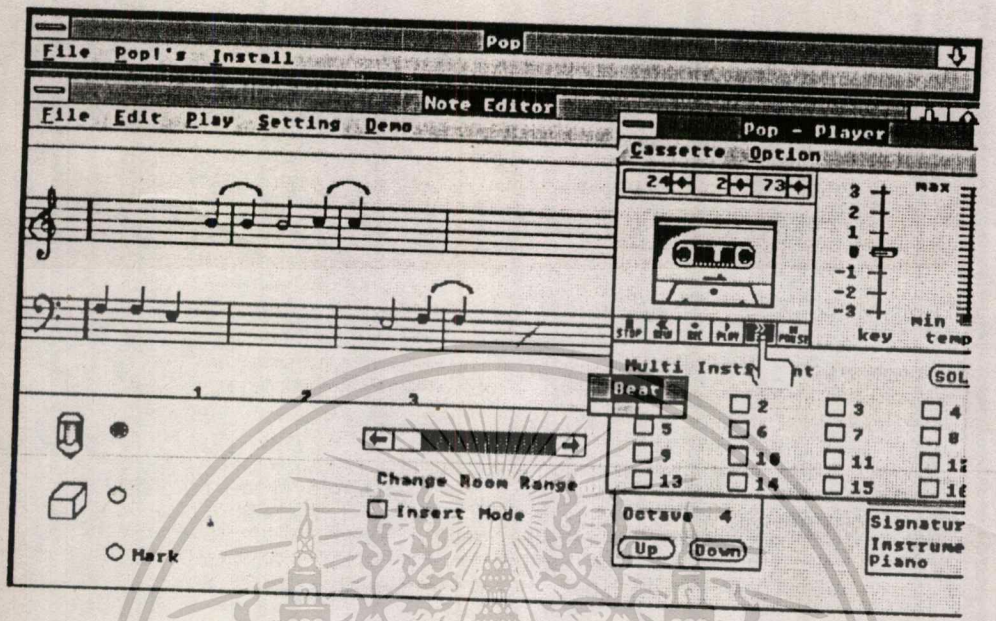
จุดประสงค์ เพื่อใช้เป็นส่วนควบคุมการเล่นเพลงของโปรแกรมทั้งหมดกับส่วนเชื่อมต่อกับมิดิ และให้ผู้ใช้สามารถเขียนโน๊ตบนโน๊ตอิดิเตอร์ได้หลายแบบเพื่อที่จะนำมาเล่นในระบบรีลไทม์ โดยได้ออกแบบมีการแปลงข้อมูลจากส่วนที่เป็นสเตปไทม์มาเป็นรีลไทม์ หรือสามารถรับอินพุทที่เป็นรีลไทม์ได้โดยตรง

ความสามารถ ลักษณะที่ออกแบบมีจุดประสงค์เพื่อให้ผู้ใช้ใช้ได้ง่ายที่สุด โดยมีการออกแบบช่องหน้าต่างให้เป็นรูปเทปแคสเซท มีปุ่มสำหรับเล่น(Play),ถอยหลัง(Review),เดินหน้า (Fast Forward),หยุดชั่วคราว(Pause),หยุด(Stop) และบันทึก (Record) ได้ สำหรับปุ่มบันทึกนี้ออกแบบเผื่อสำหรับการพัฒนาต่อไปที่สามารถรับอินพุทส่วนรีลไทม์ได้โดยตรง นอกจากนั้นยังมีตัวเลื่อนสำหรับปรับความเร็วช้า (Tempo Slide) ของเพลงที่เล่นได้ตามต้องการมีตัวเลื่อนความสูงต่ำของเสียง (Key Slide) เป็นตัวปรับเสียงที่ออกมาให้สูงต่ำเข้ากับเสียงนักร้องได้ รวมทั้งมีปุ่มสำหรับการเลือกที่จะเล่นเครื่องดนตรีชิ้น (แชนแนล) ใด หรือเล่นพร้อมกันทุกชิ้นก็ได้ โดยแสดงลำดับตัวโน๊ต ห้อง และอีเวนต์ให้ผู้ใช้ทราบตลอดเวลา

ข้อจำกัด ยังไม่สามารถให้ข้อมูลของแต่ละแชนแนลออกมาได้ ซึ่งอันที่จริงแล้วควรยอมให้ผู้ใช้เข้าไปแก้ไขในแต่ละแชนแนลได้ และความสามารถอีกอย่างที่ควรจะมีคือ การจำลองเสียงออกทางลำโพงของเครื่องโดยไม่จำเป็นต้องต่อส่วนเชื่อมต่อหรือเครื่องดนตรีจริง ทั้งนี้เพราะเวลาในการทำงานมีไม่พอ



รูปที่ 3.15 หน้าจอของแคสเซทเพลย์เยอร์

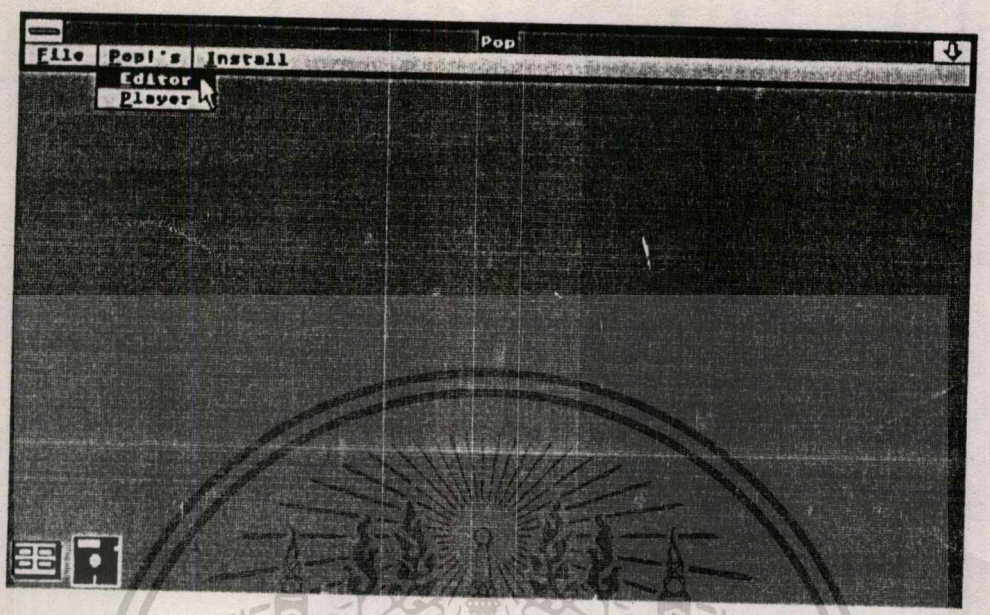


รูปที่ 3.15 หน้าจอของแคสเซทเพลย์เยอร์(ต่อ)

3.2.4 โปรแกรมป๊อป (POP) หรือซองแมเนเจอร์ (Song Manager)

จุดประสงค์ เพื่ออำนวยความสะดวกให้กัน 2 โปรแกรมข้องต้น เช่น การติดตั้ง (Installation) อุปกรณ์หรือเครื่องดนตรีชิ้นใหม่ ให้ 2 โปรแกรมนั้นรู้ถึงรายละเอียดต่างของอุปกรณ์ชิ้นนั้นให้เพียงพอที่จะนำมาใช้ประโยชน์ และควบคุมการบรรเลงของเครื่องดนตรีแต่ละชิ้นให้รวมเป็นเพลง การแปลงโน๊ตแบบสเตปไทม์ให้เป็นรีลไทม์ และเพื่อลดขนาดของข้อมูลในส่วนของโน๊ตอิดิเตอร์

ความสามารถ เป็นโปรแกรมที่ต้องนำมารันก่อน โน๊ตอิดิเตอร์ และแคสเซทเพลย์เยอร์ โดยจะมีลักษณะเป็นช่องหน้าต่างเล็กๆอยู่บนสุดของจอภาพ และจะมีอยู่ตลอดเวลาของการใช้งานโปรแกรมนี้ มีความสามารถในการแปลง (Translate) โน๊ตจากรูปแบบของโน๊ตอิดิเตอร์ ให้ไปเป็นรูปแบบของแคสเซทเพลย์เยอร์ เหตุที่ต้องใช้รูปแบบของโน๊ตที่ต่างๆกันในสองส่วนก็เพราะความง่ายในการทำงานของโปรแกรมนั้น เช่น รูปแบบของโน๊ตของโน๊ตอิดิเตอร์ก็สามารถแปลงจากโน๊ตบนจอลงมาได้ง่าย นอกจากนั้นยังสามารถจัดการติดตั้งเครื่องดนตรีดิจิตอลให้ทั้งสองโปรแกรมรู้จัก

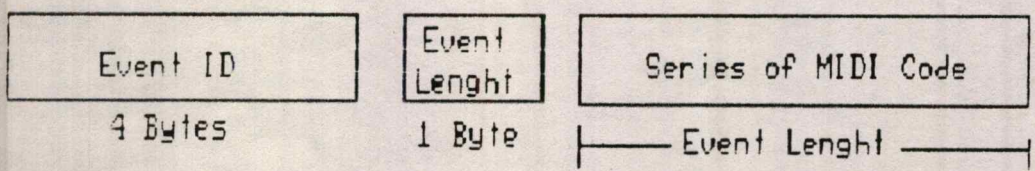


รูปที่ 3.16 หน้าจอของซองแมเนเจอร์

3.2.5 โปรแกรมไดรฟเวอร์ (Driver)

จุดประสงค์ เพื่อให้ส่วนเชื่อมต่อ (MIDI Insterface) สามารถทำงานได้ภายใต้โปรแกรมที่มีสภาพแวดล้อม (Environment) ต่างๆได้ ทั้งภายใต้ไมโครซอฟต์วินโดว์และภายใต้สภาพแวดล้อมอื่น เช่น เอ็มเอสดอส

ความสามารถ เป็นโปรแกรมแบบฝังตัว (Resident) โดยจะต้องทำการเรียกก่อนโปรแกรมอื่นทั้งหมด ซึ่งจะถูกเรียกใช้โดยผ่านอินเตอร์รัพท์ฟังก์ชั่น 104 สามารถควบคุมให้ส่วเชื่อมต่อเล่นเพลงจากข้อมูลมิดิออกทางมิดิพอร์ทได้ถึง 16 แชนแนล(สูงที่สุดตามความาสามารถของมาตรฐานมิดิ) รวมทั้งควบคุมการเล่นเพลงในลักษณะต่างๆ การตั้งตำแหน่งเพลงใหม่ การหยุดเพลง ส่วนรูปแบบของมิดิโค๊ดในส่วนนี้จะเป็นดังรูป



รูปที่ 3.17 ลักษณะของมิดิโค๊ด

ข้อสังเกตุ 1 อีเวนต์จะกินเวลา 1/96 ของ 1 เทมโป ซึ่งมีความละเอียดพอที่จะเป็นรีลไทม์ได้ (ตามความสามารถของมนุษย์ที่จะเล่นได้ทัน)

บทที่ 4
การทดลองและผลที่ได้

ผลการทดลองที่ได้ในโครงงานนี้สามารถแบ่งได้เป็นสองส่วน คือในส่วนของเทอมแรกและในส่วนของเทอมสอง ซึ่งจะขอกล่าวถึงทีละส่วน

4.1 **การทดลองและผลในเทอมแรก** สามารถแบ่งออกได้เป็น 2 ส่วนใหญ่ๆ คือ

4.1.1 **การทดลองวงจรเชื่อมต่อ** ในเทอมแรกนั้นวงจรเชื่อมต่ออยู่จะทดลองบนแผ่นโปรโตบอร์ดเพื่อความคล่องตัวในการปรับปรุงเปลี่ยนแปลงต่างๆ ในขั้นแรกของการทดลอง คือ การทดสอบความพร้อมในการทำงานของอุปกรณ์ต่างๆ คือ ทดสอบการส่งข้อมูลแบบSerial ให้ได้ความเร็วและรูปแบบที่ต้องการ และทดสอบการทำงานของส่วน หลังจากนั้นได้ทดลองส่งรหัสมิดิแบบง่ายๆเพื่อติดต่อกับคีย์บอร์ด โดยในระยะแรกไม่สามารถติดต่อได้เพราะสายสัญญาณไม่ดีพอ หลังจากที่แก้ไขแล้วก็สามารถติดต่อได้ ในขั้นสุดท้ายจึงได้มีการเชื่อมกับส่วนโปรแกรมจริงๆ

4.1.2 **การทดลองส่วนโปรแกรม** ได้ทำการเขียนโปรแกรมในลักษณะของ Step time player / Sequencer เพื่อทดสอบการเก็บข้อมูลและจังหวะของเสียงเพลง ด้วยภาษา ซีผลปรากฏเป็นที่น่าพอใจโดยไม่ผิดพลาดในจังหวะที่ค่อนข้างเร็ว โดยในขั้นต้นนี้ โปรแกรมยังไม่สามารถใส่ข้อมูลเป็นโน๊ตโดยตรงได้ จึงจำเป็นต้องใส่ข้อมูลเป็นรหัสไว้ก่อนในขั้นต้น

4.2 **การทดลองและผลในเทอมที่สอง**

งานที่ทำในเทอม 2 ส่วนใหญ่จะเป็นการออกแบบและพัฒนาโปรแกรม งานในส่วนฮาร์ดแวร์ที่ต้องทำนั้นคือ การนำอุปกรณ์ลงบนการ์ดที่สามารถเสียบบนสล็อตของเครื่องพีซีได้ ซึ่งปัญหาทางด้านฮาร์ดแวร์ที่เกิดขึ้นในเทอมนี้มีน้อยมาก ส่วนมากเป็นปัญหาที่เกิดจากความบกพร่องของการต่อวงจร งานที่ได้ทำเพิ่มขึ้นจริงๆคือ การออกแบบวงจรส่วนการเปลี่ยนทางเดินของการอินเตอร์รัพต์ให้สามารถเลือกได้ด้วยซอฟต์แวร์ว่าจะยอมให้ 8253 หรือ Z-80 DART สามารถที่จะส่งสัญญาณเข้าไปอินเตอร์รัพต์หน่วยประมวลผลได้ ผลที่ได้คือ สามารถทำงานได้ตามต้องการ ดังที่ได้กล่าวมาแล้วว่า การทดลองและผลที่ได้ส่วนใหญ่เกิดจากงานส่วนซอฟต์แวร์ ส่วนที่เหลือของบทนี้จึงเป็นการกล่าวถึงด้านการออกแบบและพัฒนาโปรแกรมเพียงอย่างเดียว

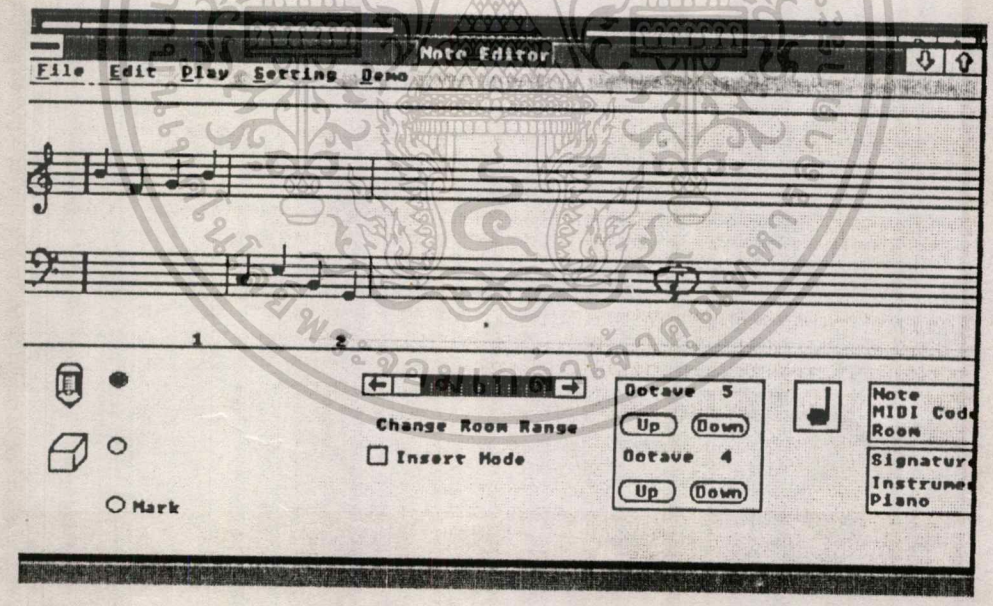
จากการที่ได้ทดลองออกไปสำรวจความเห็นในเรื่องการใช้คอมพิวเตอร์ในงานด้านนี้จากนักดนตรีหลายๆแขนง ทำให้ได้ทราบว่านักดนตรีส่วนใหญ่นั้นมักจะไม่ค่อยมีความรู้ด้านการเขียนโน๊ต ผู้ที่มีความเชี่ยวชาญชำนาญในด้านนี้มักจะเป็นผู้ประพันธ์เพลง (Composer) หรือผู้ที่ทำงานด้านดนตรีดิจิตอลจริงๆ กอรปกับได้ไปค้นคว้าจากโปรแกรมในแนวนี้ที่มีอยู่ในท้องตลาดไม่ว่าจะเป็น เครื่องของแมคอินทอช เครื่องของอมิก้า หรือของอะตาริ ทำให้สามารถเปรียบเทียบข้อบกพร่องและข้อดีของแต่ละโปรแกรมเมื่อนำมาผสมผสานกับความเห็นที่ได้มาจากนักดนตรีก็ทำให้

สามารถออกแบบระบบในแนวของตนเองขึ้นมาได้

การทดลองส่วนโปรแกรม

ในเทอมนี้ได้เริ่มเขียนโปรแกรมภาษาซีที่ทำงานภายใต้ไมโครซอฟต์วินโดว์ โดยในช่วงแรกการเขียนโปรแกรมเป็นไปได้ด้วยความยากลำบาก เพราะถึงแม้โปรแกรมที่ทำงานใต้วินโดว์นั้นจะทำให้ผู้ใช้สามารถใช้ได้ง่าย แต่แนวความคิดของการเขียนโปรแกรมแตกต่างไปจากธรรมดามาก เช่น ไม่มีการเรียกใช้โปรแกรมย่อย การทำงานพร้อมกันของทุกวินโดว์ การไม่สามารถใช้ทรัพยากรของเครื่องได้อย่างเต็มที่ แต่เมื่อเริ่มเข้าใจแล้วทำให้การเขียนโปรแกรมเป็นไปอย่างสะดวกมาก

การทดลองในส่วนของโน๊ตอิดิเตอร์นั้น ส่วนมากจะเป็นการทดลองในเรื่องของการใช้ทฤษฎีดนตรีที่มีอยู่ให้สามารถอำนวยความสะดวกให้กับผู้ใช้ให้มากที่สุด เช่น การกั้นห้องให้โดยอัตโนมัติ ก็ให้ผลว่าสามารถทำการกั้นได้อย่างสมบูรณ์ พร้อมทั้งการแตกโน๊ตที่เกินออกมาของห้องให้เป็นโน๊ตสองตัวที่มีเครื่องหมายทาย (Tie) เสียงเชื่อมได้ พร้อมกันนั้นก็สามารถเก็บโค๊ดของโน๊ตที่ทำการแก้ไขลงในไฟล์โดยมีรูปแบบที่แน่นอนดังตัวอย่าง



รูปที่ 4.1 การแปลงโน๊ตจากหน้าจอเป็นโค๊ดของโน๊ตเหล่านั้น

จากรูปเมื่อทำการแปลงเรียบร้อยแล้ว เก็บลงไฟล์เมื่อเรียกออกมาดูจะเห็นเป็นโค๊ดดังนี้

4A 00 80 17 4C 00 80 17 4F 00 80 17 50 00 80 17

30 00 80 17 34 00 80 17 2f 00 80 17 2b 00 80 17

โดยที่ไบท์แรกจะเป็นชื่อของโน๊ตตัวนั้น (ตามมาตรฐานของมิดิ) ส่วนไบท์ถัดมาเป็น ชนิดของโน๊ตตัวนั้น ในโปรแกรมนี้ได้กำหนดไว้ให้

00 หมายถึง โน๊ตตัวดำ

01 หมายถึง โน๊ตตัวขาว

02 หมายถึง โน๊ตตัวกลม

03 หมายถึง โน๊ตขเบ็จ 1 ชั้น

04 หมายถึง โน๊ตขเบ็จ 2 ชั้น

05 หมายถึง โน๊ตขเบ็จ 3 ชั้น

ส่วนในไบท์ที่สามเป็นโค๊ดของอีเวนต์ โดยที่จะบอกว่าไบท์ถัดมา(ไบท์ที่ 4) เป็นจำนวนอีเวนต์ที่ว่าง

การทดลองในส่วนแคสเซ็ทเพลเยอร์ ได้มีการทดลองฟังก์ชันการทำงานพื้นฐาน เช่นการเพลย์ การถอยหลังและการเซ็ทตัวนับ โดยมีการจำลองปุ่มต่างๆของเครื่องเล่นเทป ให้ผู้ใช้รู้สึกคุ้นเคย พร้อมทั้งมีการทดลองนำเอาสไลด์สำหรับเปลี่ยนจังหวะ และเสียงสูงต่ำของเพลงที่จะเล่นได้ และสามารถเลือกฟังเครื่องดนตรีทีละชิ้น บางชิ้นหรือพร้อมกันหมดทุกชิ้นได้ สำหรับผลการทดลองในส่วนนี้ปรากฏว่าได้ผลตามที่ต้องการทุกฟังก์ชัน จากการเปลี่ยนจากโน๊ตบนจอมาเป็นรูปแบบการเก็บของโน๊ตอิดิเตอร์ ซึ่งมีการเก็บเป็นแบบสเตปไทม์ เมื่อนำมาผ่านโปรแกรมแมเนเจอร์แล้วนี้แล้วก็จะได้รูปแบบของโน๊ตเหล่านั้นที่เป็นรูปแบบของรีลไทม์ โดยจากตัวอย่างข้างบนสามารถแปลงออกมาได้เป็น

```
00000000  03  90  4A  50  <-- โน๊ตออน
00000058  03  80  4A  00
00000060  03  90  4C  50  <-- โน๊ตออน
000000B8  03  80  4C  00
000000C0  03  90  4F  50  <-- โน๊ตออน
00000118  03  80  4F  00
00000120  03  90  50  50
00000178  03  80  50  00
00000180  03  90  30  50
000001D8  03  80  30  00
000001E0  03  90  34  50
00000238  03  80  34  00
00000240  03  90  2F  50
```

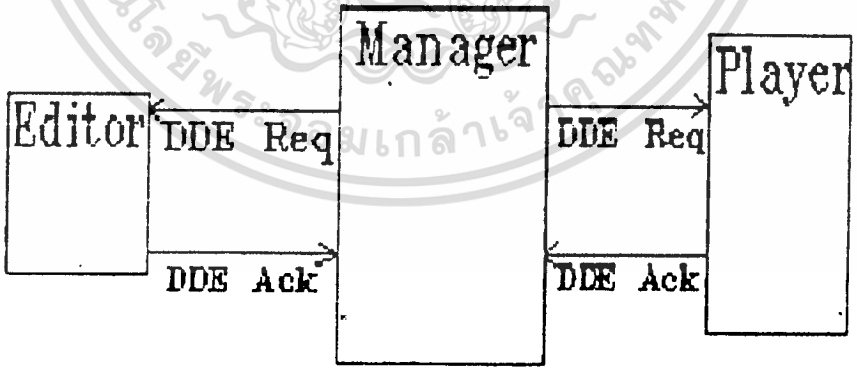
```
00000298   03   80   2F   00
000002A0   03   90   2B   50   <-- โน๊ตออน
000002F8   03   80   2B   00
```

สามารถอธิบายความหมายของไบท์ต่างๆได้ดังนี้

4 ไบท์แรกเป็นค่าของอีเวนต์ที่เหตุการณ์นั้นเกิดขึ้น โดยที่ความละเอียดของอีเวนต์ละเอียดถึง 1/96 ของ 1 จังหวะ และสามารถครอบคลุมเพลงได้ ถึง 194 ชั่วโมงที่เทมโป้ 240 ส่วนที่เทมโป้ 110 จะได้ถึง 6778 ชั่วโมง ส่วนในไบท์ที่สองจะเป็นค่าของความยาวของมิดิโค๊ดที่จะส่งออกไปที่เหตุการณ์นี้ ต่อจากนั้นเป็นมิดิโค๊ดที่มีอยู่ในเหตุการณ์ช่วงนั้นทั้งหมด

การทดลองในส่วนไดรเวอร์ ได้ทำการใส่ฟังก์ชั่นการทำงานพื้นฐานเช่น การตั้งตำแหน่งของเพลงที่จะเริ่มเล่น การตั้งตำแหน่งของเพลงที่จะเลิกเล่น เปลี่ยนคีย์ของเพลง ตั้งค่าตัวชี้ตำแหน่งในเพลงได้ และสามารถที่จะเปลี่ยนคีย์ของเพลงได้ ผลการทดลองปรากฏว่าได้ผลเป็นที่น่าพอใจคือสามารถรับส่งโค๊ดมิดิกับเครื่องดนตรีได้

เนื่องจากตัวไมโครซอฟต์วินโดว์มีการทำงานแบบมัลติทาส์ค การเขียนโฟลว์ชาร์ตของการทำงานจึงไม่สามารถทำได้ แต่หากมองเป็นภาพรวมของการทำงานร่วมกันของทั้งสามโปรแกรมสามารถเขียนเป็นรูปภาพได้ดังรูปที่ 4.2 และ 4.3

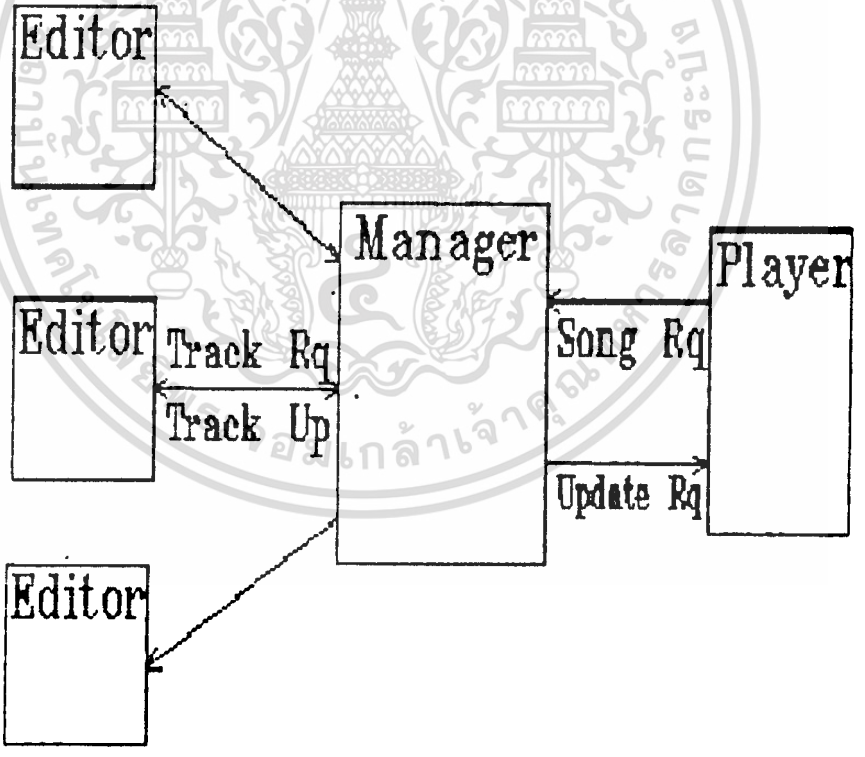


รูปที่ 4.2 การเริ่มต้นการทำงาน

เมื่อเริ่มเข้าสู่โปรแกรม โปรแกรมแมเนเจอร์ส่งสัญญาณ (DDE Request) ออกไปยังอีก 2 โปรแกรม ซึ่งก็จะมีสัญญาณตอบกลับด้วย (DDE Ack) ส่วนในตัวโน๊ตอิดิเตอร์จะมีสัญญาณการร้องขอแทรก (Track Request) ถ้ามีข้อมูลดนตรีอยู่ในขณะนั้นตัวแมเนเจอร์ก็จะส่งสัญญาณ

Track Updated กลับไปให้ตัวอิดิเตอร์ ก็จะเป็นการสิ้นสุดกระบวนการเริ่มต้นการทำงาน

หลังจากนั้นการทำงานของโปรแกรมก็จะเข้าสู่การทำงานในแบบปกติ (รูปที่ 4.3) เมื่อโปรแกรมเพลเยอร์ต้องการเล่นเพลงออกทางมิดิพอร์ทก็จะส่งสัญญาณร้องขอ (Request) ไปยังโปรแกรมแมเนเจอร์ หลังจากนั้นตัวโปรแกรมแมเนเจอร์ก็จะไปถามโปรแกรมอิดิเตอร์แต่ละตัวที่ทำงานอยู่ในขณะนั้นว่าข้อมูลมีการเปลี่ยนแปลงหรือไม่ ถ้ามีการเปลี่ยนแปลงก็จะรับข้อมูลใหม่มาและทำการแปลส่งไปให้โปรแกมเพลเยอร์ เมื่อการเก็บหรือดึงข้อมูลที่เป็นเพลงจากไฟล์ ตัวแมเนเจอร์ก็จะร้องขอไปยังตัวอิดิเตอร์ที่กำลังทำการแก้ไขเพลงในแต่ละแทรกอยู่ให้ทำการเก็บข้อมูลทุกแทรกนั้นลงในเพลงเดียวกัน เพื่อเก็บลงไฟล์เดียวกัน หรือถ้ามีการอ่านเพลงขึ้นมาจากไฟล์ โปรแกรมแมเนเจอร์ก็จะทำการแบ่งเพลงออกเป็นแทรกเพื่อทำการแจกจ่ายให้อิดิเตอร์แต่ละตัวทำการแก้ไขแต่ละแทรก (ดังรูปที่ 4.3) หากตอนนั้นตัวอิดิเตอร์มีไม่พอกับแทรกที่อ่านขึ้นมา ตัวแมเนเจอร์ก็จะทำการเรียก(Spawn)โปรแกรมอิดิเตอร์ตัวใหม่ออกมา



รูปที่ 4.3 การทำงานในภาวะปกติ

บทที่ 5

สรุป ปัญหาและแนวทางการพัฒนาต่อ

เนื่องจากโครงงานนี้มีลักษณะงานที่แยกกันอย่างเห็นได้ชัด คือในส่วนฮาร์ดแวร์ (วงจรเชื่อมต่อมิดิ) และ ส่วนซอฟต์แวร์ คือ โปรแกรมทั้ง 4 โปรแกรมที่ได้กล่าวมาในบทก่อนๆแล้ว การสรุปจึงขอสรุปเป็น 2 ส่วนใหญ่ๆ คือ

5.1 ทางด้านฮาร์ดแวร์

วงจรที่ได้ออกแบบและดัดแปลงมาสามารถทำงานได้เป็นที่น่าพอใจ คือ จากของเดิมที่มี 8253 ทำหน้าที่นับเวลาให้ระบบ และมี Z-80 DART เป็นตัวส่งข้อมูล ส่วนฮาร์ดแวร์ที่ได้ทำขึ้นมาใหม่นี้ก็ได้มีการดัดแปลงเอาสัญญาณอินเตอร์รัพต์มาจาก 8253 แทนของเดิมที่มาจากตัว Z-80 DART เพื่อนำไปใช้สร้างความเร็วช้าให้กับเพลงที่เล่น โดยจะมีผลดีคือ สามารถที่จะเล่นเพลงไปพร้อมๆกับงานอื่นโดยไม่ทำให้จังหวะของเพลงเปลี่ยนไป ส่วนที่ทำการดัดแปลงต่อไปเป็นการทำให้สามารถเลือกอุปกรณ์ที่จะส่งสัญญาณอินเตอร์รัพต์ซีพียูได้ด้วยซอฟต์แวร์ ทำให้ตัว Z-80 DART สามารถอินเตอร์รัพต์ได้อีกครั้ง เพื่อสำหรับการพัฒนาโปรแกรมให้มีการรับสัญญาณมิดิจากภายนอก (ความสามารถของการ์ดในขณะนี้สามารถรับข้อมูลได้อย่างถูกต้องแล้ว)

ปัญหาที่เกิดขึ้นในส่วนฮาร์ดแวร์ของเทอมนี้ มีปัญหาที่เกิดขึ้นเพียงอย่างเดียว คือ การอ่านค่าจากเฟลกของ Z-80 DART เพื่อทำการตรวจสอบสถานะของบัฟเฟอร์ว่าว่างที่จะให้ส่งข้อมูลไปได้หรือยัง จึงเลี่ยงไปใช้วิธีคำนวนเวลาที่ข้อมูลจะถูกส่งออกไปหมดแล้วจึงส่งข้อมูลตัวต่อไปไปใหม่

แนวทางพัฒนาต่อ ทางด้านฮาร์ดแวร์ ได้แก่การเพิ่มความสามารถให้แก่วงจรอินเทอร์เฟสให้สูงขึ้น เช่น การเพิ่มตัวประมวลผลลงบนวงจรเพื่อให้การทำงานของซีพียูในเครื่องเป็นอิสระกับงานด้านนี้มากขึ้น (คือไม่ต้องมาคอยทำรูทีนบริการอินเตอร์รัพต์ให้กับโปรแกรมทางด้านนี้) หรืออาจจะพัฒนาไปถึงการสร้างการ์ดที่มีความใช้กันได้ (Compatible) กับอุปกรณ์เชื่อมต่อมิดิที่มีความนิยมสูง (MPU-401) เพื่อที่สามารถรันโปรแกรมอื่นๆที่ทำงานกับเครื่องรุ่นนี้ได้ทันที

5.2 ทางด้านซอฟต์แวร์

โปรแกรมที่เขียนสามารถทำงานได้ตามโครงการที่ตั้งเอาไว้ คือสามารถทำการบันทึกเพลงได้ในแบบสเตปไทม์โดยมีลักษณะการนำข้อมูลเข้าได้ง่ายและมีประสิทธิภาพ ในรูปของการวางโน๊ตลงบนบรรทัด 5 เส้น โดยมีทฤษฎีดนตรีพื้นฐานช่วยอำนวยความสะดวก สามารถเล่นกลับพร้อม ๆ กับทำงานอื่นได้ในหลายๆ รูปแบบ รวมทั้งจัดเก็บข้อมูลที่บันทึกไว้ลงในแฟ้มข้อมูลบนแผ่นดิสก์เพื่อการนำกลับมาใช้อีก

ได้พบปัญหาสำคัญคือ เอนเวอร์รอนเมนท์ที่แปลกไปจากเดิมของไมโครซอฟต์วินโดว์ ทำให้การพัฒนาโปรแกรมในบางครั้งขลุกขลัก แต่ทำให้การพัฒนาที่จำเป็นต้องอาศัย อุปกรณ์ชี้ตำแหน่ง,

ระบบการแสดงผลแบบกราฟฟิก และ ระบบการติดต่อกับผู้ใช้ ง่ายขึ้นกว่าเดิมมาก รวมทั้งประสบการณ์แปลกใหม่ในการพัฒนาโปรแกรม

แนวทางพัฒนาต่อ คือการเพิ่มระบบการบันทึกแบบรีลไทม์ เติมในส่วนของคาสเซตเพลย์เยอร์ให้ก้าวหน้ามากขึ้น ในส่วนของโน๊ตอิดิเตอร์อาจเพิ่มทฤษฎีดนตรีที่ก้าวหน้ามากขึ้น หรืออาจเพิ่มเติมในลักษณะของความฉลาดเทียมเพื่อช่วยในการแต่งเพลงได้ง่ายๆ และอาจเพิ่มเติมการแสดงผลจากการบันทึกในระบบรีลไทม์ ในส่วนของโน๊ตอิดิเตอร์หรือซองนมเนเจอร์ เพื่อให้นักดนตรีที่ไม่รู้จักโน๊ตสามารถเข้าใจตัวโน๊ตต่างๆ

ภาคผนวก ก.

9.5 THE ZILOG Z-80 SIO AND Z-80 DART

For serial communications the Z-80 is supported by the *Z-80 SIO serial input/output controller* and the *Z-80 DART dual asynchronous receiver/transmitter.* Both devices provide two separate and independent serial communications channels. This means that one channel can be programmed to communicate with a 19,200-baud VDT while the other is interfaced to a modem at 300 baud, for example.

The SIO has both asynchronous and synchronous capabilities, including compatibility with the bisync, SDLC, and HDLC synchronous protocols. Automatic CRC checking and generation is also provided in this mode. For asynchronous-mode-only applications, the Z-80 DART should be selected. It has all of the SIO's asynchronous capabilities and is pin-compatible with the Z-80 SIO/0 version of the SIO.

Similar to other Z-80 support devices, all control functions of the SIO and DART are programmable by the microprocessor. Several status registers are provided that allow monitoring of all important UART flags and error conditions. Also supported is the (standard technique for all Zilog peripheral controllers) Z-80 mode 2 interrupt scheme, including the daisy-chain-priority structure.

Clock multipliers of ×1, ×16, ×32, and ×64 are programmable and data rates up to one-fifth of the system clock frequency are possible. A Z-80A SIO with a 4-MHz clock can operate at 800K baud in the synchronous mode and as high as 50K baud in the asynchronous mode (×16 clock).

Comparing the SIO and the DART. Figure 9.22 shows the pin assignments for the three versions of the SIO and the single version of the DART. As mentioned, the DART pinning is identical to the SIO/0 option except that pins 11 and 29 are

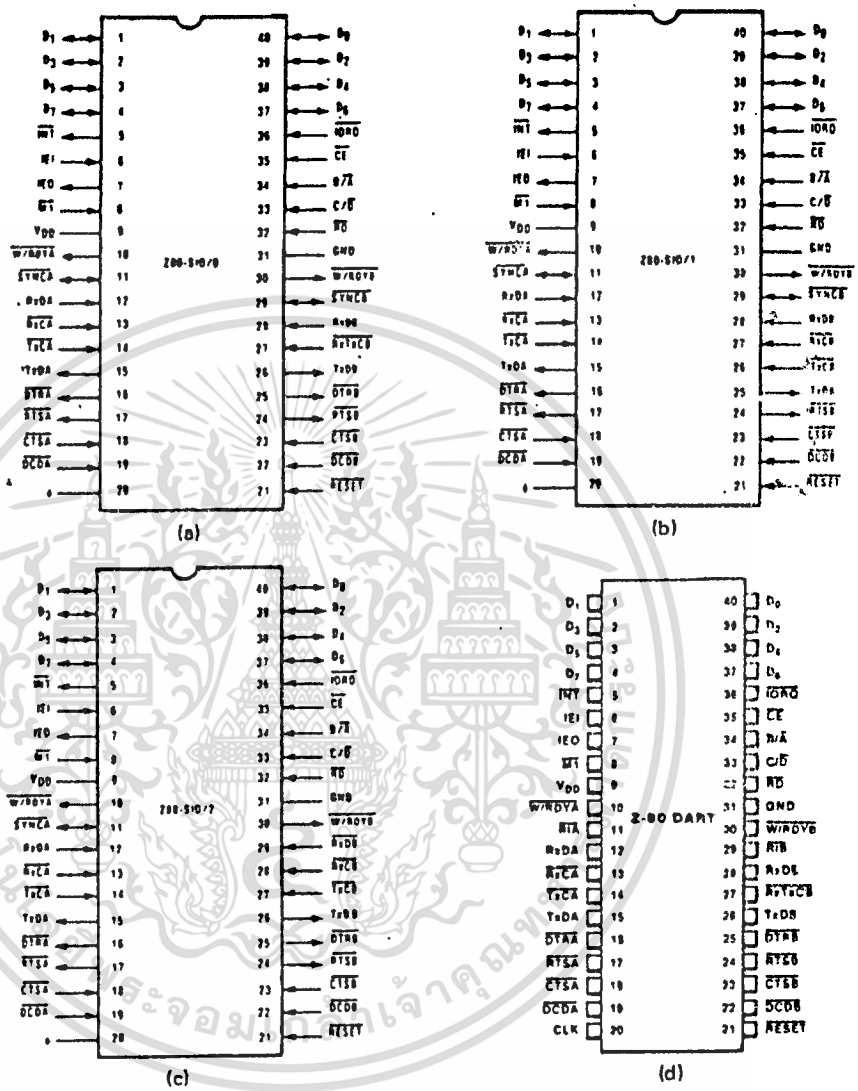


Figure 9.22 (a)–(c) Pin assignments for the three versions of the Z-80 SIO; (d) single version of the Z-80 DART. The SIO/0 and DART have identical pin assignments. (Courtesy of Zilog, Inc.)

labeled $\overline{\text{SYNCA}}$ and $\overline{\text{SYNCB}}$ on the SIO and $\overline{\text{RIA}}$ and $\overline{\text{RIB}}$ on the DART. In the asynchronous mode (the only mode of operation for the DART) these pins are general-purpose inputs that have no special function. Indeed, they could be connected to a ring indicator signal from a modem (hence the DART signal names).

When operated in the synchronous mode (the SIO) $\overline{\text{SYNCA}}$ and $\overline{\text{SYNCB}}$ signals indicate reception of valid sync characters. More details on the SYNC pins will be provided later in this section.

The differences between the three versions of the SIO are:

1. The SIO/0 channel B receiver and transmitter have a common clock pin.
2. The SIO/1 lacks the $\overline{\text{DTRB}}$ signal.
3. The SIO/2 lacks the $\overline{\text{SYNCB}}$ signal.

Note that this means that channel B of the SIO/0 and the DART have a common transmitter and receiver clock but that channel A can have separate clocks. In the discussion to follow the Z-80 SIO/0 will be shown, but all comments regarding asynchronous operation apply equally to the Z-80 DART.

Interfacing the Z-80 SIO. Interfacing the Z-80 SIO to the Z-80 microprocessor is straightforward, as illustrated in Fig. 9.23. All control signals are directly compatible with the Z-80. Internally, the SIO decodes $\overline{\text{IORQ}}$, $\overline{\text{MI}}$, and $\overline{\text{RD}}$ to generate $\overline{\text{IOR}}$, $\overline{\text{IOW}}$, and $\overline{\text{INTA}}$ (see Table 8.3). All communications between the Z-80 and the SIO is done using the bidirectional data bus, and to the processor, the SIO appears to be four parallel I/O ports.

The B/$\overline{\text{A}}$ input selects channel A or B and the C/$\overline{\text{D}}$ input determines if the control or data registers will be examined. The $\overline{\text{CE}}$ input must be low for all I/O read or write operations with the processor (it need not be low when transmitting or receiving data). Table 9.5 lists specific port addresses based on the address decoding in Fig. 9.23.

For variety, a more elaborate address decoder than is required is shown. This circuit generates three additional chip-enable signals that could be used with additional SIO chips or other peripheral controllers, such as the Z-80 PIO or CTC. If this is done, the IEI and IEO daisy-chain controls should be used to establish interrupt priorities. In Fig. 9.23 the SIO is given highest priority by wiring its IEI input to +5 V.

Examining the serial interfaces, two channels are provided labeled Channel A and Channel B. Note that each provides separate serial in and out lines and has independent clock inputs. The SIO/0 option shown has the channel B receiver and transmitter pins internally connected.

As mentioned in Sec. 9.4, the baud rate can be controlled by changing the clock multiplier (×16, ×32, ×64) without changing the baud rate clock frequency. The baud rate generator itself can be a crystal-controlled TTL oscillator, a Z-80 CTC programmed for a specific baud rate, or a special baud rate generator IC such as the Motorola MC14411. Because these options were discussed in detail in Sec. 9.4, that material will not be repeated here.

Four modem control signals are provided for each channel: two inputs and two outputs. These are used to establish a handshaking protocol between the serial peripheral and the SIO. More detail will be provided in Sec. 9.7. Note, however, that the signals are identical to those provided on the Intel 8251 except that $\overline{\text{DSR}}$ is replaced with $\overline{\text{DCD}}$ on the SIO.

If the auto enables function is selected, $\overline{\text{DCD}}$ and $\overline{\text{CTS}}$ become the receiver and transmitter enables, respectively. If disabled, they function as general purpose inputs.

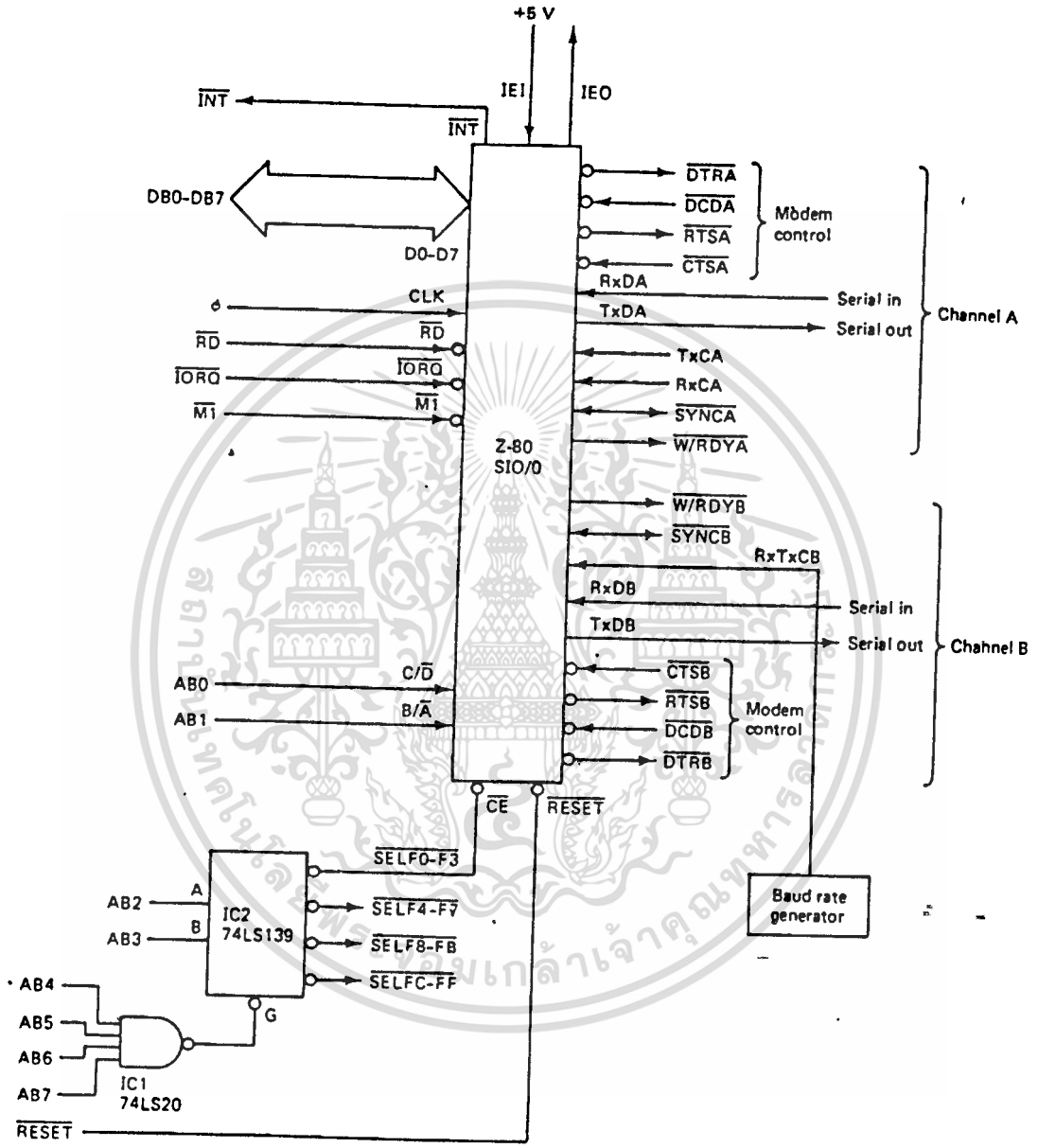


Figure 9.23 Interfacing the Z-80 SIO to the three-bus system architecture of the Z-80 microprocessor.

Each channel (of the SIO/0) also has two special-purpose control signals, $\overline{SYNC}$ and $\overline{W/RDY}$.

1. $\overline{SYNC}$ can be programmed to be an input or output signal when the SIO is operated in the synchronous mode. As an output it indicates valid sync characters

TABLE 9.5 PORT ADDRESSES FOR THE Z-80 INTERFACE IN FIG. 9.23

B/$\overline{A}$	C/$\overline{D}$	$\overline{IORQ}$	$\overline{RD}$	Figure 9.23 port address	Function	
0	0	0	0	F0H	Data read	Channel A
0	0	0	1	F0H	Data write	
0	1	0	0	F1H	Status read	
0	1	0	1	F1H	Control write	
1	0	0	0	F2H	Data read	Channel B
1	0	0	1	F2H	Data write	
1	1	0	0	F3H	Status read	
1	1	0	1	F3H	Control write	

are being received and could be used by non-Z-80 processors to initiate an interrupt request. As an input, $\overline{SYNC}$ allows an external signal to indicate synchronization and cause the SIO to begin capturing the synchronous data.

In the asynchronous mode $\overline{SYNC}$ is a general-purpose input that can be monitored via a status register. A typical application is to detect the ring signal output by a modem.

2. $\overline{W/RDY}$ is an output that can be programmed as an open drain *WAIT* request signal. It is used for block transfers to synchronize the data rate between the SIO and a DMA controller or the processor. It can also be programmed as a RDY signal compatible with the Z-80 DMA RDY input.

Programming the Z-80 SIO: Asynchronous Mode. Figures 9.24 and 9.25 describe the read and write registers of the Z-80 SIO. These are usually referred to as RR0–RR2 (read registers 0 through 2) and WR0–WR7 (write registers 0 through 7). Registers WR2 and RR2 can be accessed only when B/$\overline{A}$ = 1. This does not mean that the interrupt vector can be specified only for channel B. Rather, the interrupt vector is common to channels A and B and therefore need not be specified for both.

All other registers are duplicated for channel A and channel B. Note that WR6 and WR7 control the synchronous mode of operation exclusively and are not available in the Z-80 DART. In general, all bits dedicated to the synchronous mode are "do nothing" bits for the DART.

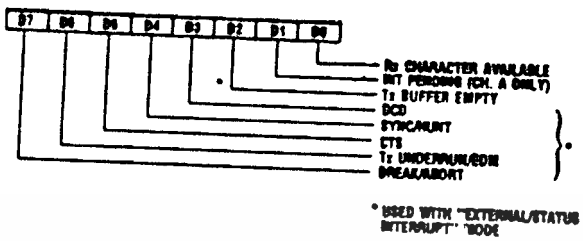
You may be wondering how eight write registers and three read registers can be accessed through one control port. The trick is to use WR0 as a *pointer register*. For example, by specifying D2–D0 as 010, RR2 will be accessed with the next control port read operation, and WR2 will be written to with the next control port write.

As with all programmable I/O devices, an initialization sequence must be followed before the device can be used. Figure 9.26 shows the suggested sequence for the SIO. As can be seen, WR0 must repeatedly be programmed to point at the desired register.

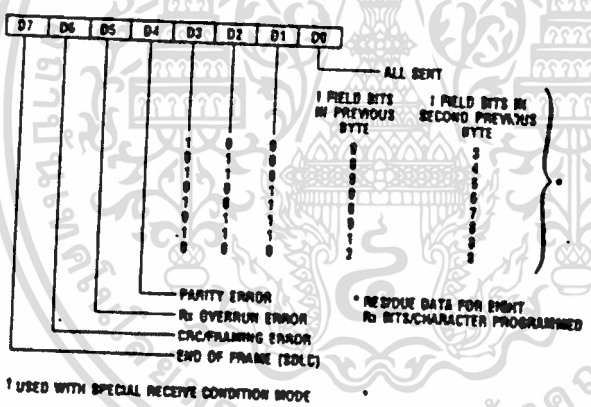
The SIO can be controlled using any of the familiar I/O techniques:

1. Polling
2. Interrupts
3. DMA

READ REGISTER 0



READ REGISTER 1†



READ REGISTER 2

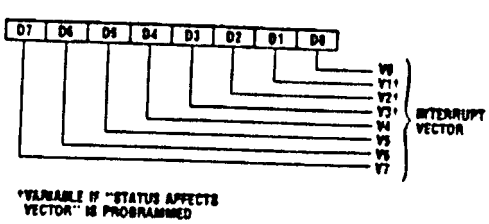


Figure 9.24 Z-80 SIO read register bit functions. (Courtesy of Zilog, Inc.)

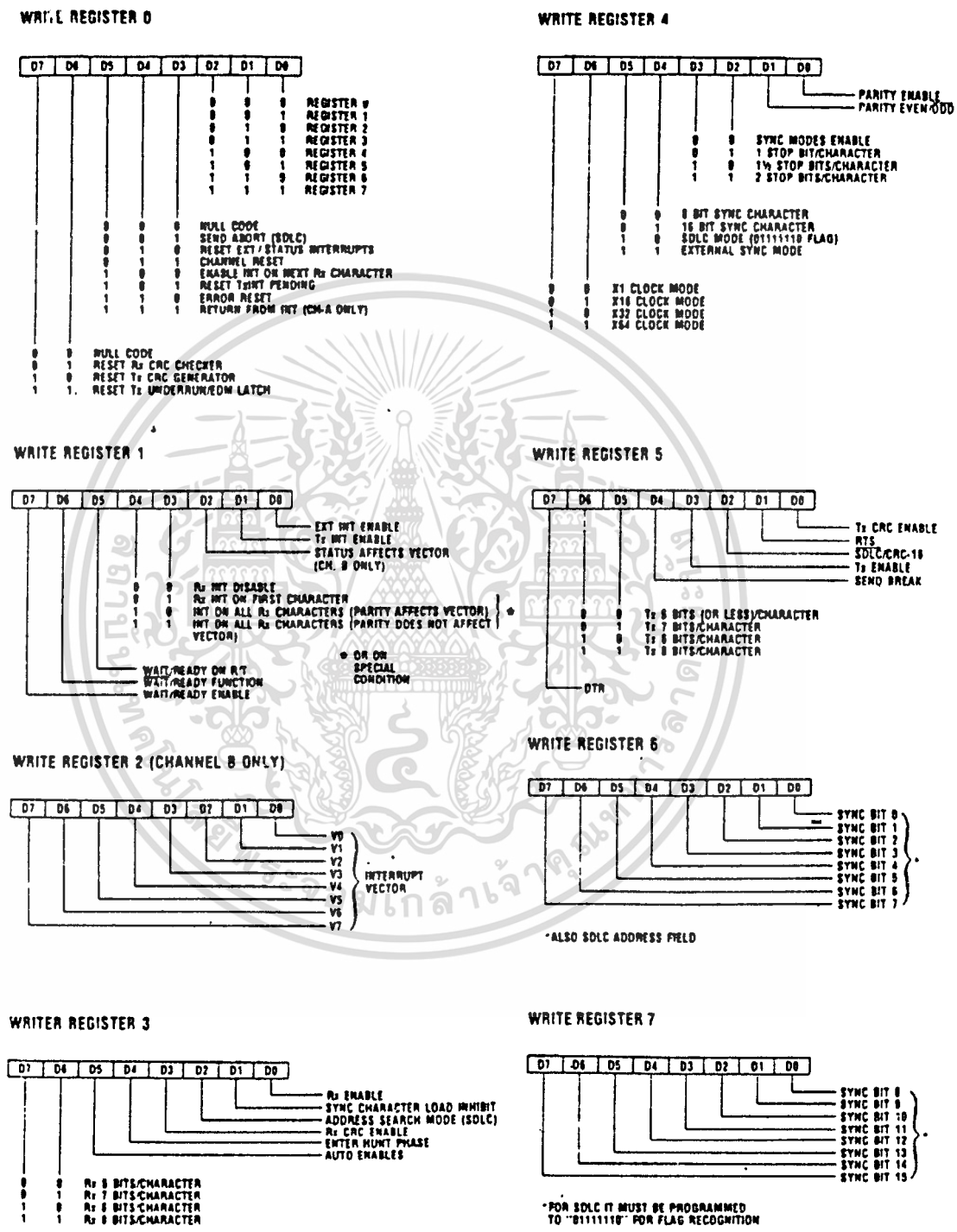


Figure 9.25 Z-80 SIO write register bit functions. Registers 6 and 7 are not available in the Z-80 DART. (Courtesy of Zilog, Inc.)

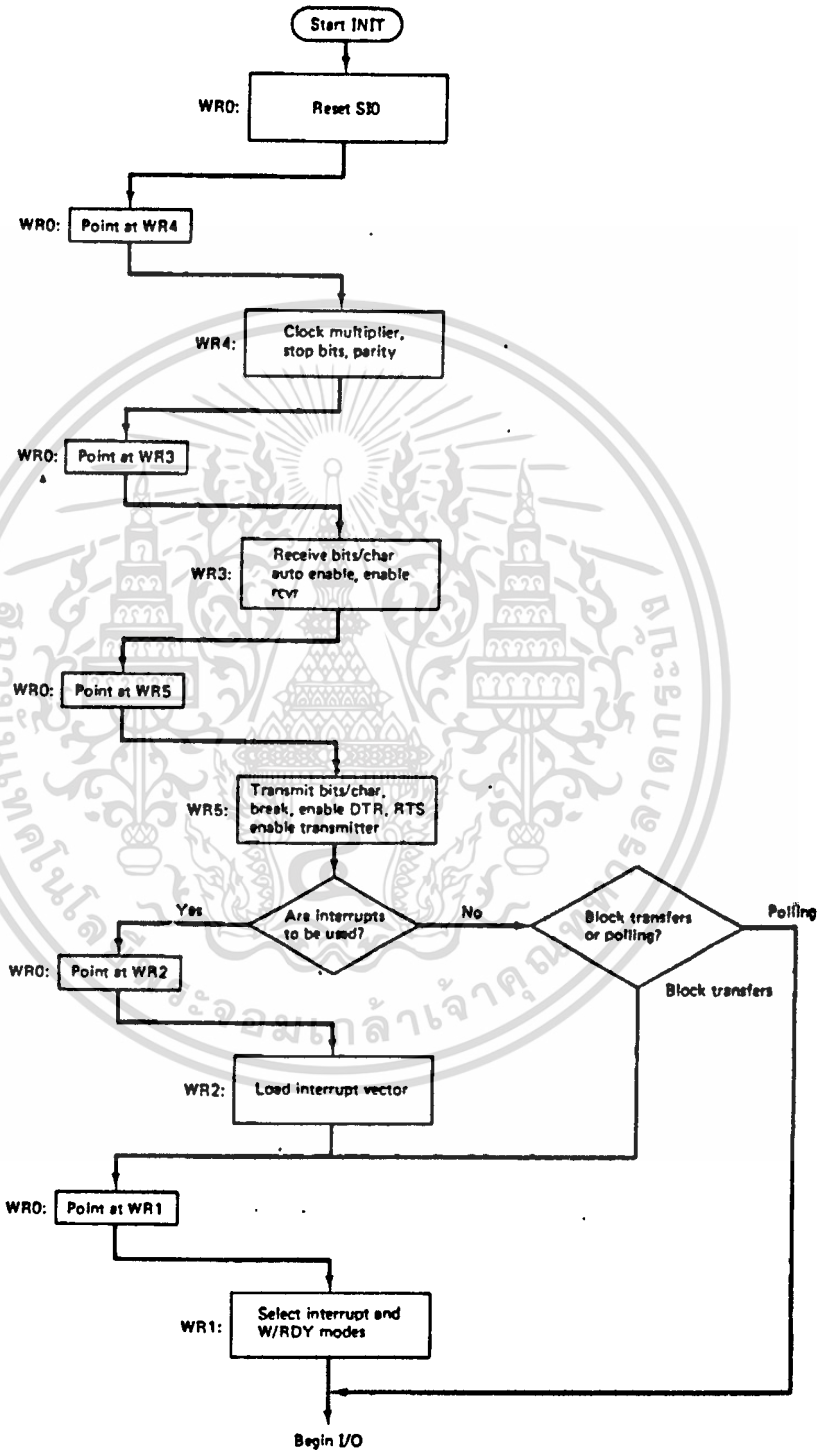


Figure 9.26 Suggested initialization sequence for the Z-80 SIO in the asynchronous mode.

The particular scheme used affects the last steps of the initialization sequence. As an illustration, consider the following example, which will program the SIO for simple polling.

Example 9.13

Determine the codes and write the program required to initialize channel B of the Z-80 SIO interface in Fig. 9.23 for asynchronous polled transfers. Use 8 data bits per character, odd parity, 1 stop bit, and a 64× clock multiplier. Disable the auto enables function.

Solution. Figure 9.27(a) lists the required sequence of codes. Figure 9.27(b) is the Z-80 control program which puts the OTIR instruction to good use.

Controlling the Z-80 SIO in the Asynchronous Mode

Polling. The simplest way to use the SIO (although not the most efficient) is with polling. Example 9.13 gives an example of the initialization required. Once initialized, receiving and transmitting programs must be written. Flowcharts for these programs are provided in Fig. 9.28.

	Register	Binary Code Required	Hex	Explanation
1.	WR0	00 011 000	18	Channel reset command sent to WR0.
2.	WR0	00 010 100	14	Reset status/interrupts and point at WR4.*
3.	WR4	11 XX 01 01	C5	64X clock, 1 stop bit, odd parity (trans and rcvr).
4.	WR0	00 010 011	13	Point to WR3.
5.	WR3	11 000001	C1	8 bits/char, $\overline{DCD}$ and $\overline{CTS}$ disable (auto enables disable), rcvr enable.
6.	WR0	00 010 101	15	Point to WR5.
7.	WR5	0 11 01000	68	No DTR or RTS, 8 bits/char, no break, trans enable.

*When any of the external lines $\overline{DCD}$, $\overline{CTS}$ or a Break condition occurs, the status bits of RR0 are latched. These bits should be reset during initialization. It is just as easy to do this with each write to WR0 and no harm is done. This is done in this example.

(a)

```
        ;Z-80 SIO ASYNCHRONOUS INITIALIZATION ROUTINE
        ;
        ;THIS PROGRAM INITIALIZES THE Z-80 SIO FOR 1 STOP
        ;BIT, ODD PARITY, 8 DATA BITS AND 64X CLOCK MODE.
        ;ALL MODEM CONTROL SIGNALS ARE DISABLED.
        ;
INSIO:  LD      C,0F3H          ;CHANNEL B CONTROL PORT
        LD      B,7             ;7 BYTES TO PROGRAM
        LD      HL,CODES        ;POINT HL AT CODE TABLE
        OTIR                    ;OUTPUT THE CONTROL CODES
        ;
        ;I/O CAN BEGIN HERE
        ;
CODES   DB      18H,14H,0C5H    ;THESE ARE THE INITIALIZATION
        DB      13H,0C1H,15H    ;CODES
        DB      68H
```

(b)

Figure 9.27 (a) Control codes required for Ex. 9.13; (b) Z-80 initialization routine.

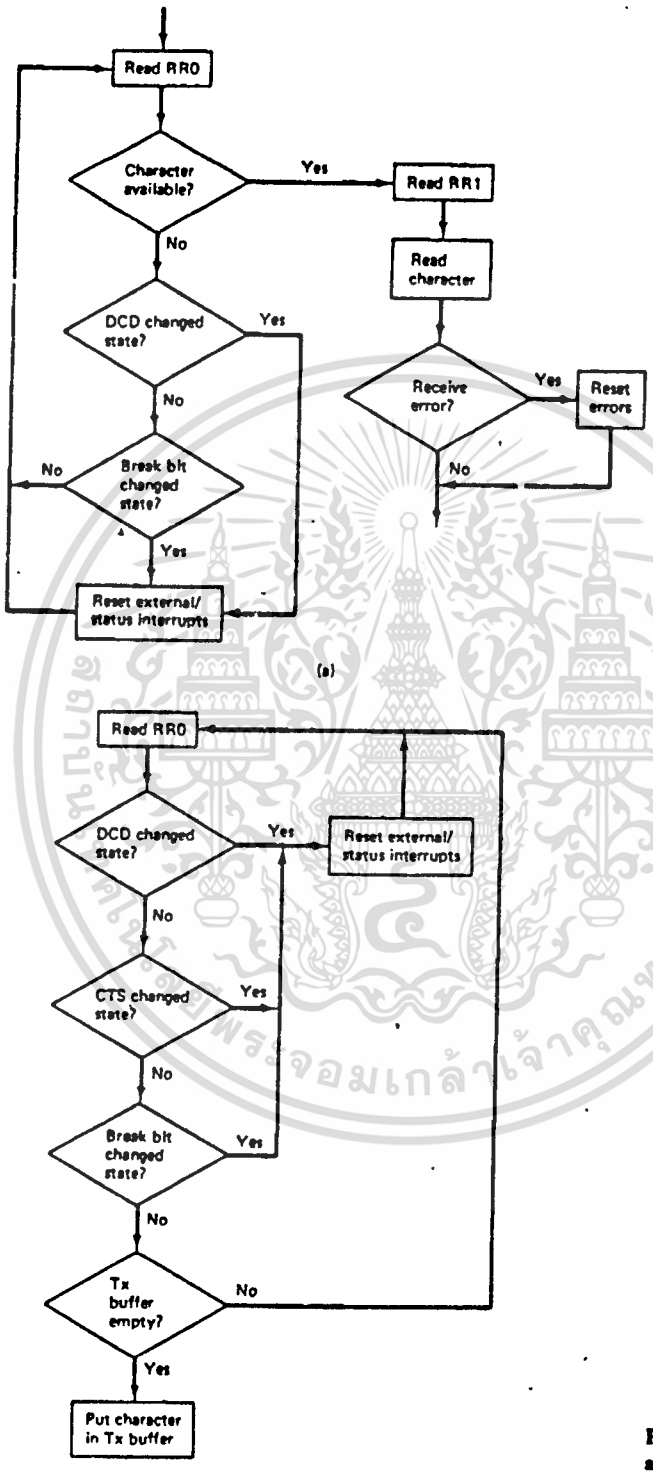


Figure. 9.28 Z-80 SIO flowcharts for asynchronous mode polling: (a) receiver; (b) transmitter.

Referring to the read register definitions in Fig. 9.24, bit 0 of RR0 is the receive data ready flag. When this bit is high, one to three data bytes are available to be read (the SIO has a three-character buffer). If bit 0 is not high, the $\overline{DCD}$ bit can be tested (this line is usually used by a modem to indicate that it has a valid carrier from the distant station) or the break condition tested. The break character is a continuous logic 0 level and is used to interrupt the transmitter.

When bit 0 goes high, RR1 should be read and its contents saved. As indicated in Fig. 9.24, this register stores the error conditions. Next the data character itself should be read, and after testing RR1 for errors, stored in a buffer. If errors did occur, retransmission can be requested. Note that RR1 *latches* the error bits, so that unless they are reset, they will ... be there on the next read. An output write to WR0 with D5–D3 = 110 accomplishes this. Because the error bits are latched, it may be desirable to read a *block* of characters and then test for an error. This technique would be more appropriate when interrupts are used, as it would allow the processor maximum time between character reads.

Example 9.14

Assuming that the initialization routine in Fig. 9.27(b) has been executed, write a subroutine that polls the Z-80 SIO in Fig. 9.23 and returns with the received character in register A. If an error has occurred (parity, overrun, or framing error), return with the Z flag reset and all error bits reset. Assume that the modem control lines and the break character are not used.

Solution. The program is shown in Fig. 9.29 and follows the receiver flowchart in Fig. 9.28(a).

```
        ;Z-80 SIO ASYNCHRONOUS RECEIVER SUBROUTINE
        ;
        ;THIS PROGRAM READS ONE CHARACTER FROM CHANNEL B OF
        ;THE Z-80 SIO INTERFACE SHOWN IN FIG. 9-23.  POLLING
        ;IS USED AND THE CHARACTER READ IS RETURNED IN
        ;REGISTER A.  IF AN ERROR OCCURS THE SUBROUTINE
        ;RETURNS WITH THE Z FLAG RESET.
        ;
        LD      C,0F3H          ;CHANNEL B CONTROL PORT ADDR
        LD      A,00000000B     ;POINT WR0 AT RR0
        OUT     (C),A           ;PROGRAM WR0
POLL    IN      A,(C)           ;READ RR0 - CH. B STATUS PORT
        BIT     0,A             ;TEST RECEIVER READY FLAG
        JR      Z,POLL          ;WAIT FOR A CHARACTER
        ;
        ;GOT A CHARACTER - STORE IT AND TEST FOR ERRORS
        ;
        LD      A,00000001B     ;POINT WR0 AT RR1
        OUT     (C),A           ;PROGRAM WR0
        IN      A,(C)           ;ERROR STATUS TO A
        AND     70H             ;TEST BITS 4,5,6
        IN      A,(0F2H)        ;CHARACTER TO A
        RET     Z               ;NO ERRORS IF ZERO
        ;
        ;RESET ERROR FLAGS FOR NEXT READ
        ;
        LD      B,00110000B     ;RESET ERROR BITS
        OUT     (C),B           ;PROGRAM WR0
        RET                     ;RETURN WITH Z FLAG RESET
```

Figure 9.29 Receiver polling program for Ex. 9.14.

Note: See Prob. 9.24 for the corresponding polled transmitter program.

Interrupts. The flowchart in Fig. 9.26 indicates that WR2 and WR1 must also be initialized when using the SIO with interrupts. WR2 holds the base interrupt vector for both channels A and B. The SIO can be programmed to output this vector for all interrupt conditions or to output up to eight variations of the vector corresponding to different status conditions. Bit D2 of WR1 for channel B controls the selection.

Assuming that this bit is set, Table 9.6 lists the conditions tested and the resulting modifications to the interrupt vector. The external/status change refers to the modem input control signals $\overline{DCD}$ and $\overline{CTS}$, and the $\overline{SYNC}$ pin programmed as an input.

TABLE 9.6. EFFECT ON THE INTERRUPT VECTOR DUE TO ENABLING THE STATUS EFFECTS VECTOR, BIT D2 OF WR1

	V_3	V_2	V_1	
Channel B	0	0	0	Transmit buffer empty
	0	0	1	External/status change
	0	1	0	Receive character available
	0	1	1	Special receive condition*
Channel A	1	0	0	Transmit buffer empty
	1	0	1	External/status change
	1	1	0	Receive character available
	1	1	1	Special receive condition*

* Special receive conditions: parity error, Rx overrun error, framing error, end of frame (SDLC).

Source: Courtesy of Zilog Corporation.

WR1 controls the interrupt mode. Using this register you control the source of the interrupts (receiver, transmitter, or special receive conditions) and whether to interrupt on all received characters or just the first. You can also program parity errors to alter the interrupt vector if desired.

Example 9.15

Modify the initialization codes given for the program in Ex. 9.13 to allow interrupt-driven I/O instead of polled I/O. Choose the codes such that all interrupts have their own address. The jump table base address is 1080H. Assume that the modem control signals are not used.

Solution. The seven codes specified in Ex. 9.13 need not be changed, but four additional codes will have to be added. These are shown in Fig. 9.30(a). Figure 9.30(b) shows the Z-80 program required.

Example 9.16

Specify jump table addresses for the interrupt sources initialized in Ex. 9.13.

Solution. Refer to Table 9.6 and use a base address of 1080H:

1080	Channel B transmitter buffer empty
1082	Not enabled
1084	Channel B receive character available
1086	Channel B parity, overrun, or framing error

Note: The four addresses for channel A are not enabled.

Using the SIO with the initialization program given in Fig. 9.30(b), transmitter, receiver, and error routines would have to be located at the addresses stored in the jump table shown in Ex. 9.16.

Block transfer mode. At first the thought of transferring data to or from the SIO in blocks does not seem logical. However, what is intended is to have the SIO facilitate a block transfer without direct CPU intervention. Certainly, the transfer of individual bytes will occur (relatively) slowly, but if implemented properly the CPU will be free to perform other tasks while the block is being transferred.

Block transfers are more commonly done in the synchronous mode but can be done asynchronously as well. Two methods are possible.

	Register	Binary Code Required	Hex	Explanation
8.	WR0	00 010 010	12	Point to WR2.
9.	WR2	10000000	80	Interrupt vector.
10.	WR0	00 010 001	11	Point to WR1.
11.	WR1	XXX 10 110	16	Interrupt on all rcvd char, status affects vector, trans interrupts enabled, external interrupts (the modem control signals) disabled.

(a)

```
;Z-80 SIO ASYNCHRONOUS INITIALIZATION ROUTINE
;              (WITH INTERRUPTS)
;
;THIS PROGRAM IS SIMILAR TO FIG. 9-27(B) BUT INCLUDES
;INITIALIZATION FOR A Z-80 MODE 2 INTERRUPT VECTOR.
;
        IM      2               ;MODE 2 INTERRUPTS
        LD      A,10H           ;HIGH ORDER JUMP TABLE ADDRESS
        LD      I,A             ;TO REGISTER I
;
;THE REMAINDER IS THE SAME AS FIG. 9-27(B)
;
        LD      C,0F3H          ;CHANNEL B CONTROL PORT
        LD      B,0BH           ;11 BYTES TO PROGRAM
        LD      HL,CODES        ;POINT HL AT CODE TABLE
        OTIR                    ;OUTPUT THE CODES
        EI                      ;ENABLE INTERRUPTS
;
;I/O CAN BEGIN HERE
;
CODES   DB      18H,14H,0C5H    ;THESE ARE THE INITIALIZATION
        DB      13H,C1H,15H     ;CODES
        DB      68H,12H,80H
        DB      11H,16H
```

(b)

Figure 9.30 (a) Additional control codes required when initializing the Z-80 SIO for interrupts; (b) initialization program for Ex. 9.15.

1. *DMA controlled*: A Z-80 DMA can be programmed for byte mode transfers and the $\overline{W/RDY}$ output of the SIO programmed as a RDY signal to synchronize the transfer. In this way the transfer of data to the SIO will occur very rapidly and allow the CPU considerable time for alternate processing between each byte.
2. *Z-80 block transfer instructions*: The Z-80 has several block transfer instructions which should be capable of keeping up with even the fastest SIO baud rates. The SIO $\overline{W/RDY}$ output can be programmed to request *WAIT* states automatically and thereby synchronize the SIO and Z-80 CPU. Of course, this technique does not allow the processor to perform other tasks simultaneously, but does eliminate the need for polling and provides the highest transfer rate possible without going to the Z-80 DMA.

Control of the $\overline{W/RDY}$ pin is via bits 5–7 of WR1, as summarized in Fig. 9.31. Regardless of the block transfer technique, the SIO would normally be programmed to interrupt on the first character received, after which the block would be transferred until completion.

Using the Z-80 SIO in the Synchronous Mode. As mentioned earlier in this chapter, synchronous communications involves more than deleting the start and stop bits of asynchronous serial. The receiving and transmitting stations must strictly adhere to a *protocol* governing the form of the data transfer.

The Z-80 SIO supports four such protocols:

1. Monosync
2. Bisync
3. External sync
4. SDLC

The first three are character-oriented, which means that the data field is made up of fixed-length characters (8 bits, for example). Figure 9.7 illustrated one frame

If $D_7 = 0$	
And $D_6 = 1$	And $D_6 = 0$
$\overline{READY}$ is High	$\overline{WAIT}$ is floating

If $D_7 = 1$	
And $D_5 = 0$	And $D_5 = 1$
$\overline{READY}$ Is High when transmit buffer is full.	$\overline{READY}$ Is High when receive buffer is empty.
$\overline{WAIT}$ Is Low when transmit buffer is full and an SIO data port is selected.	$\overline{WAIT}$ Is Low when receive buffer is empty and an SIO data port is selected.
$\overline{READY}$ Is Low when transmit buffer is empty.	$\overline{READY}$ Is Low when receive buffer is full.
$\overline{WAIT}$ Is floating when transmit buffer is empty.	$\overline{WAIT}$ Is floating when receive buffer is full.

Figure 9.31 The W/RDY output is controlled by bits 5–7 of WR1. (Courtesy of Zilog, Inc.)

ภาคผนวก ข.

Chapter 7

INTRODUCTION

This chapter will show you how to use the 8253 programmable timer chip with the Z80 microprocessor. We will begin by examining and describing this timer chip. We will then interface it with the Z80 microprocessor. Finally, we will write software that allows for its use in a variety of applications.

7-1: Block Diagram of the 8253 Programmable Timer

Let's begin by examining the block diagram in Figure 7.1 and exploring the internal registers and operating modes of this device. The diagram shows that the timer has three independent, programmable counters—and that they are all identical. In this chapter we will learn how to apply and use each counter.

Also shown in Figure 7.1 is the *data bus buffer.* This block contains the logic that buffers the data bus to and from the microprocessor, to the 8253 internal registers. In addition, there is the block labeled *read/write logic,* which controls the reading and writing of the counter registers. The final block—the *control word register*—contains the programmed information that is sent to the device from the system microprocessor. In effect this register defines how the chip logically operates. Figure 7.2 shows a pin configuration for the 8253 device.

7-2: The Three Counter Lines: Clock, Gate, and Out

Each counter in the block diagram in Figure 7.1 has three logical lines connected to it. Two of these lines, clock and gate, are inputs. The third, labeled out, is an output. The function of these lines changes and depends

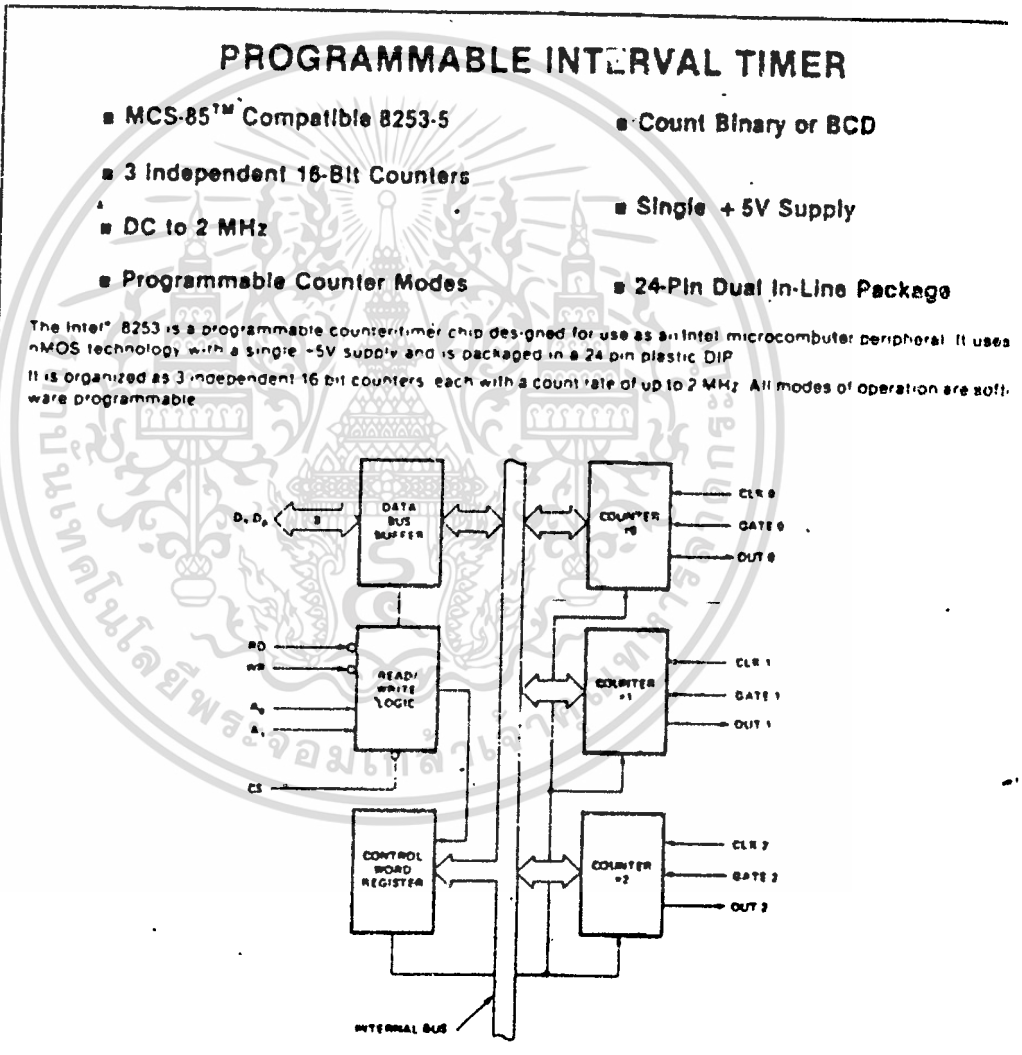


Figure 7.1: A block diagram of an 8253 programmable interval timer.

on how the device is initialized or programmed. Here is a general definition of the lines:

Clock This input is the clock input for the counter. The counter is 16 bits. The maximum clock frequency is 1/380 nanoseconds or 2.6 megahertz. The minimum clock frequency is DC or static operation.

Gate This input can act as a gate for the clock input line, or it can act as a start pulse, depending on the programmed mode of the counter.

Out This single output line is the signal that is the final programmed output of the device. Actual operation of the out line depends on how the device has been programmed.

7-3: 8253 Internal Registers

A list of the internal registers of the 8253 device appears in Figure 7.3. Let's first examine and discuss the *mode word register*. This register defines the overall operation of the device. Since each of the three counters is fully independent, each one can be programmed by outputting the correct data to the mode word register. We will show how this can be accomplished as our discussion proceeds. Let's first define the four internal registers shown in Figure 7.3.

Control Word Register This internal register is used to write information to, prior to using the device. This register is addressed when A0 and A1 inputs are logical 1's. The data in this register controls the operating mode and the selection of either binary or BCD (binary coded decimal)

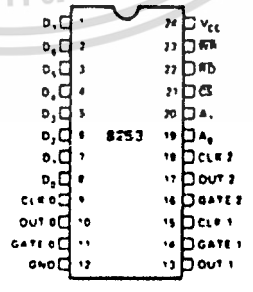


Figure 7.2: Pinout of the 8253 programmable interval timer.

counting format. This register can only be written to. The programmer cannot read information from this register.

Counter #0, #1, #2 Each counter is identical, and each consists of a 16-bit, pre-settable, down counter. Each is fully independent and can be made to count in BCD or binary. Contents of the counters can be easily read by the microprocessor. When the counter is read, the data within the counter is not disturbed. This allows the system to monitor the counter's value at any time, without disrupting the overall function of the device.

7-4: Connecting the 8253 to the Z80

Before we learn to program the 8253, let's learn how to connect it with the Z80 microprocessor.

The 8253 may be thought of as four separate I/O ports. The main selection of the I/O ports is accomplished with the $\overline{CS}$ input. When this input is logical 0, the 8253 is selected for communication with the Z80.

Address lines A0 and A1 determine which I/O ports are communicated with during the input or output cycle. This I/O architecture can be referred to as device port I/O. The device can be thought of as the 8253 and the ports associated with the device are the internal registers.

	RD	WR	A0	A1	
COUNTER 0	1	0	0	0	LOAD COUNTER 0
	0	1	0	0	READ COUNTER 0
COUNTER 1	1	0	0	1	LOAD COUNTER 1
	0	1	0	1	READ COUNTER 1
COUNTER 2	1	0	1	0	LOAD COUNTER 2
	0	1	1	0	READ COUNTER 2
MODE WORD OR CONTROL WORD	1	0	1	1	WRITE MODE WORD
	0	1	1	1	NO-OPERATION

Figure 7.3: A list of the internal 8253 registers that will program the internal counters of the device.

Decoding of the 8253 $\overline{CS}$ input is accomplished with the upper 6 bits of the lower byte address lines, namely A7–A2. As an example, let's map the device into the I/O architecture of a typical system. We will assume that the 8253 is mapped into the I/O space 30 to 33. This means that I/O ports 30H, 31H, 32H, and 33H are all associated with the 8253 (see Figure 7.4).

The $\overline{RD}$ and $\overline{WR}$ inputs to the 8253 are connected directly to the $\overline{IOR}$ and $\overline{IOW}$ system control lines. This setup is identical to the one for the 8255 device that we saw in Chapter 6. Figure 7.5 shows this connection.

Data lines on the 8253 can be connected directly to the Z80 data bus lines. (We are assuming that no data buffers are required for this application.) Figure 7.6 shows a complete connection of the 8253 device to the Z80, using

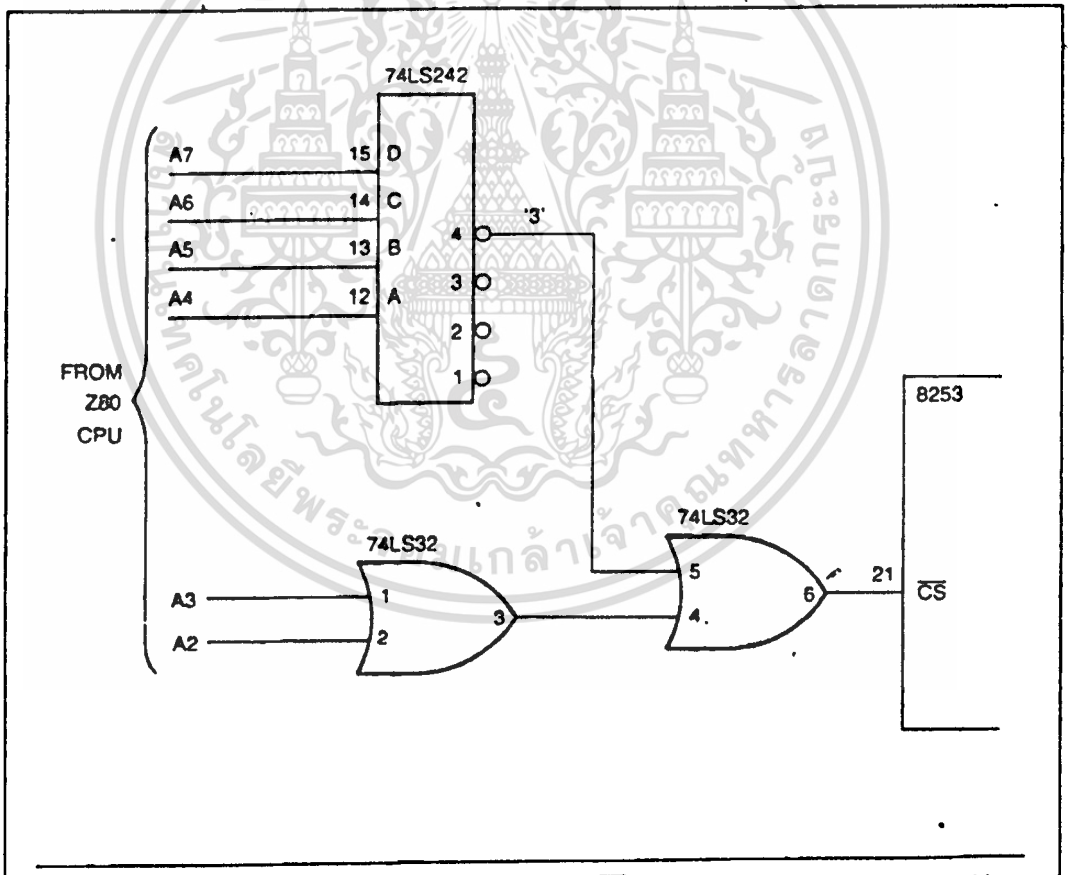


Figure 7.4: A schematic diagram showing how the $\overline{CS}$ input can be decoded from address lines A7–A2.

a non-buffered data scheme, while Figure 7.7 shows a complete connection, using a buffered data bus scheme. The data bus buffer used is the 74LS245. In Figure 7.7, whenever the 8253 is selected and the $\overline{RD}$ input is a logical 0, the 74LS245 is set to place the data from the 8253 onto the Z80 system data bus. At all other times, the 74LS245 places the data from the Z80 system data bus onto the 8255 data input pins.

Using the circuit connections shown in Figures 7.6 and 7.7, it is possible to reliably communicate between the Z80 and the 8253 device. The task now becomes one of programming the device so that it will perform in a way that your system application demands.

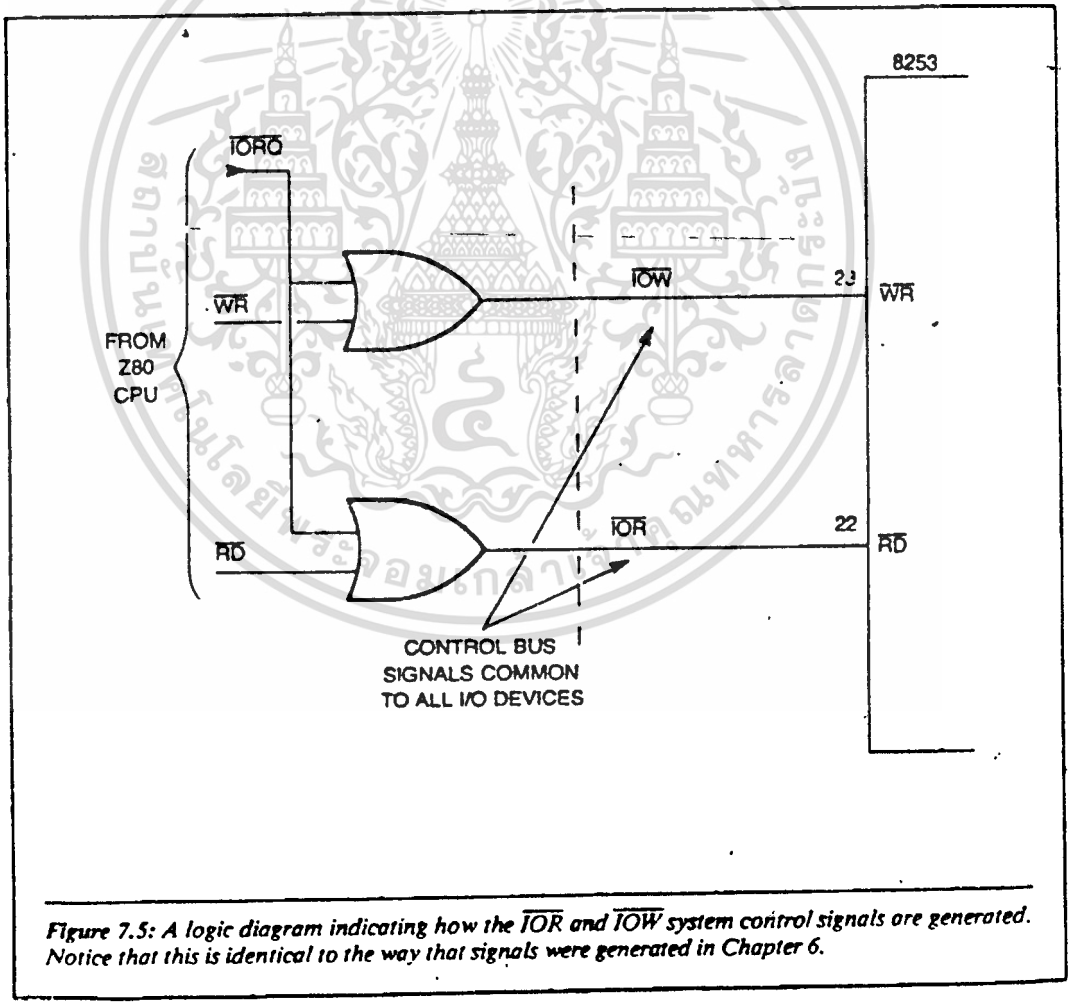


Figure 7.5: A logic diagram indicating how the $\overline{IOR}$ and $\overline{IOW}$ system control signals are generated. Notice that this is identical to the way that signals were generated in Chapter 6.

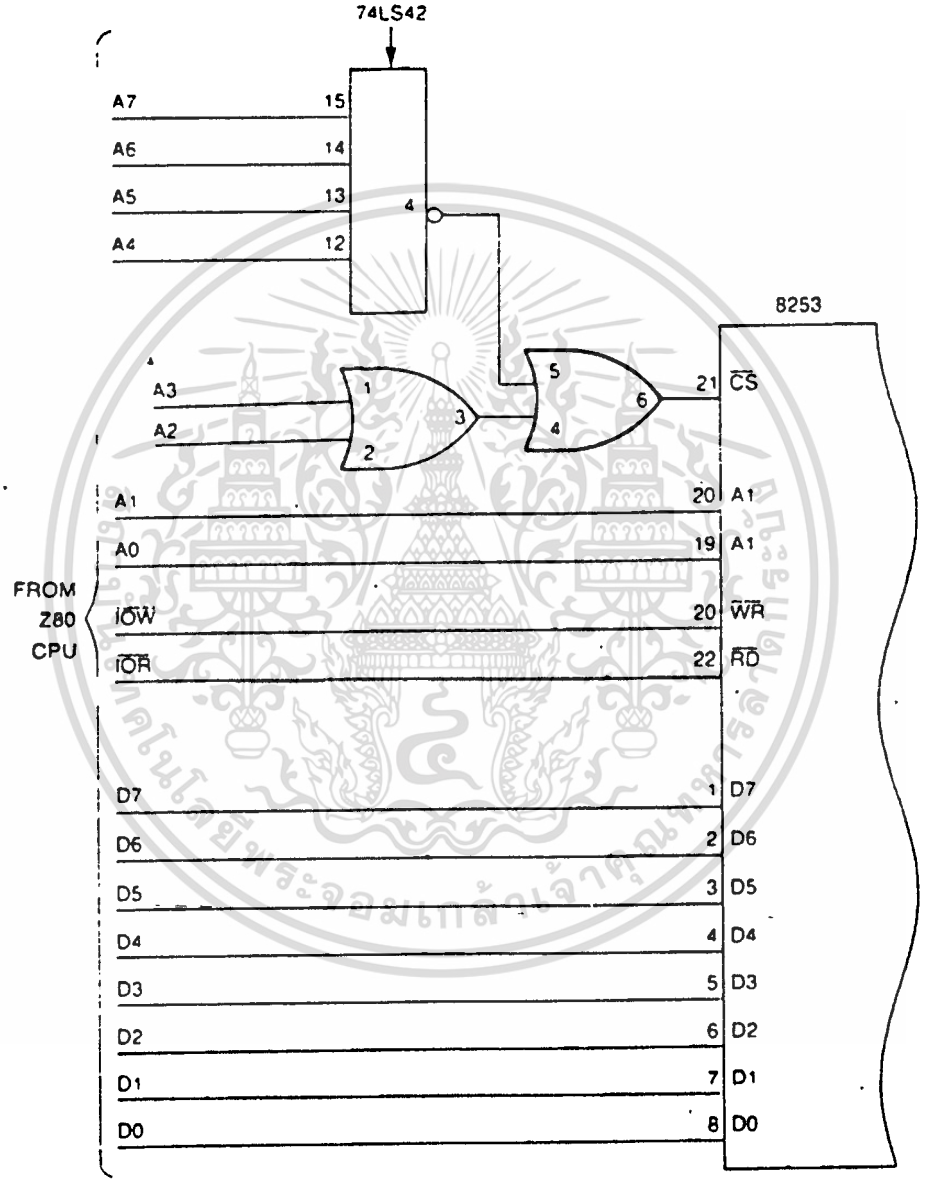


Figure 7.6: A complete connection between the 8253 and the Z80 system busses. No data buffering is used in this figure.

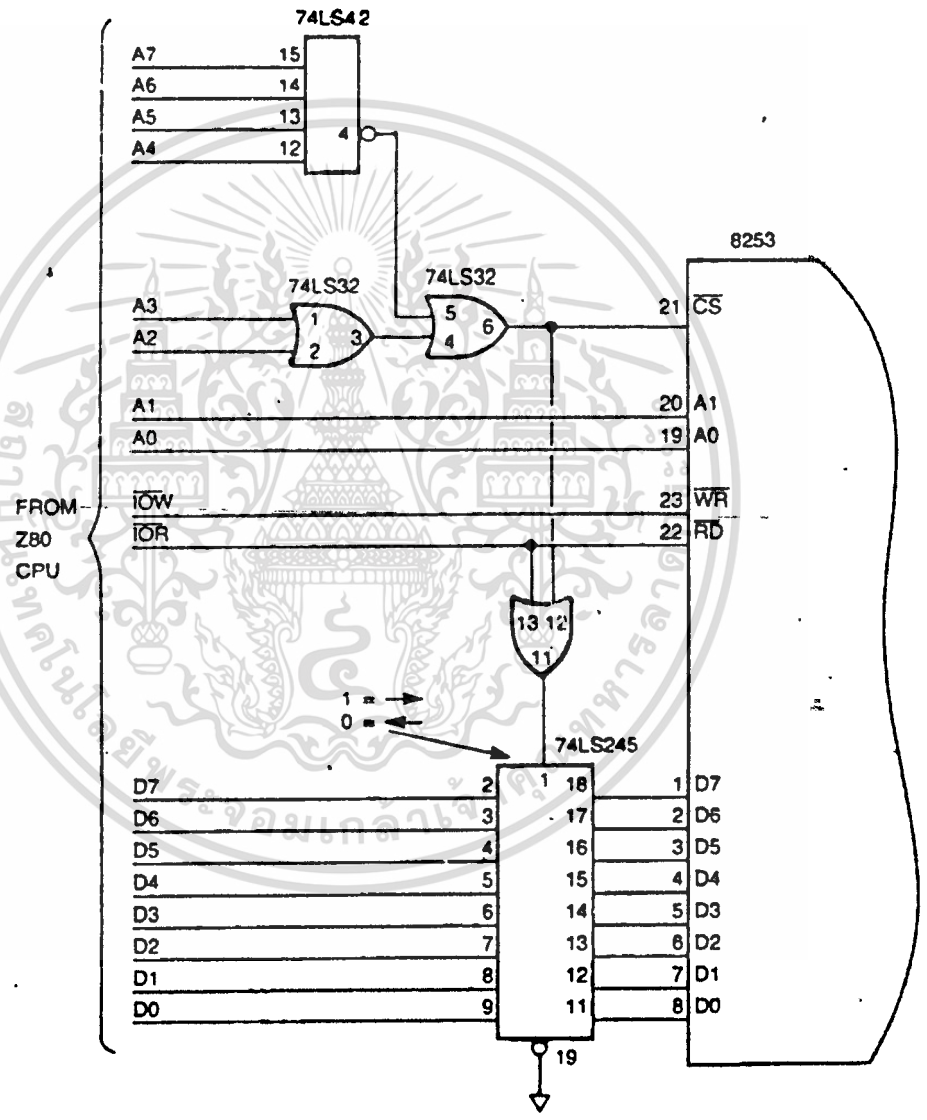


Figure 7.7: Connection of the Z80 and 8253 using data buffers.

7-5: Programming the Device (Control Word Format)

All of the operating modes for the counters are selected by writing bytes to the control register. Figure 7.8 shows the control word format. The address for the control word is A0 = 1 and A1 = 1. In this system application the control word address is 33H.

In Figure 7.8, bits D7 and D6 are labeled SC1 and SC0. These bits select the counter to be programmed. Before any counter can be programmed, it is necessary to define, using the control bits D7 and D6, which counter is being set up. It should be noted that once a counter is set up, it will remain that way until it is changed by another control word. The bits D7 and D6 are defined as follows:

D7	D6	COUNTER SELECT
0	0	0
0	1	1
1	0	2
1	1	illegal value

Bits D5 and D4 of the control word in Figure 7.8 are defined as the read/load mode for the register that is selected by bits D7 and D6. Bits D5 and D4 define how the particular counter is to have data read from or written to it by the microprocessor. These bits are defined as:

D5	D4	R/L DEFINITION
0	0	Counter value is latched. This means that the selected counter has its contents transferred into a temporary latch, which can then be read by the CPU.
0	1	Read/load least-significant byte only.
1	0	Read/load most-significant byte only.
1	1	Read/load least-significant byte first, then most-significant byte.

CONTROL BYTE D7–D0

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

Figure 7.8: Bit definition of the counter control register.

What do these bits tell the device to do? The first value, 00H, is the *counter latch mode*. It is useful for taking counter readings during the counter operation. When this mode is specified, the counter value is latched into an internal register at the time of the I/O write operation to the control register. When a read of the counter occurs, it is this latched value that is read.

If the latch mode is not used, then it is possible that the data read back may be in the process of changing while the read is occurring. (See Figure 7.9.) This could result in erroneous data being input by the CPU. To read the counter value while the counter is still in the process of counting, one must first issue a latch control word, and then issue another control word that indicates the order of the bytes to be read.

An alternative method of obtaining a stable count from the timer chip is to externally inhibit counting while the register is being read. This technique is shown in Figure 7.10. Each technique has certain disadvantages. The latching method may give the microprocessor a reading that is "old" by several cycles, depending on the speed of the count and which byte of the counter is being read. The external inhibiting function requires additional hardware. In addition, it may change the overall system operation. It is up to you to determine the best way to operate the device in a specific application.

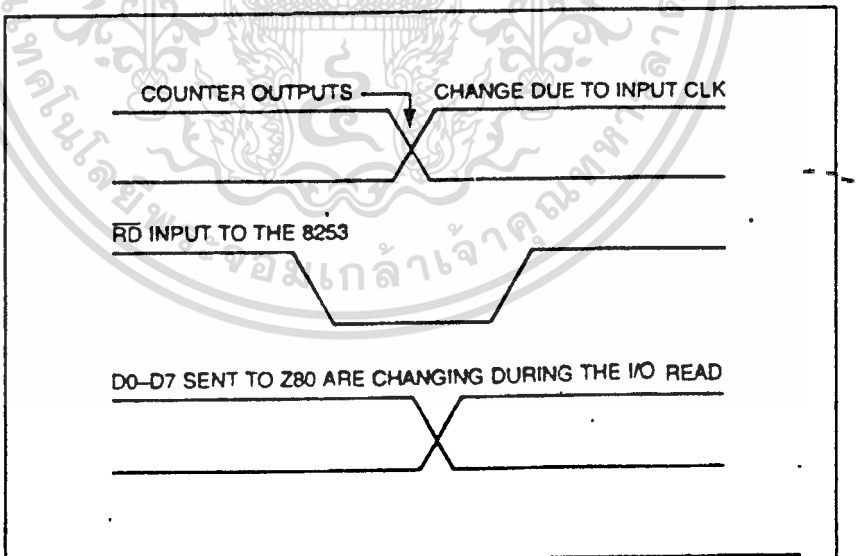


Figure 7.9: A timing diagram showing how the counter outputs may be changing while the microprocessor is electrically reading the 8253 device. This can be avoided by latching the counter output count prior to reading the counter value.

The next three bits of the control word in Figure 7.8 are D3, D2, and D1. These bits determine the basic mode of operation for the counter. We will now describe the mode and then follow with examples showing how to use the counter in each of the five modes. Here are the mode descriptions:

D3	D2	D1	MODE VALUE
0	0	0	mode 0: interrupt on terminal count
0	0	1	mode 1: programmable one-shot
x	1	0	mode 2: rate generator
x	1	1	mode 3: square wave generator
1	0	0	mode 4: software triggered strobe
1	0	1	mode 5: hardware triggered strobe

The final bit of the control register D0 determines how the register will count—that is, if it will count in BCD or Binary. If D0 is a logical 1, the count will be in BCD; if it is a logical 0, the count will be in binary. The maximum values for the count in each count mode are 2^{16} in binary, and 10^4 in BCD.

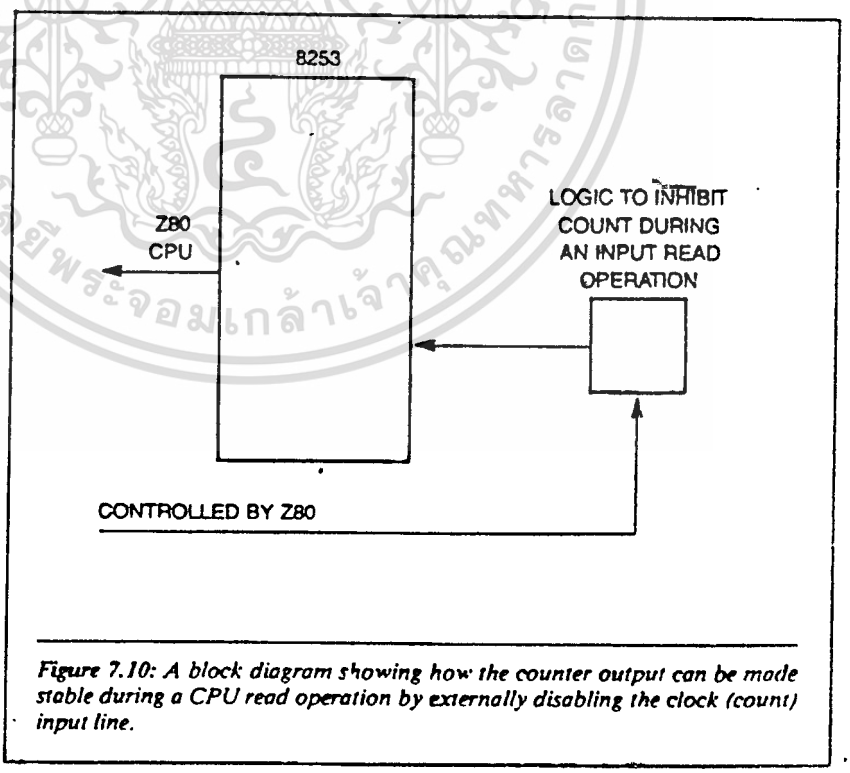


Figure 7.10: A block diagram showing how the counter output can be made stable during a CPU read operation by externally disabling the clock (count) input line.

```
;
;  WE ASSUME Z80 USING MODE 1 INTERRUPTS
;
;
;  THE COUNT HAS REACHED 125 AT THIS TIME WE
;  RELOAD THE COUNTER 0
;
0038    3E25            LD A,25H
003A    D330            OUT (30H),A     ;LOAD LSB
003C    3E01            LD A,01H
003E    D330            OUT (30H),A     ;LOAD MSB
;
;  NEW COUNT DOWN HAS STARTED
;
0040    FB              EI
0041    C9              RET
;
; PART OF THE INTERRUPT ROUTINE WOULD BE TO
; PERFORM SOME ACTION DUE TO THE INTERRUPT HAVING
; OCCURRED. IN THIS EXAMPLE WE SIMPLY RESET THE
; COUNTER AND START OVER.
;
                        END
```

Figure 7.13: A Z80 interrupt service routine using mode 1 interrupts. The 8253 is operating in mode 0.

7-7: Mode 1: Programmable One-Shot

In mode 1, the 8253 can be made to give an output pulse that is an integer number of clock pulses. The one-shot is triggered on the rising edge of the gate input. If the trigger occurs during the pulse output, the device will again be retriggered. This is shown in Figure 7.14.

Let's assume that the input clock frequency is one megahertz. We will program the 8253 for an output pulse of exactly 75 microseconds. The Z80 program to perform this setup is shown in Figure 7.15. In this figure we have assumed that counter 1 is being used as the programmable one-shot.

7-8: Mode 2: Rate Generator

In mode 2, the 8253 becomes a "divide by n" counter. The output pin of the counter goes low for one input clock period. The time between the output low-going pulses is dependent on the present count in the counter register. See Figure 7.16. If the count is loaded between output pulses, the present period will not be affected. A new period will occur during the next count sequence.

As an example, suppose that we wish to provide an output frequency on

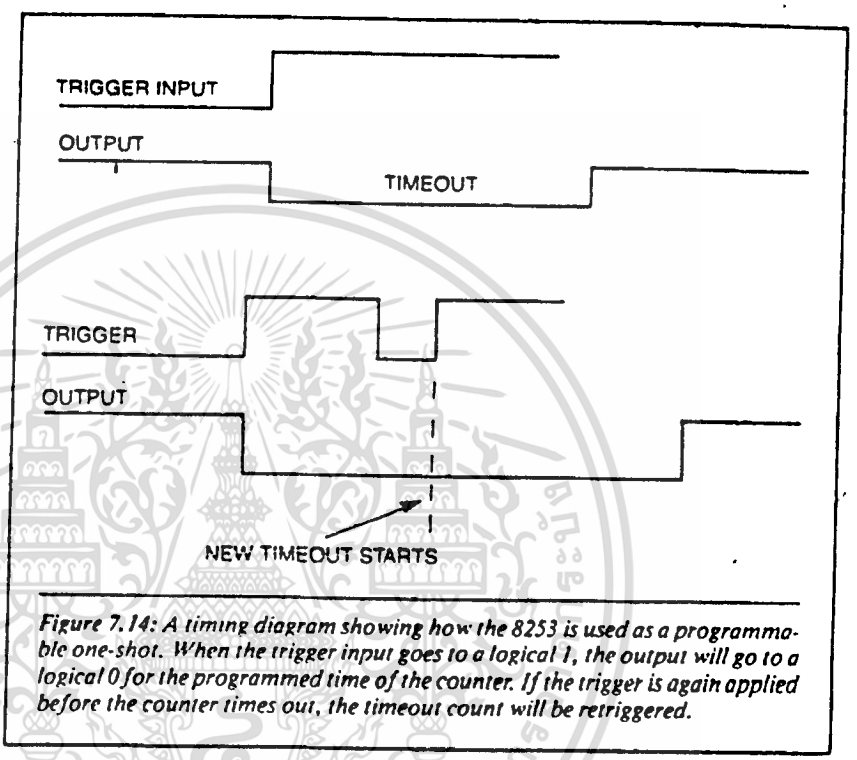


Figure 7.14: A timing diagram showing how the 8253 is used as a programmable one-shot. When the trigger input goes to a logical 1, the output will go to a logical 0 for the programmed time of the counter. If the trigger is again applied before the counter times out, the timeout count will be retriggered.

```
;
;
;
0000    3E72            LD A, 01110010B
0002    D333            OUT (33H),A         ;OUTPUT CONTROL WORD TO 8253
;
; CTR 1, RL = 3, M=1, BINARY COUNT
;
0004    3E4B            LD A, 75            ;DECIMAL 75
0006    D331            OUT (31H),A         ;OUTPUT TO CTR1 LSB
0008    3E00            LD A, 00
000A    D331            OUT (31H),A         ;OUTPUT TO CTR1 MSB
;
;
; THE DEVICE IS NOW SET UP WAITING FOR A TRIGGER
; INPUT TO THE COUNTER
;
                        END
```

Figure 7.15: A Z80 program for using the 8253 as a programmable one-shot.

counter 2 that is equal to 638 hertz. This would be a period of 1/638 = 1.567 milliseconds or 1,567 microseconds. If an input clock of 1 megahertz were applied to the clock input of counter 2, then the counter would need to be programmed to 1,567. This could be done in BCD or decimal. Figure 7.17 shows an example of a Z80 program that would program the 8253 for an operation such as this.

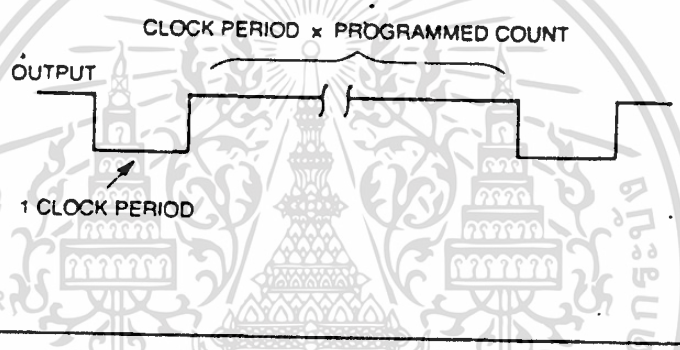


Figure 7.16: A timing diagram showing how the output of the 8253 would appear when used in mode 2 as a rate generator. The output will go to a logical 0 for one input clock period. The time between pulses is determined by the count programmed in the device.

```
;
;
0000   3EB5          LD A,10110101B
0002   D333          OUT (33H),A          ;OUTPUT CONTROL WORD
0004   3E67          LD A,67H
0006   D332          OUT (32H),A          ;CTR 2, LSB IN BCD
0008   3E15          LD A,15H
000A   D332          OUT (32H),A          ;CTR 2, MSB IN BCD
;
;
;  THE DEVICE IS NOW INITIALIZED AND THE OUTPUT FREQUENCY
;  WILL BE 638 HERTZ, WITH AN INPUT FREQUENCY OF 1 MHZ.
;
                     END
```

Figure 7.17: Z80 program for using the 8253 device as a rate generator. The output frequency will be 683 hertz. The input clock frequency is 1 megahertz.

7-9: Mode 3: Square Wave Generator

Mode 3 is similar to mode 2 except that the output will be high for half the period and low for half. If the count is odd, the output will be high for $(n + 1)/2$ and low for $(n - 1)/2$ counts. Figure 7.18 shows a Z80 program to setup counter 0 for a square wave frequency of 10k hertz, assuming that the input clock frequency is equal to 1 megahertz.

7-10: Mode 4: Software Triggered Strobe

In this mode the programmer can set up the counter to give an output timeout that will start when the count register is loaded. On the terminal count, when the counter equals zero, the output will go low for one clock period and then it will go high again. This is shown in Figure 7.19. When the mode is set, the output will go high.

7-11: An Example

As an example, let's program the device to provide a 100 millisecond, software-triggered delay at counter output 2. We will assume that the input frequency equals one megahertz, which requires an input clock count of 10^5. In addition, we wish to perform the count in BCD. The maximum BCD count in any register is equal to 10^4. Therefore, we will use two counters. The first will divide the frequency down to 1 kilohertz. The second will

```
;
;
;
0000   3E36            LD A,00110110B
0002   D333            OUT (33H),A        ;SETUP CTR0 CONTROL WORD
0004   3E64            LD A, 100
0006   D330            OUT (30H),A        ;CTR0 LSB IN BINARY
0008   3E00            LD A, 00H
000A   D330            OUT (30H),A        ;CTR0 MSB IN BINARY
;
;
; THE DEVICE IS NOW SETUP AND RUNNING WITH AN OUTPUT
; FREQUENCY OF 10KHZ WITH AN INPUT FREQUENCY OF 1MHZ.
;
                       END
```

Figure 7.18: A Z80 program for using the 8253 device as a square wave generator. The output frequency is 10 kilohertz. The input frequency clock is 1 megahertz.

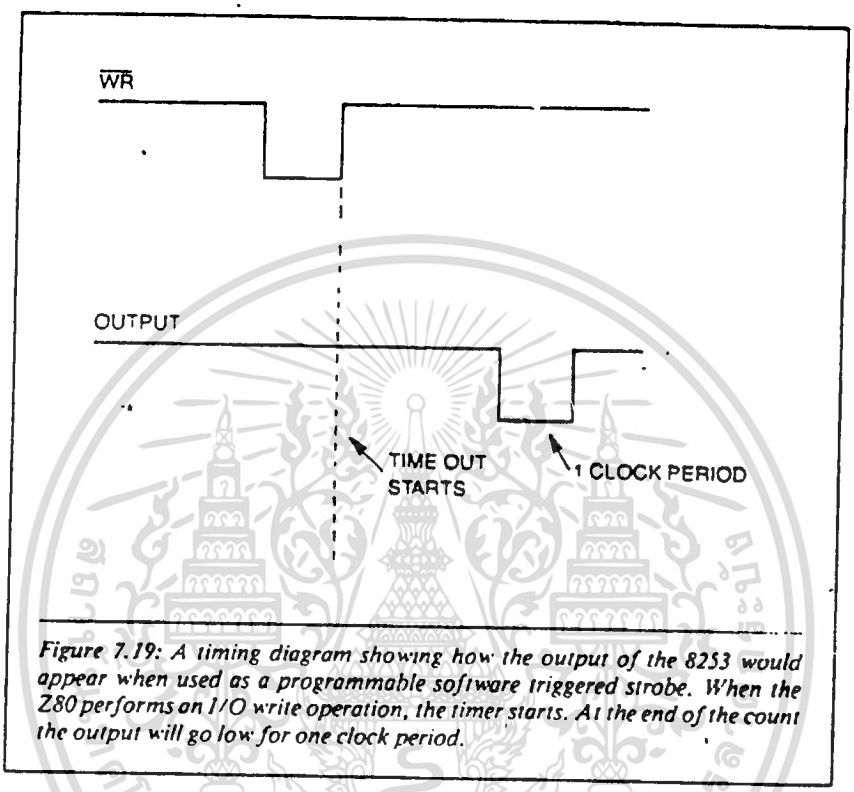


Figure 7.19: A timing diagram showing how the output of the 8253 would appear when used as a programmable software triggered strobe. When the Z80 performs an I/O write operation, the timer starts. At the end of the count the output will go low for one clock period.

provide the software-triggered strobe. This is shown in Figure 7.20. Figure 7.21 presents a Z80 program that uses the 8253 in the manner described in this example.

7-12: Mode 5: Hardware Triggered Strobe

With the counter in this mode, the rising edge of the trigger input will start the counter counting. The output will go low for one clock at the terminal count. The device is retriggerable, thus meaning that if the trigger input is taken low and then high during a count sequence, the sequence will start over (see Figure 7.22). Figure 7.23 shows a Z80 program to initialize the 8253 counter 1 to be used as a hardware triggered strobe.

7-13: Uses of the Gate Input Pin

Each mode of operation for the counter chip has a different use for the gate input pin. The table shown in Figure 7.24 summarizes the operation of this particular input pin on the device.

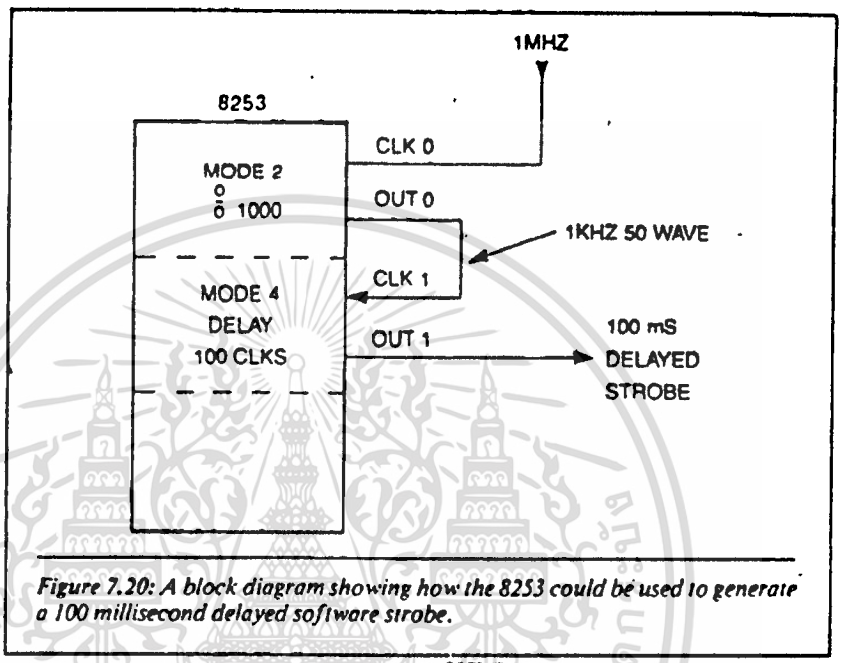


Figure 7.20: A block diagram showing how the 8253 could be used to generate a 100 millisecond delayed software strobe.

```
0000    3E35            LD A,00110101B
0002    D333            OUT (33H),A         ;SETUP COUNTER 0 CONTROL WORD
0004    3E00            LD A,00H
0006    D330            OUT (30H),A         ;LOAD CTR0 LSB
0008    3E10            LD A,10H
000A    D330            OUT (30H),A         ;LOAD CTR0 MSB
;
;
;   COUNTER 0 IS SETUP WITH AN OUTPUT FREQUENCY OF 1KHZ
;   SQUARE WAVE WITH AN INPUT FREQUENCY OF 1MHZ. THIS
;   OUTPUT IS CONNECTED TO CTR1 CLOCK INPUT.
;
000C    3E79            LD A,01111001B
000E    D333            OUT (33H),A         ;CTR1 CONTROL WORD
0010    3E00            LD A,00H
0012    D331            OUT (31H),A         ;CTR1 LSB
0014    3E01            LD A,01H
0016    D331            OUT (31H),A         ;CTR1 MSB
;
;   THE LAST OUT TO CTR1 NEED NOT BE DONE UNTIL THE
;   TIME OUT IS WANTED
;
                        END
```

Figure 7.21: A Z80 program to setup the 8253 to operate a 100 millisecond software strobe. This program makes use of the block diagram given in Figure 7.20.

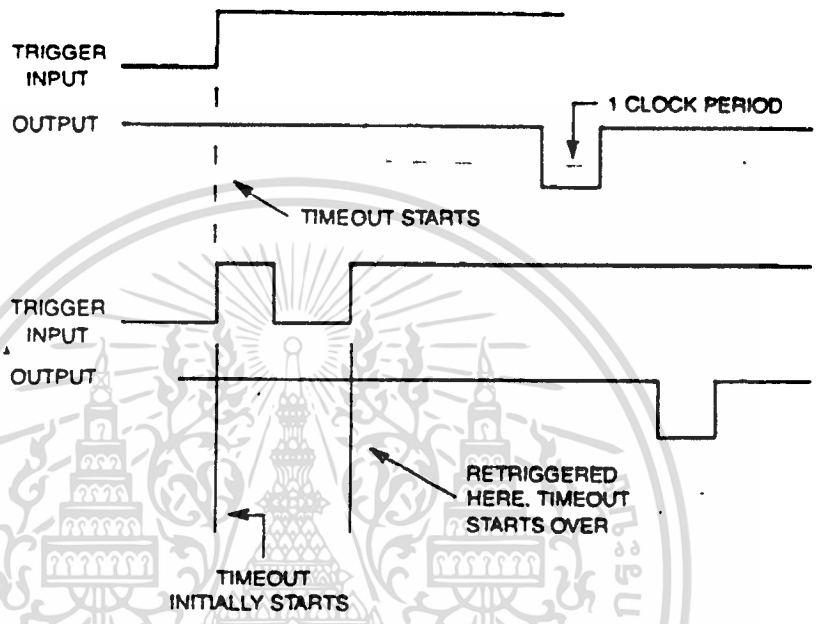
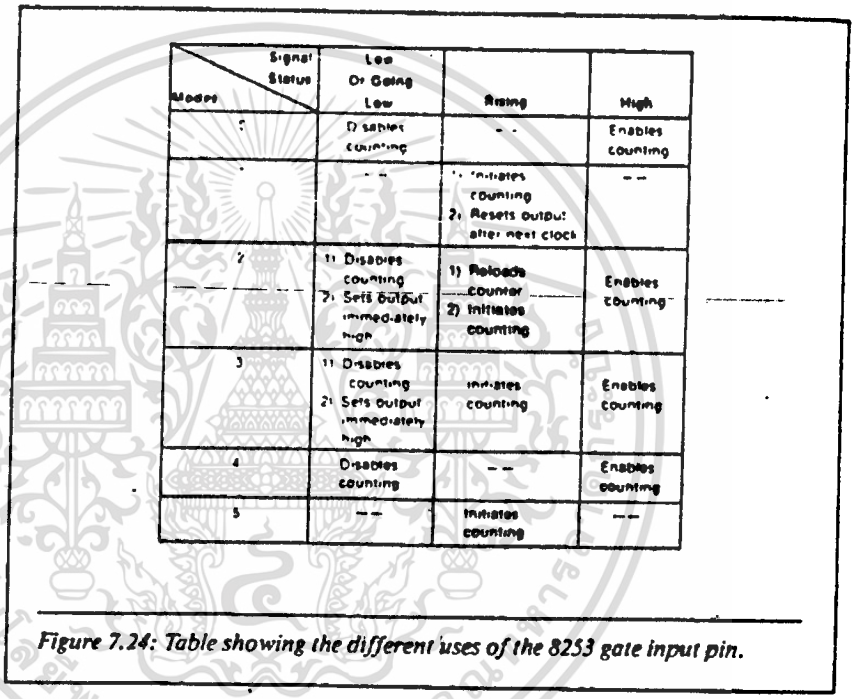


Figure 7.22: A timing diagram showing how the 8253 operates as a hardware triggered strobe. When the external trigger input goes to a logical 1, the timer will start to time out. If the external trigger occurs again, prior to the time completing a full timeout, the timer will retrigger.

```
;
;
;
;
0000   3E7A            LD A,01111010B
0002   D333            OUT (33H),A        ;CTR1 CONTROL WORD
0004   3E55            LD A,85
0006   D331            OUT (31H),A        ;OUTPUT 85 BINARY TO CTR1 LS
0008   3E00            LD A, 00H
000A   D331            OUT (31H),A        ;OUTPUT 00 TO CTR1 MSB
;
;  THE CTR IS NOW SET UP WAITING FOR AN EXTERNAL
;  TRIGGER INPUT ON TRG1.
;
                       END.
```

Figure 7.23: A Z80 program to setup and use the 8253 in the hardware-triggered strobe mode as shown in the timing diagram in Figure 7.22.



Mode / Signal Status	Low Or Going Low	Rising	High
0	Disables counting	– –	Enables counting
1	– –	1) Initiates counting 2) Resets output after next clock	– –
2	1) Disables counting 2) Sets output immediately high	1) Reloads counter 2) Initiates counting	Enables counting
3	1) Disables counting 2) Sets output immediately high	Initiates counting	Enables counting
4	Disables counting	– –	Enables counting
5	– –	Initiates counting	– –

Figure 7.24: Table showing the different uses of the 8253 gate input pin.

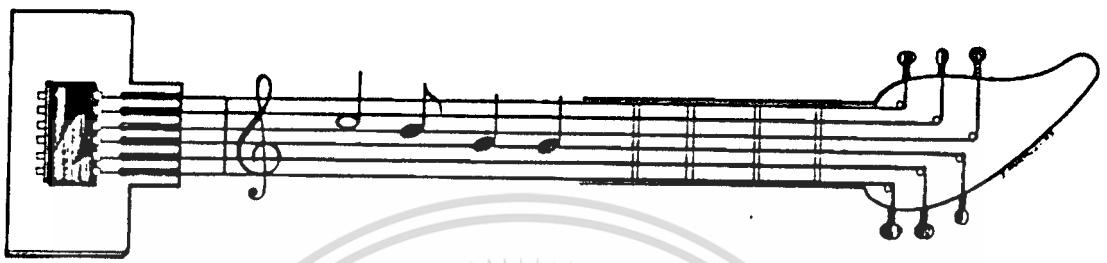
CHAPTER SUMMARY

In this chapter we have examined the important points of using the 8253 programmable timer with the Z80 CPU. We have presented a block diagram of the device and a discussion of each operating mode. The 8253 can be used to serve a majority of timing functions in many system applications that use the Z80 as a system controller. The examples in this chapter have shown that using this device is not a complex task.

ภาคผนวก ค.

MIDI – Musical Instrument Digital Interface

Richard Evers, Editor



The idea started slow, but rapidly picked up speed.

Throughout the ages, man has always strived for new and exiting ways in which to amuse himself. Man is the master amuser. One way in which amusement was found, and still is today, is with music. In the beginning, there is a hint that the first musical instruments were rocks, producing a somewhat pleasing sound when smashed together in whatever unison they could muster. Soon, wood was found to possess the enthralling capacity to produce various sounds when struck. As time passed, man developed the art of cutting and forming pieces of wood to produce other truly unique sounds. From this simple beginning, we now have every instrument from the bongo drum to the grand piano. Over thousands of years of patient evolution, man has developed the most beautiful art of all – music.

This unique beginning is a primer to enable you to gain easy entrance into todays world of digital music synthesis, a world far removed from what music has been in past. Although the music synthesizer is capable of cloning the sounds of all man made instruments, it also has the ability to allow or disallow the limitations imposed on the instrument by mans capabilities. With a statement sure to be written off as sheer personal judgement, I will state that the finest of todays music synthesizers are capable of mimicking the sounds of man made instruments to such an extent that the synthetic counterpart will be preferred. But, as with all new concepts, many people will strive to retain the older, more familiar yet less refined methods of past.

Today, in what is known as the age of the microprocessor, the true " State Of The Art " (what does that mean?) phase of our existence, analog systems still persist in great numbers. Although the digital movement has taken all forms of music reproduction by storm, ie. the compact disk, digital recorded record albums and, of course, music sythesis, analog is still with us. Time will be the sole test of our loyalty to a friend long past its prime. Achieving results that were once thought impossible using analog logic now appear common place due the advent of todays digital high technology. But, justly enough, within a few years travel, our current high tech will also be considered obsolete. The sad fate of time. Obsolescence.

To get on track, a few years ago, when digital synthesis was beginning to reach its apex, a very obvious road block materialized. There were no standards to meet, no industry standard that would allow various components to be easily connected together for use. The manufacturers were producing terrific products, but the buyer was restricted to one name brand for add ons. A restricting situation.

To solve this problem, a few key people in the music sythesis industry started thinking, talking, and attracting attention along this line. The idea started slow, but rapidly picked up speed. MIDI, The Musical Instrument Digital Interface, was the solution presented to combat this problem.

The MIDI system is a series of guidelines that manufacturers should meet in order to ensure the compatibility of their products with others. As with all set guidelines, MIDI is not perfect and does actually impose limitations at times. Without coming down too hard at first, let me explain the guidelines.

The system is based on the almost current 8 bit technology, mainly because the designers were trying to produce a system that would easily fit into the budget of most starving artists. They chose a universal cable connection of the 5 pin DIN plug, using only 3 of the 5 pins. Pin #2 is ground, with pin #'s 4 and 5 being used for a current loop.

" The interface operates at 31.25 kilobaud (±1%), asynchronous, with a start bit, 8 data bits (D0 to D7), and a stop bit. This makes a total of 10 bits for a period of 320 microseconds per serial byte. "

The above paragraph was lifted directly from the MIDI 1.0 Specifications, Document No. MIDI-1.0, Dated August 5, 1983.

With each MIDI equipped unit, you will notice either two or three connectors for MIDI use. There is the MIDI In, MIDI Out, and possibly, MIDI Through (American spelling Thru). MIDI In is the connector for data coming into the unit from other MIDI equipped units on line. This can include drum and rhythm machines, extra keyboards, electric guitar and drum interfaces, controllers and sequencers, or extra synthesizers.

A total of 16 channels, numbered 1 through 16, are allowed under MIDI but more than 16 instruments can be on line at any one time. By assigning duplicate channel numbers to units, or having some units respond to all messages (see Omni Mode), vast numbers of units can be serviced. One main limitation to this provision is the extended line length required, which produces added line loss and a greater time delay with transmitted and received data. By reading more than a few articles on this subject, one fact becomes apparent. When using more than a couple of MIDI units on line, noticeable delays appear in what is deemed simultaneous sound reproduction. But, I have recently read an outstanding interview with a person well qualified in the field. He stated that the time delay between units was not completely the fault of the MIDI implementation. In his opinion, the rate at which the units process the information provided leads to the noticeable time lag. Without greater experience in the subject, it is hard to take a side.

MIDI Out is the data being transmitted from the MIDI equipped unit. MIDI Through is a copy of the information currently being passed into the unit through MIDI In. This feature, if provided, gives the user the ability to slave units together. According to MIDI specs, the cable length between units cannot exceed 50 feet, using shielded, twisted cable with Pin #2 connected to the shielding at both ends.

The transmitted data is quite easy to identify. All transmitted information is 8 bits in length, with the Status bytes having the high bit set and data bytes having the high bit clear. As we will soon discuss, the Status byte has been provided to control the system. Within the Status class will be found all commands necessary to determine what the system will do, when it will do it, and how it should be done. The Data bytes are really the workers of the system. Not only do they carry information such as the note value to be played, but they also labour under the rule of the Status bytes to ensure that all commands are understood. For the purpose of this article, the eight bit byte is arranged from bits 7 to 0, with bit 7 acting as the high bit.

The charts below have been prepared to show all the possible byte values encountered while working with the MIDI system. Complete explanations will follow.

Table Of Notes Corresponding To Data Values
Expressed In Hex

	Notes											
Octave	C	C#	D	D#	E	F	F#	G	G#	A	A#	B
–	$00	$01	$02	$03	$04	$05	$06	$07	$08	$09	$0A	$0B
0	$0C	$0D	$0E	$0F	$10	$11	$12	$13	$14	$15	$16	$17
1	$18	$19	$1A	$1B	$1C	$1D	$1E	$1F	$20	$21	$22	$23
2	$24	$25	$26	$27	$28	$29	$2A	$2B	$2C	$2D	$2E	$2F
3	$30	$31	$32	$33	$34	$35	$36	$37	$38	$39	$3A	$3B
4	$3C	$3D	$3E	$3F	$40	$41	$42	$43	$44	$45	$46	$47
5	$48	$49	$4A	$4B	$4C	$4D	$4E	$4F	$50	$51	$52	$53
6	$54	$55	$56	$57	$58	$59	$5A	$5B	$5C	$5D	$5E	$5F
7	$60	$61	$62	$63	$64	$65	$66	$67	$68	$69	$6A	$6B
8	$6C	$6D	$6E	$6F	$70	$71	$72	$73	$74	$75	$76	$77
9	$78	$79	$7A	$7B	$7C	$7D	$7E	$7F				

Status Bytes With Messages

Bit Pattern	Decimal Value	Hex Value	Data Bytes	Status Byte Message
1000 xxxx	128 to 143	$80 to $8F	2	Note Off
1001 xxxx	144 to 159	$90 to $9F	2	Note On
1010 xxxx	160 to 175	$A0 to $AF	2	Polyphonic Key Pressure
				After Touch
1011 xxxx	176 to 191	$B0 to $BF	2	Control Change
1011 xxxx	176 to 191	$B0 to $BF	2	Select Channel Mode
1100 xxxx	192 to 207	$C0 to $CF	1	Program Change
1101 xxxx	208 to 223	$D0 to $DF	1	Channel Pressure
				After Touch
1110 xxxx	224 to 239	$E0 to $EF	2	Pitch Wheel Change
1111 0000	240	$F0	variable	System Exclusive
1111 0001	241	$F1	–	Unimplemented
1111 0010	242	$F2	2	Song Position Pointer
1111 0011	243	$F3	1	Song Select
1111 0100	244	$F4	–	Unimplemented
1111 0101	245	$F5	–	Unimplemented
1111 0110	246	$F6	0	Tune Request
1111 0111	247	$F7	0	End Of Exclusive
1111 1000	248	$F8	0	Timing Clock
1111 1001	249	$F9	–	Unimplemented
1111 1010	250	$FA	0	Start
1111 1011	251	$FB	0	Continue
1111 1100	252	$FC	0	Stop
1111 1101	253	$FD	–	Unimplemented
1111 1110	254	$FE	0	Active Sensing
1111 1111	255	$FF	0	System Reset

After that unrelenting barrage of MIDI information, I feel that it is only fitting to supply you with a bit more of a detailed description. To begin, the frequency data bytes, hex values $00 to $7F, are obviously the note values as expressed through almost 10 complete octaves. This is one drawback of the system. 7 bits of resolution is not sufficient to allow graduations in smaller increments than semitones. One recent speculation made regarding this point stated that future revisions of the system specifications should allow note increments in graduations of cents, 1/100's of a semitone. A possible but difficult to implement idea.

Status Messages, in contrast to the note values, require more than a little explanation before becoming coherent to any degree. Therefore, there are two different Status messages that can be encountered. The Channel Message and the System Message.

The Channel Message

A Channel message is one that states a message and a specific unit number to which the message is addressed. The lower 4 bits, bits 0-3, are used for this identification. In the charts shown above, plus the descriptions to follow, this has been signified by the xxxx in place of the lower nibble in each appropriate bit pattern. This 4 bit channel designation also explains the limitation of this system to 16 channels (decimal range 0-15).

There are two types of channel messages: Voice and Mode. Voice messages control each separate units voices, which are sent over the voice channels. The Mode message defines the instruments response to the Voice messages, sent over the instruments basic channel. The Mode message can control such things as Local Control On and Off, All Notes Off, Omni Mode On or Off, Mono Mode On/Poly Mode Off, or Poly Mode On/Mono Mode Off. See message Select Channel Mode for more information.

The System Message

System messages are ones that do not pertain to any one specific channel. Being the elite of the system entitles them to speak and except all to listen, immediately. Within the realm of the System message falls the Common, Real-Time, and Exclusive messages.

Common messages are directed to all units, regardless of channel number. The Common messages are comprised of Song Position Pointer, Song Select, Tune Request, and End Of Exclusive. Each Common message consists of a single byte, of which an in depth explanation will follow promptly.

Real-Time messages are also intended to be heard by all units in the system, at any time, even during the transmission of other data by a Status byte. A Real-Time message is one in which can be tested for and ignored by the units, or acted upon if the information is desired. Real-Time messages are comprised of the Timing Clock, Start, Continue, Stop, Active Sensing, plus System Reset. Explanations will follow.

There is only one Exclusive Message, of which carries one Status byte, plus any length of data bytes following. It is terminated by either the Common message End Of Exclusive (EOX), or any other Status byte. An Exclusive Message is one that has been incorporated to identify the manufacturer of each piece of equipment, plus allow the manufacturer to transmit whatever message they please. A form of personalized service, finally.

Status Byte Explanations

Message:**Note Off**
Pattern: 1000 xxxx

The Note Off Message, used in conjunction with the Note On message, determine when each note will start and stop for the channel affected. Following this byte are always two data bytes. The first byte determines which note within a specific octave is to be affected, as can be demonstrated in the note chart above. The second byte is the Note Off Velocity. This value is used in synthesizers capable of ADSR (attack, decay, sustain, release), to set the rate of release from the sustained level. In synthesizers not so enabled, the release is immediate.

Message:**Note On**
Pattern: 1001 xxxx

As specified above, the Note On command is used to determine which note is stuck on a specified channel. The Note On Velocity, the second data byte, has a unique use this time. It determines the loudness setting of the note value struck. In units capable of ADSR, this level of loudness will be attained within a user predetermined time limit. If the unit is incapable of ADSR, the level will be reached immediately. The maximum loudness setting, considering we are working with the lower 7 of 8 bits, is a decimal value of 127. A value of 0 would be the same as turning the Note Off. According to specs, a value of 1 is equivalent to "triple pianissimo", very quite. A value of 127 (maximum), is "triple forte", very loud. The middle of the scale is 64, which is somewhere between "mezzo-piano" and "mezzo-forte" - middle scale. If the unit is incapable of various velocity settings, a value of 64 is used and transmitted.

Message:**Polyphonic Key Pressure After Touch**
Pattern: 1010 xxxx

Polyphonic Key Pressure After Touch is really just what is implied. The new value of key pressure attained after touching it. If the value has been set before to reflect the key pressure exerted, and the user strikes the key once again, this new pressure must be reflected. This status byte has two data bytes following. The first byte is the note stuck, with the second value byte representing the pressure. The values of pressure range from 0, no pressure at all, to 127, bashed through the keyboard

Message:**Control Change**
Pattern: 1011 xxxx

Control Change is a status byte that is transmitted whenever a controller mechanism is adjusted. Controller mechanisms in this context refer to foot pedals, knobs (pots), modulation wheels, sliders, and switches. Two data bytes are used after the status message to reflect this change. The first byte indicates which controller was affected, while the second byte determines the new value attained. There are four ranges allowed for various controller mechanism used, as shown below.

Value Range	Controller Type
$00 to $3F	Continuous (ie. Foot Pedals, Knobs, Modulation Wheels, Sliders)
$40 to $5F	Switches
$60 to $7A	Presently Undefined
$7B to $7F	Channel Mode Messages (see Select Channel Mode following)

The first range, Continuous, often requires a more subtle range of graduations in comparison to say a switch. Either a switch is off or on, 0 or greater than 0. Continuous devices such as a slider can graduate across the scale in as large or small increments as desired, therefore a very subtle range is required. For this reason, the Control Change message allows for three data bytes if necessary. As stated before, the first data byte determines the controller affected, with the second data byte determining the value to set. This acts as the High Byte, or increment of 256 Lower Bytes. By now you know that the third byte is the Low Byte, thereby allowing a range of 14 bits, or 16,384 graduations. The Low Byte is not mandatory, therefore it can be left off if not required.

Message:**Select Channel Mode**
Pattern: 1011 xxxx

Select Channel Mode is a unique message that allows alterations to the way a MIDI unit will respond to, and transmit MIDI channel messages. Two data bytes are used to allow this transformation, with the first data byte having a limited range of $7A to $7F. This limitation is imposed due to the fact that two different messages share the same bit pattern (see Control Change above). The second data byte is used in conjunction with the first to achieve the result desired, as shown below.

Byte 1	Byte 2	Result
$7A	$00	Local Control Off
$7A	$7F	Local Control On
$7B	$00	All Notes Off
$7C	$00	Omni Mode Off, All Notes Off
$7D	$00	Omni Mode On, All Notes Off
$7E	$0x	Mono Mode On, Poly Mode Off, All Notes Off (x = number channels)
$7F	$00	Poly Mode On, Mono Mode Off, All Notes Off

The term Poly refers to Polyphonic sound reproduction, which is the ability for the unit to allow more than one note to be played simultaneously. Mono refers to Monophonic sound reproduction, which is reproduction of sound that only allows one note to be played during any one time period.

Omni is a term that is used to describe the ability for a unit to respond to all system messages, or only ones addressed to it's basic channel. When Omni is Off, the unit will only listen for messages addressed to itself. When Omni is On, it will listen and act upon every message coming over the bus.

Local Control is the ability for a unit to act through, or bypass it's own circuitry for the generation of sound. With Local Control On, the synthesizer will act like it normally does. With Local Control Off, the keyboard will still produce MIDI data as it is played, but the synthesizer will not produce any sound to compliment the data. This feature was incorporated to allow keyboards to control instruments other than the synthesizer attached.

Message:**Program Change**
Pattern: 1100 xxxx

Program Change is used with synthesizers that have banks of memory set aside for various user chosen sounds. Often times these sounds are assigned by changing the bank or patch number currently in use. Program Change allows a reflection of a change in this bank number for the channel affected, with one data byte assigned for the bank number chosen. The bank number can have a value of 0 to 127.

Message:**Channel Pressure After Touch**
Pattern: 1101 xxxx

The Channel Pressure After Touch is an interesting feature that determines the average pressure values for the unit at any instant. With this average pressure determined, a desired variation of the overall timbre value or volume of the instrument can be calculated by determining the deviation from the average required. The single data byte allowed can have a value of 0, no pressure, to 127, as much pressure as possible all around.

Message:**Pitch Wheel Change**
Pattern: 1110 xxxx

This message is one that allows a reflection of any change in the setting of the pitch wheel. As with Continuous Controller Mechanisms such as sliders, a fairly large scale is required to reflect variations in the pitch wheel setting. For this purpose, 14 bits of resolution have been provided. One odd point to note about the MIDI system specs at this point. With the Control Change message, a resolution of 14 bits was available if required. The data bytes were read High to Low. For a Pitch Wheel Change, 14 bits are also provided, but they are read Low to High. This poorly thought out variation could lead to confusion in writing code if taken for granted. So much for conventions.

Message **System Exclusive**
Pattern: 1111 0000

A System Exclusive message is one that I find to be refreshing. It allows identification of the manufacturer, and also allows the same to get his two bits in. A System Exclusive message is comprised of the first data byte signifying the identification number assigned to the manufacturer, as per the IMA (International MIDI Association In California), and as many bytes following as the manufacturer requires to tell his story. This message is terminated either by an End Of Exclusive Message (EOX - 1111 0111) or any other Status byte that happens by. Below can be found a partial list of manufacturers ID codes, as supplied by the IMA

Manufacturer	ID
Sequential Circuits Inc.	$01
Big Briar	$02
Octavew/Plateau	$03
Moog Music	$04
Passport Designs	$05
Lexicon	$06
Bon Tempi	$20
S.I.E.L.	$21
Kawai	$40
Roland	$41
Korg	$42
Yamaha	$43

Message **Song Position Pointer**
Pattern: 1111 0010

The Song Position Pointer is a 14 bit value that allows a record to be kept of the number of beats since the start of a song session. This has been incorporated for use with synthesizers equipped with a sequencer (digital recorder), or for a rhythm machine. With this feature enabled, a flag can be set to allow music to be played from a specific location within the song. The two data bytes are read Low to High.

Message **Song Select**
Pattern: 1111 0011

As with the Song Position Pointer, Song Select is also meant to be used with synthesizers equipped with a sequencer, or for a rhythm machine. Song Select uses one data byte to select which song or note sequence is to be played after a Start message has been received.

Message **Tune Request**
Pattern: 1111 0110

Tune Request is a throw back into the age of Analog synthesizers. This single Status byte, no data bytes, is used to request a tuning of the Analog synthesizers oscillators.

Message:**End Of System Exclusive (EOX)**
Pattern: 1111 0111

The EOX message is a single Status byte, no data bytes, that flags when a System Exclusive Message has been completed.

Message:**Timing Clock**
Pattern: 1111 1000

The Timing Clock Message is one that can be used to synchronize all sequencers and/or rhythm machines on line. The clock transmits its message at a rate of 6 messages per beat. As stated earlier, this type of message will appear at a regular intervals, regardless of the current state of other Status messages.

Message:**Start**
Pattern: 1111 1010

As before, this Status message is intended for use with a synthesizer equipped with a sequencer or rhythm machine. This message will inform the sequencer/rhythm machine to begin playing a pre-arranged song or note sequence from the beginning. There is only one Status Byte, no data bytes, for this message. See the Song Select Message for a little more information regarding Start.

Message:**Continue**
Pattern: 1111 1011

As before, the Continue message has been incorporated for use with synthesizers equipped with a sequencer or rhythm machine. This message is used to restart the current song sequence after receiving the next Timing Clock message. The sequence is then picked up from the next position in the Song Position Pointer. This message is flagged by the user pressing the Continue button on the sythesizer. As before, this message carries one Status byte and no data bytes.

Message:**Stop**
Pattern: 1111 1100

Again, the Stop message is for synthesizers equipped with a sequencer or rhythm machine. When received, this message tells the sequencer to stop playing its current sequence. This message carries one Status byte and no data bytes.

Message:**Active Sensing**
Pattern: 1111 1110

This message is one that is transmitted by any MIDI instrument on line, powered up, but not actively involved in anything. This message is transmitted once every 300 milliseconds if

there is no activity on the MIDI bus. One Status byte and no data bytes.

Message: **System Reset**
Pattern 1111 1111

The System Reset message performs exactly as you might expect. It tells all MIDI instruments on line to perform a power up sequence to return them to a freshly powered up state.

Retrospect

The MIDI system, as has been expressed throughout this article, is a series of well thought of specifications. Through each manufacturers implementation, musicians can use instruments of different origin and expect predictable results. This is the basic flaw in the industry today. MIDI on paper appears quite explicit in its goals. But soon after implementation, the manufacturers discovered many minute points not completely taken into consideration by MIDI Version 1.0. Today, it is this problem that confronts every musician who considers the step into the world of music synthesis.

In partial explanation, manufacturers immediately found problems implementing MIDI 1.0 following its release. Although each manufacturer tried their best to work the MIDI system within the boundaries of their machines, hindsight informed us of the inevitable. The manufacturers did not work together to make sure the systems were compatible. Each operated within their own collective vacuum, producing equipment meeting untested specifications, without considering the fact that MIDI 1.0 might be vague enough to allow multiple interpretations.

When the first MIDI machines were released on the market, problems became apparent immediately. Although many of the machines were compatible, some subtle to extreme cases of incompatibility did exist. And, as with any manufacturer faced with high R&D costs, plus further misinterpretations, many of the problems went unresolved. Take for example the Yamaha DX7 music synthesizer. It's a great machine, yet its MIDI implementation is not completely compatible with say a Roland or a Korg. In this relatively new field of digital music synthesis, it is not hard to find people who have personally become victims in this rush to compatibility.

Although I may appear to hold conflicting opinions regarding the MIDI, this is not so. MIDI in general does have its fine points. But I think that it is time for the manufacturers to get together and try to work their problems out. The specifications do not have to be altered just yet, just clarified down to the finest detail. Following this, each manufacturer can regroup to produce low cost true MIDI updates for the machines currently out. The longer this move is neglected, the greater the chance of more permanent problems. So much for a bit of sage advice.

In Summation

In closing, I would like to reflect on possible extensions to the MIDI 1.0 specifications. Due to the fact that 16 and 32 bit chips have dropped in price, implementations using these chips could be considered. With these chips, high speed communications, multi tasking, and unbelievable control over vast amounts of RAM and ROM could be taken advantage of. Further to this, we have had a chance to work with the MIDI 1.0 for quite a while now. This time of reflection has enabled musicians from all over to discover most of the weak points inherent in the system. With these points in mind, plus the technology available to us today, a true implementation could be performed, with one problem. The market has already been flooded with MIDI 1.0 equipped units. A revision at this stage would make everything else obsolete, as far as current thought allows. Due to this single fact, the MIDI specifications will not be allowed revision on a radical scale for some time to come. Although it may be hard to accept this twist of fate, the human side of this story requires consideration. Musical technology is slated to stagnate for the next few years, after which time will come an age of radical change. What our dreams are composed of today is the reality of tomorrow.

Finally, I would like to thank a few people who have helped me understand MIDI more than I ever thought possible. They include Vera Barycky, who supplied me with vast amounts of hard to get information that otherwise would be inaccessible. My father, Ted Evers, for his continuous stream of information and knowledge, and my brother, John Evers, for all his related experience working within the field, from which I was able to extract some particularly critical information.

References

The MIDI Handbook	by Paul Vytas
MIDI Specification 1.0	by the IMA in California
The Roland Juno–60 Manual	by Roland
The Roland J.:-3P Implementation	by Roland
The Yamaha RX15 Digital Rhythm Programmer Manual	by Yamaha
Keyboard Magazine	June 84 & July 85
Mix Magazine	August 1984
Computing Now	December 1984
A stack of IMA newsletters	by the IMA in California

there is no activity on the MIDI bus. One Status byte and no data bytes.

Message: **System Reset**
Pattern: 1111 1111

The System Reset message performs exactly as you might expect. It tells all MIDI instruments on line to perform a power up sequence to return them to a freshly powered up state.

Retrospect

The MIDI system, as has been expressed throughout this article, is a series of well thought of specifications. Through each manufacturers implementation, musicians can use instruments of different origin and expect predictable results. This is the least flaw in the industry today. MIDI on paper appears quite explicit in its goals. But soon after implementation, the manufacturers discovered many minute points not completely taken into consideration by MIDI Version 1.0. Today, it is this problem that confronts every musician who considers the step into the world of music synthesis.

In partial explanation, manufacturers immediately found problems implementing MIDI 1.0 following its release. Although each manufacturer tried their best to work the MIDI system within the boundaries of their machines, hindsight informed us of the inevitable. The manufacturers did not work together to make sure the systems were compatible. Each operated within their own collective vacuum, producing equipment meeting untested specifications, without considering the fact that MIDI 1.0 might be vague enough to allow multiple interpretations.

When the first MIDI machines were released on the market, problems became apparent immediately. Although many of the machines were compatible, some subtle to extreme cases of incompatibility did exist. And, as with any manufacturer faced with high R&D costs, plus further misinterpretations, many of the problems went unresolved. Take for example the Yamaha DX7 music synthesizer. It's a great machine, yet its MIDI implementation is not completely compatible with say a Roland or a Korg. In this relatively new field of digital music synthesis, it is not hard to find people who have personally become victims in this rush to compatibility.

Although I may appear to hold conflicting opinions regarding the MIDI, this is not so. MIDI in general does have its fine points. But I think that it is time for the manufacturers to get together and try to work their problems out. The specifications do not have to be altered just yet, just clarified down to the finest detail. Following this, each manufacturer can regroup to produce low cost true MIDI updates for the machines currently out. The longer this move is neglected, the greater the chance of more permanent problems. So much for a bit of sage advice.

In Summation

In closing, I would like to reflect on possible extensions to the MIDI 1.0 specifications. Due to the fact that 16 and 32 bit chips have dropped in price, implementations using these chips could be considered. With these chips, high speed communications, multi tasking, and unbelievable control over vast amounts of RAM and ROM could be taken advantage of. Further to this, we have had a chance to work with the MIDI 1.0 for quite a while now. This time of reflection has enabled musicians from all over to discover most of the weak points inherent in the system. With these points in mind, plus the technology available to us today, a true implementation could be performed, with one problem. The market has already been flooded with MIDI 1.0 equipped units. A revision at this stage would make everything else obsolete, as far as current thought allows. Due to this single fact, the MIDI specifications will not be allowed revision on a radical scale for some time to come. Although it may be hard to accept this twist of fate, the human side of this story requires consideration. Musical technology is slated to stagnate for the next few years, after which time will come an age of radical change. What our dreams are composed of today is the reality of tomorrow.

Finally, I would like to thank a few people who have helped me understand MIDI more than I ever thought possible. They include Vera Barycky, who supplied me with vast amounts of hard to get information that otherwise would be inaccessible. My father, Ted Evers, for his continuous stream of information and knowledge, and my brother, John Evers, for all his related experience working within the field, from which I was able to extract some particularly critical information.

References

The MIDI Handbook	by Paul Vytas
MIDI Specification 1.0	by the IMA in California
The Roland Juno–60 Manual	by Roland
The Roland JX–3P Implementation	by Roland
The Yamaha RX15 Digital Rhythm Programmer Manual	by Yamaha
Keyboard Magazine	June 84 & July 85
Mix Magazine	August 1984
Computing Now	December 1984
A stack of IMA newsletters	by the IMA in California

ภาคผนวก จ.

Model CT-640 MIDI Implementation Chart

Function...	Transmitted	Recognized	Remarks
Basic Default Channel Changed	1 x	1 - 4 x	(*1)
Mode Default Messages Altered	Mode 3 x **********	Mode 3 x	
Note Number: True voice	36-96 **********	0-127 36-96	0-11,12-23,24-35=36-47 97-108,109-120,121-127 =85-96(*2)
Velocity Note ON Note OFF	x 9n v=64 x 9n v=0	x 9n v=1-127 64 x 9n v=0,8n v=xx	(*2) xx.. No Function
After Key's Touch Ch's	x x	x x	
Pitch Bender	x	x	
Control Change 64	o	o	Sustain pedal (*2)
Prog Chang: True#	o 0-29 **********	o 0-29	(*3)
System Exclusive	x	x	
System :Song Pos :Song Sel Common :Tune	x x x	x x x	
System :Clock Real Time :Command	o o	x x	
Aux :Local ON/OFF :All Note OFF Mes- :Active Sense sages :Reset	x x x x	x x x x	
Notes	MIDI messages transmitted/received only when set to MIDI mode. *1) Multi messages received on CH 1-4 *2) Not received on CH-4 *3) Transission/reception of 0-19 on CH-4		

Mode 1: OMNI ON, POLY Mode 2: OMNI ON, MONO
Mode 3: OMNI OFF, POLY Mode 4: OMNI OFF, MONO
o : Yes x : No

Casiotone CT-640 Electronic Musical Instrument 's MIDI Operation Manual

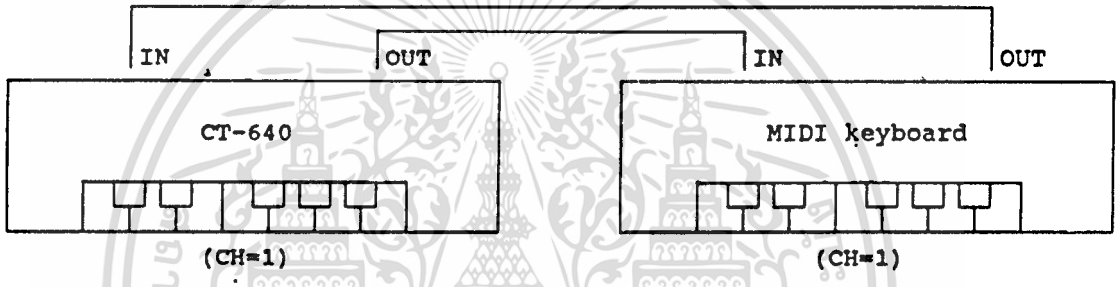
"MIDI" stands for "Musical Instrument Digital Interface". Practically speaking, it lets you connect this keyboard to other MIDI-equi musical instruments and devices, such as synthesizers, drum machines, sequencers, and even personal computers.

MIDI IN	Receives MIDI signal from external device.
MIDI OUT	Sends MIDI signal to external device.
MIDI THRU	Passes unchanged signal received from one device through MIDI IN to another device.

* Set Chord/MIDI selector to "MIDI" to use MIDI function of this keyboard.

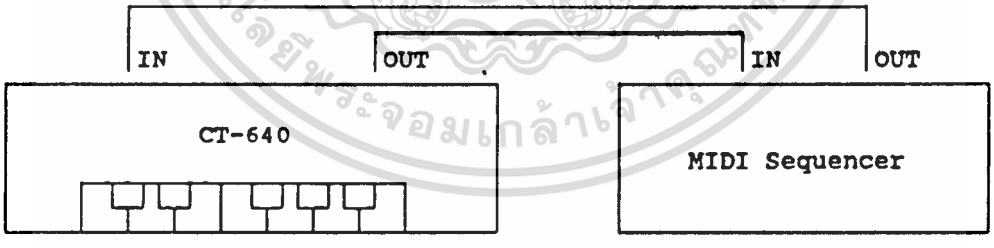
[1] Transmits/Receives MIDI massages as a single keyboard (6-note poly).

<EXAMPLE CONNECTION 1>



[2] Splits keyboard into 3 sections, a 6-note poly section, a 4-note poly section and a 2-note poly section for use as a multiple sound source. The tone of each channel can be freely changed by program change messag (refer to "MIDI Data Transmit/Recieve" for details).

<EXAMPLE CONNECTION 2>



CH=1	freely assigned	(CH=1~4)
CH=2	to one of the	(CLOCK=EXT)
CH=3	preset tones	
CH=4 Auto-rhythm **		

(CLOCK=INT)

** Only program change can be recieved.

Use as a MIDI precussion sound source

It is possible to use this keyboard as a MIDI percussion sound sourc
as all percussion instrument sounds are assigned independent MIDI note numbe
as listed below;

Instruments	Note Number	Percussion sound
1	36/38	bass drum 1/2
2	40/41	snare drum 1/2
3	43	gated snare drum
4	45	limb short
5	47/48	closed hihat 1/2
6	50/52	open hihat 1/2
7	53/55/57/59	crush cymbal 1/2/3/mixed
8	54/56/58	synth. cymbal 1/2/3
9	60/62	ride cymbal 1/2
10	61/63	gong 1/2
11	64/65/67/69/71	tom 1/2/3/4/mixed
12	66/68/70	synth. tom 1/2/3
13	72/74	cow bell 1/2
14	73/75	triangle open/mute
15	76/77/79/81	timbales; high mute/high/middle/low/mixed
16	78/80/82	computer game sound 1/2/3
17	83/84	agogo high/low
18	86/88/89/91/93	conga; high mute/high/middle/low/mixed
19	95/96	clave high/low

MIDI Data Transmit/Receive

this unit is capable of transmitting/receiving the following MIDI data.

<Receiving>

MESSAGES		CH 1	CH 2	CH 3	CH 4
1	Note on/off	o	o	o	x
2	Program change	0-29 (*1)			0-19 (*2)
3	Sustain on/off	o			x
4	Clock	x			

<Transmitting>

MESSAGES		CH 1	CH 4
1	Note on/off	o	x
2	Program change	0-29 (*1)	0-19 (*2)
3	Sustain on/off	o	x
4	Clock	o	

(*1) changing preset tones.
(*2) changing auto-rhythms.

ภาคผนวก ฉ.

บทที่ 10
ศัพท์และเครื่องหมายเพื่อการปฏิบัติ

1. ศัพท์ทางดนตรี มีกำหนดไว้เป็นภาษาต่าง ๆ เช่นภาษาอิตาเลียน ภาษาฝรั่งเศส ภาษาเยอรมัน ตลอดจนภาษาลาติน นักดนตรีและนักศึกษาวิชาดนตรี จึงควรรู้ศัพท์ ตลอดจนจดจำเครื่องหมายทางดนตรี เพื่อประโยชน์ในการปฏิบัติทางดนตรีได้ถูกต้อง ไพเราะตามที่บทเพลงนั้นกำหนดไว้ สำหรับศัพท์ที่จะศึกษาในเบื้องต้นนั้น คือ ภาษาอิตาเลียนเป็นส่วนมาก พร้อมกับมีคำอ่าน คำแปลเป็นภาษาอังกฤษ และภาษาไทยกำกับไว้ด้วย

เราจึงควรจำและเขียนสะกดคำศัพท์ทางดนตรี ตลอดจนเครื่องหมาย คำย่อต่าง ๆ ได้ถูกต้องแม่นยำ โดยใช้หลายวิธี และวิธีหนึ่งก็คือ ให้หมั่นท่องจำและเขียนไปด้วยขณะท่อง ครั้งละจำนวนน้อย ๆ ซึ่งจะเป็นครั้งละ 3-5 คำก็ได้ แล้วจึงค่อยท่องคำต่อไป เพื่อเพิ่มจำนวน ไม่ควรเร่งโหมดูศัพท์และเครื่องหมายเมื่อใกล้กำหนดทดสอบ เพราะอาจทำให้สับสนจำไม่ได้

2. ศัพท์ เครื่องหมาย และคำย่อ แบ่งเป็นหมวดใหญ่ ๆ 2 หมวดคือ

2.1 แสดงความเข้มหรือความดัง-ค่อยของเสียง

2.2 แสดงความเร็ว-ช้าของจังหวะดนตรี

2.1 คำศัพท์และเครื่องหมายแสดงความเข้มหรือความดัง-ค่อยของเสียง

ภาษาอิตาเลียน	คำอ่าน	อักษรย่อหรือเครื่องหมาย	คำแปลภาษาอังกฤษ	คำแปลภาษาไทย
Pianissimo	ปิอานิสซีโม	*pp*	Very soft.	แผ่วเบามาก
Mezzo Piano	เมสโซปิอาโน	*mp*	Moderately soft.	แผ่วเบาปานกลาง
Piano	ปิอาโน	*p*	Soft.	เบา
Mezza Voce	เมสซาโวเช	*m'*	Medium tone.	เบาปานกลาง
Mezzo Forte	เมสโซ ฟอร์เต	*mf*	Moderately loud.	ดังปานกลาง
Forte	ฟอร์เต	*f*	Loud.	ดัง
Fortissimo	ฟอร์ติสซีโม	*ff*	Very loud.	ดังมาก
Crescendo	เครสเชนโด	<	Gradually becoming louder.	เพิ่มความดังขึ้นทีละเล็กทีละน้อย ตามลำดับ
Decrescendo	เดเครสเชนโด	>	Gradually becoming softer.	ลดความดังลงทีละเล็กละน้อย ตามลำดับ
Diminuendo	ดิมินูเอนโด	>		
Sforzando	สะฟอร์ซานโด	*sf*	Accented.	เน้น
Forzando	ฟอร์ซานโด	*fz*		
Rinforzando	รินฟอร์ซานโด	*rf*	Strengthening the tone.	เพิ่มความหนักแน่นเสียงนั้น

2.2 คำศัพท์แสดงความเร็ว ช้าของจังหวะดนตรี จากช้ามากที่สุดไปยังเร็วที่สุด

ภาษาอิตาเลียน	คำอ่าน	คำแปลภาษาอังกฤษ	คำแปลภาษาไทย
Grave	กราเว	Extremely slow, solemn.	ช้ามากที่สุดอย่างเศร้าโศก
Lento	เลนโต	Slow.	ช้ามาก คล้ายพระเดินจงกรม
Largo	ลาร์โก	Broad.	ช้ามากอย่างเดินยืดยาด
Larghetto	ลาร์เก็ทโต	Rather broad.	ค่อนข้างช้า
Adagio	อะดาโจ	Slow, leisurely.	ช้าอย่างสบายอารมณ์ แบบชมนกชมไม้
Andante	อันดานเต	Going at an easy pace.	ช้าอย่างเดินไปด้วยความสบาย ไม่เร่งร้อน
Andantino	อันดานติโน	At a moderate pace but not so slow as Andante.	ช้าปานกลาง แต่ไม่เท่าอันดานเต
Moderato	โมเดอราโต	Moderate speed.	ช้าปานกลาง
Allegretto	อัลเลเกร็ทโต	Rather fast.	ค่อนข้างเร็ว
Allegro	อัลเลโกร	Fast.	เร็ว
Vivace	วิวาเช	Lively.	เร็วอย่างมีชีวิตชีวา
Presto	เพร็สโต	Very quick.	เร็วมาก
Prestissimo	เพร็สติสซีโม	Very quick indeed as fast as possible.	เร็วมากที่สุดเท่าที่จะเร็วได้

3. คำศัพท์และเครื่องหมายต่อไปนี้ มีความเกี่ยวข้องกับความเร็ว-ช้า ซึ่งเรามักพบเสมอคือ

ภาษาอิตาเลียน	คำอ่าน	คำย่อหรือเครื่องหมาย	คำแปลภาษาอังกฤษ	คำแปลภาษาไทย
Accelerando	อักเซเลรานโด	*accel.*	Getting gradually faster.	เพิ่มความเร็วขึ้นเรื่อย ๆ
Rallentando	รอลเลนทานโด	*rall.*	Getting gradually slower.	ให้ช้าลงเรื่อย ๆ
Calando	คาลานโด		Softer and slower.	ให้ทำเสียงทั้งบานะช้าลงด้วย
Ritardando	ริทาร์ดานโด	*ritard*, *rit.*	Retarding the speed.	ทำให้ช้าลง(ทีละเล็กละน้อย)
Ritenuto	ริทเทนิวโต	*riten.*	Held back.	รั้งให้ช้าลงทันทีทันใด
A Tempo	อะ เทมโป		In time.	กลับคืนสู่จังหวะเดิมที่ได้ปฏิบัติไว้ตั้งแต่เริ่มแรก
Ad Libitum (ภาษาลาติน)	แอ็ดลิบิตุม	*ad lib.*	At the performer's pleasure.	ช้าหรือเร็วได้ ตามใจผู้ปฏิบัติ (คำนี้อาจใช้ในการปฏิบัติการทางดนตรีอื่น ๆ ว่าให้ได้ตามอำเภอใจอีกด้วย)
A Piacere	อะ ปิอาเชเร			
Meno Mosso	เมโน ม้อสโซ		Slower at once.	ช้าลงทันทีทันใด
Piu Mosso	ปิอุ ม้อสโซ		Quicker at once.	เร่งเร็วขึ้นทันทีทันใด
		∧ , >	Accents.	เน้นให้ชัดเจน

ภาคผนวก ช.

OFF THE RECORD

It takes all three Johnnies to meet the challenge of synth simplicity (L to R): Mike Nocito, Clark Datchler, Calvin Hayes.

SHILA ROCK

JOHNNY HATES COMPLEXITY? NO, IT'S

JOHNNY HATES JAZZ

By David Leytze

SIMPLICITY. IT'S MORE THAN JUST a word, it's a philosophy. But achieving true simplicity is always harder than it looks. In the incessant struggle for survival in the Top 40 jungle, there's a tendency to add layer after layer of gimmicks to a record in the hope that somehow the combination will be magical. Add enough gimmicks and what you get is a muddy mess. So you cut back, and then you've got not a lean, mean, fighting song but a scattered bunch of unconnected bones.

However, the distant star of simplicity is worth reaching for. From time to time, amidst the cluttered throb of sampled mayhem and digital disorder, there emerges an artist or group who has the right combination—the songwriting is right in line, the sounds are crystal-clear, the vocals are in tune.

One of the recent survivors to emerge from the studio quagmire is England's Johnny Hates Jazz—three minstrel masterminds who have the popular style down to a science. All of their songs are in the key of simplicity. "We like simple things," says co-songwriter/keyboardist/producer Calvin Hayes. "We're always trying to simplify parts. I think trying to use pads melodically, rather than just rhythmically, is the way to go. With the advent of drum machines and all of the technology, everybody started making music in the same way. You can make melodic music with this technology. That's what we've gone for."

But who are these guys? Did they come out of nowhere and hit the absolute top of the charts? Not exactly. Hayes had established contacts in the record industry through his work as an A&R man at RAK Records, and Mike Nocito had engineered for Duran Duran and the Thompson Twins. They may be new to America, but they're not new to Britain or the music industry. The band's debut single, "Shattered Dreams," went number one halfway around the world without even having an album to support it. "After we signed a single deal with Virgin Records," explains Nocito, "we recorded 'Shattered Dreams' and one-and-a-half other songs. When the single came out, it was a big hit in England. Then the record company said, 'Where's the album?' Of course, the album didn't exist. Virgin waited a long time to commit to an album deal. Once they did commit, we spent six or seven months doing the album. We tried to work constantly but we were away promoting quite a bit. We'd go away to Germany or somewhere, and then come straight back and go into the studio. Even 'Shattered Dreams' happened really fast. After we had recorded the tracks, we

Continued on page 67

"SHATTERED DREAMS"

By David Leytze

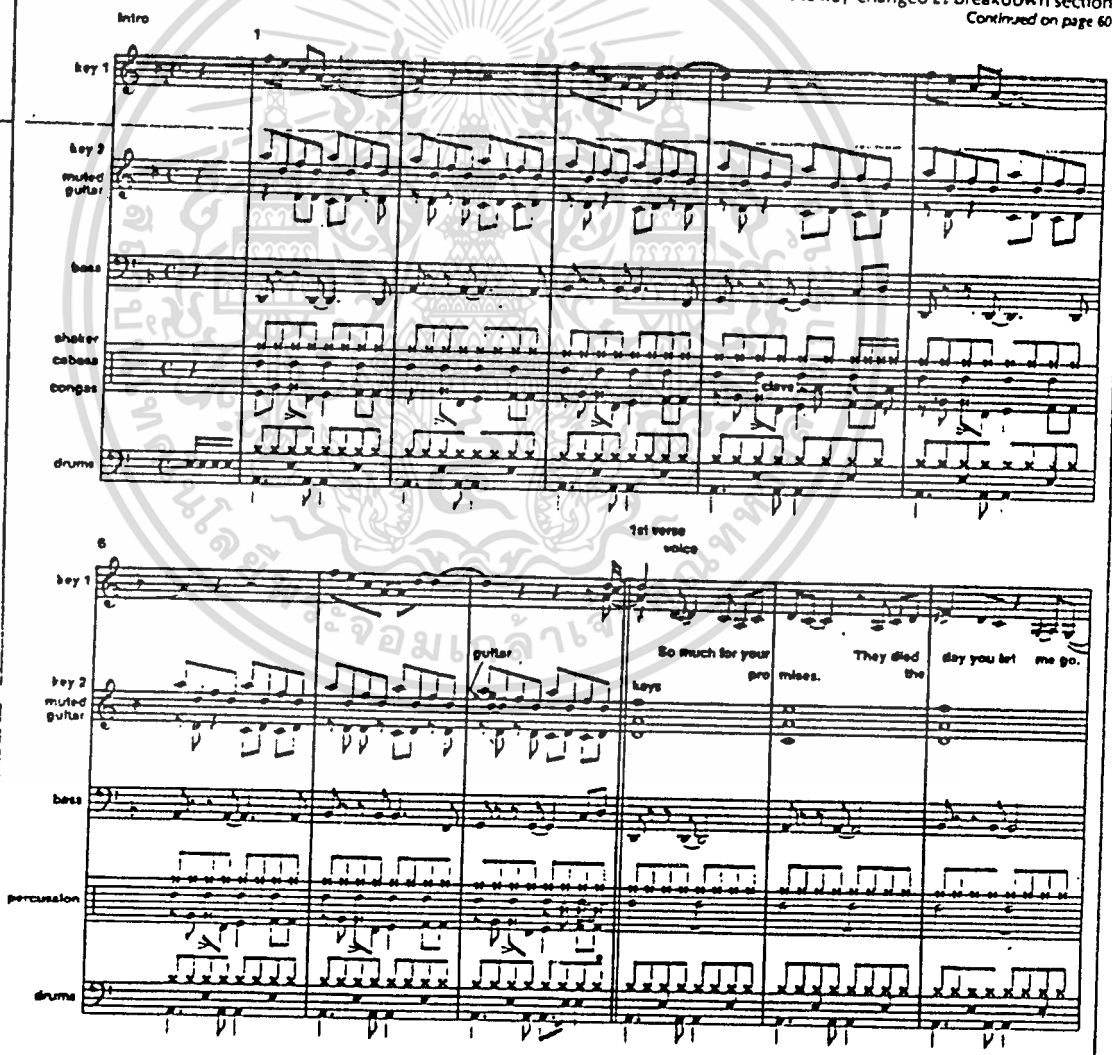
VERSE, CHORUS, VERSE, CHORUS—what could be simpler than the structure of a popular song? If that's your attitude, think again. There are an incredible number of possible forms. Some tunes use the melody of the chorus as the intro, some use different chords for the verses than for the choruses—while a few basic elements predominate, they can be combined in endless ways. One combination that's common today is using a single chord progression for both the verse and chorus, and separating them with a bridge of tension-building chords that resolve back into the base progression. Such is the case with Johnny Hates Jazz' "Shattered Dreams."

The verse and chorus of "Shattered Dreams" are based on a four-bar *Dm-Am-B♭-Gm* chord progression. The verse melody is different from the chorus melody, however, and there are changes in instrumentation. Even when a single instrument carries through from verse to chorus, the actual notes that are chosen may change. (Look at the bass line, for example.) On the other hand, the progression is simple enough to allow a two-beat *D* minor figure (transcribed here on the second staff) to repeat throughout with no change at all.

Throughout the song, the melody sticks to a simple pentatonic scale centered around Middle C. A single *E* in the bridge phrase is the only non-pentatonic note. Rhythmically, the melody of the chorus uses a type of displacement, in which the emphasis shifts in relation to the downbeat. This effect is what you'll hear on the words "given me, given me," "shattered dreams, shattered dreams," and "run away, run away." The melodic repitition of the *F* to *D* is strengthened by the repetition of the lyrics.

The instrumentation and arrangement of the choruses is closer to the intro than to the verses. The instrumental parts are identical between the intro and first chorus (muted guitar, congas, etc.), and then begin to build with each new chorus. Horns and bongos (which are also present in the key-changed *E♭* breakdown section

Continued on page 60



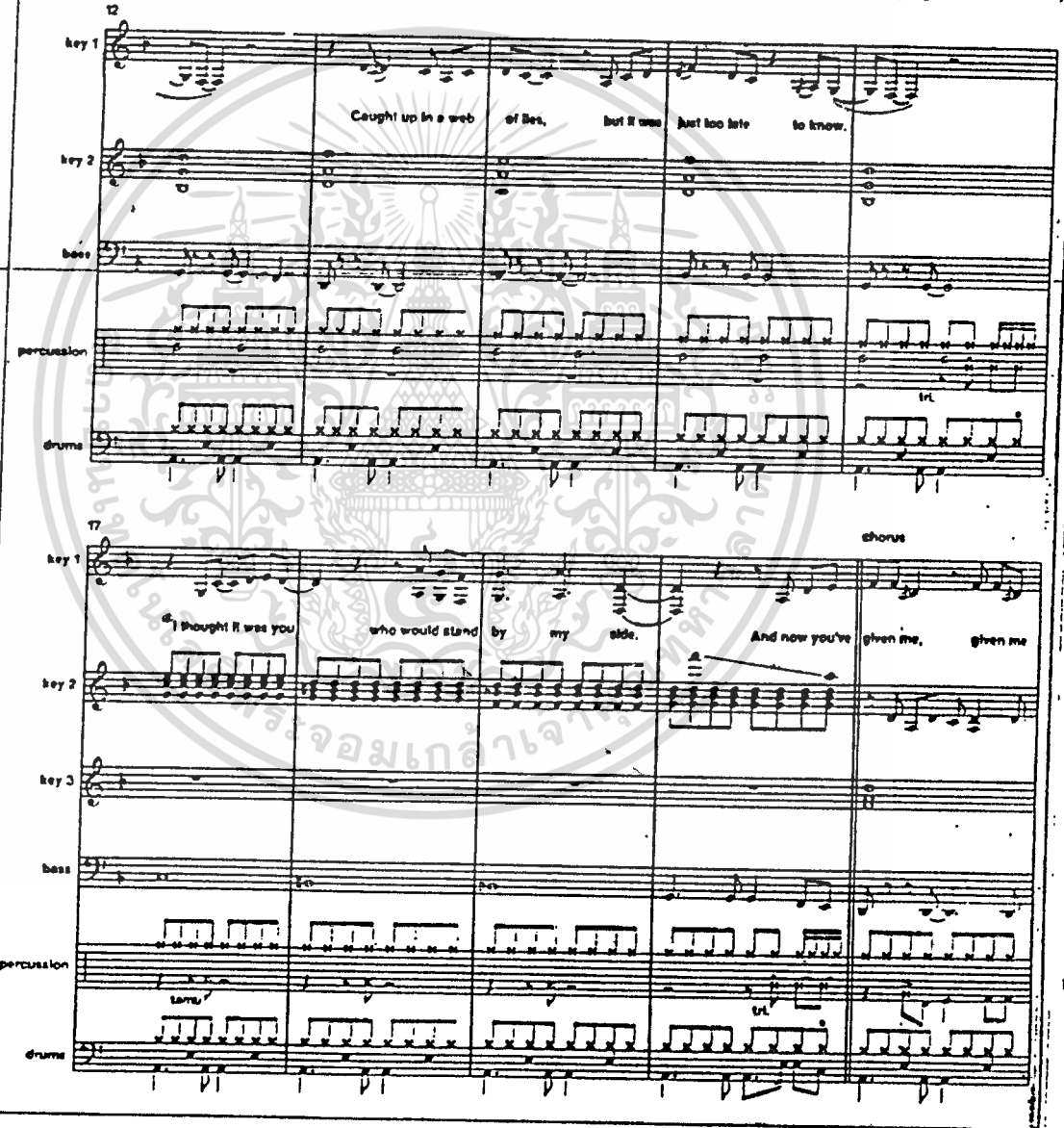
"SHATTERED DREAMS"

Continued from page 59

in the middle of the tune) are added in subsequent choruses, and the bass line is embellished further. The last four bars of the first chorus contain the exact music of the intro, except that the bass plays a C under the *A* minor chord.

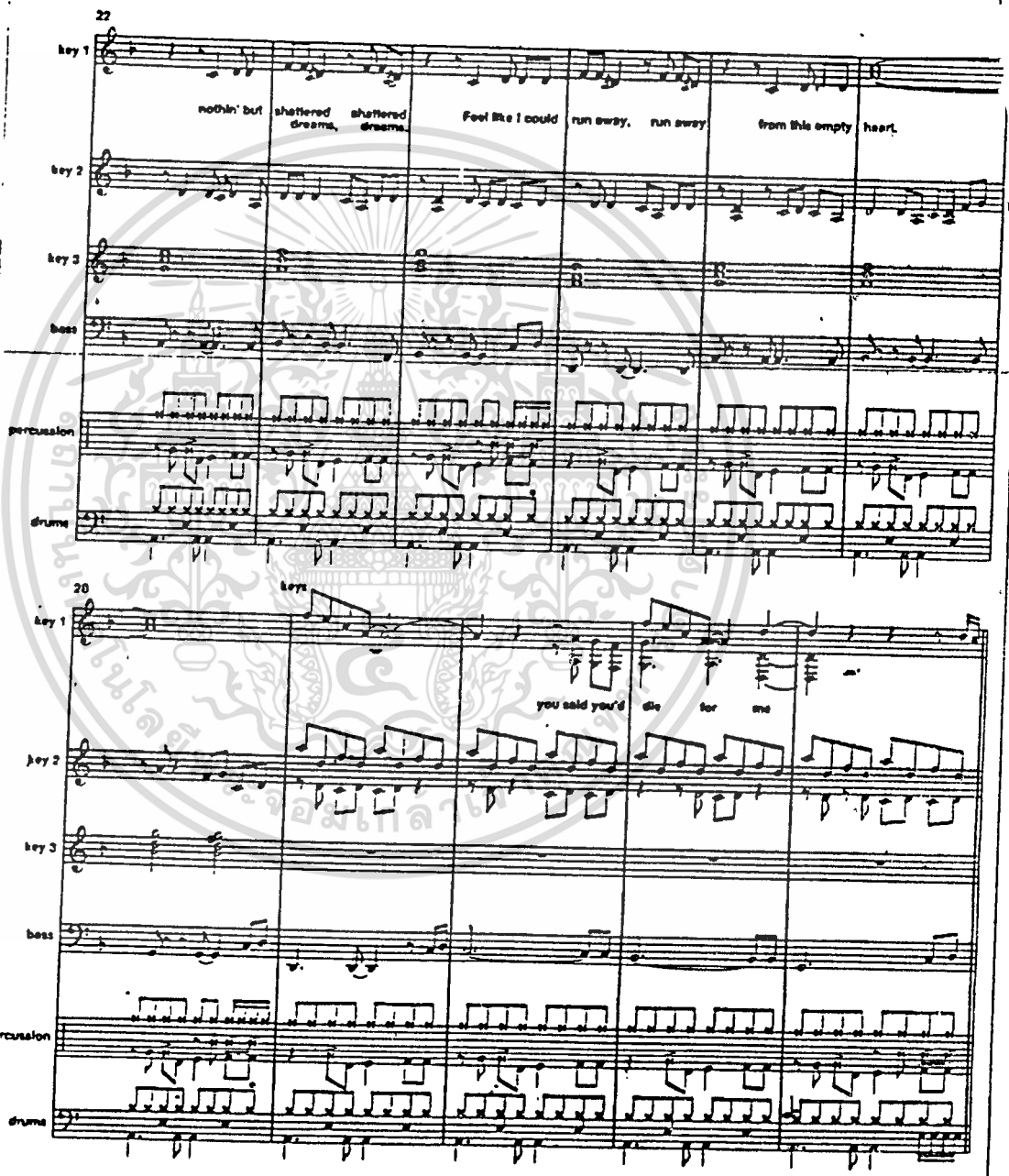
The art of simplicity in action: When the first verse begins, after the eight-bar intro, all rhythm guitar and moving keyboard parts drop out, along with all percussion parts except the shaker and cabasa, leaving a whole-note keyboard pad, a bass line, and a drum pattern to support the vocal melody. The bass line goes to a more settled rhythm pattern, omitting the pickup eighth-note used in each bar of the intro. The second verse varies slightly in that the keyboard pad is played an octave higher. By listening closely to these verses, you should get a better idea of how such a simple orchestration works.

The four-bar bridge phrase is a different story. Here, the progression changes, following a descending C-*B*-B♭-G bass line. Beginning on the seventh degree of the tonic (*D* minor) scale, this progression serves as a tension builder so that the beginning of the upcoming chorus will feel like a resolution. Only the first two chords of this phrase (C and *G* major) are actually different, however; the last two chords are the same as in the basic progression, though they are orchestrated a bit differently. The melody leaves plenty of air for the underlying eighth-note chords,



which create rhythmic tension. The orchestration thickens a bit with the addition of the eighth-note keyboard chords and very transient tambourine taps. The bridge's final push starts with an octave doubling tagged on top of the vocal line. (Note that this melodic phrase is repeated at the end of the chorus, woven into a call-and-response pattern with the instrumental melody from the intro.) In the last bar before the chorus the snare drum skips the second beat and kicks the fourth beat with an extra hit. A sliding sound in the background and a triangle pickup (repeated from the intro) give some added push.

We'd like to thank Katie Elliott at Virg- Records and Johnny Hates Jazz (Clar Datchler, Calvin Hayes, and Mike Nocito for making it possible for *Keyboard* t publish portions of "Shattered Dreams."



กิตติกรรมประกาศ

โครงงานนี้ไม่อาจจะลุล่วงไปได้ด้วยดีหากไม่ได้ความช่วยเหลือจากท่านเหล่านี้

1. ผศ.ครรชิต ไมตรี ผู้เป็นอาจารย์ที่ปรึกษาและเอื้อเฟื้ออุปกรณ์ตลอดจนสถานที่ตลอดระยะ เวลา 1 ปีของการทำงาน
2. อาจารย์วิศิษฐ์ แห่งโรงเรียนดนตรีสยามกลกาล ให้คำปรึกษาชี้แนะแนวทาง
3. พี่ๆจากบริษัทไนท์สปอตโปรดักชั่น เอื้อเฟื้อเครื่องอะคารีให้ศึกษา
4. พี่ๆจากบริษัทไมโครชอป ให้คำแนะนำทางด้านฮาร์ดแวร์ ให้ดู MPU 401 ตัวจริง
5. น้องอุทัย และน้องๆวิชาการที่ ร่วมมือด้วยดี พร้อมทั้งให้โน๊ตสำหรับมิดิมาลองเล่น
6. คุณ กฤษณ์ชัย สมสมาน เอื้อเฟื้อโทนเจนเนอเรเตอร์
7. คุณ กิจจา นาคใหม่ เอื้อเฟื้ออุปกรณ์บางชิ้น
8. คุณรัตนะ และ คุณสุรเดชที่ให้ความสะดวกตอนทำโปรเจค

หนังสืออ้างอิง

1.Roger Powell ,"The Challenge of Music Software ",BYTE, Vol 11,No.6,1986,pp.145-150

2.Jay Kubicky , "A MIDI Project" , BYTE ,Vol 11 ,No.6 , 1986 ,pp. 199-208

3.Roland Corporation ,"MIDI Guide Book",Vol.4,1987,pp.1-29

4.พิพัฒน์ วัฒนชัยพงศ์,"ตำราเรียนเล็ดคอร์ดกีตาร์",Vol 1.

5.พิพัฒน์ วัฒนชัยพงศ์,"ตำราเรียนเล็ดคอร์ดกีตาร์",Vol 2.

6.ริลดา ติรณสวัสดิ์ และ มีโชค อมรพัฒนกุล,"คอมพิวเตอร์มิวสิค 2", คณะวิศวกรรมศาสตร์ ลาดกระบัง, 2532

7.Richard Evers,"MIDI",The Transactor,Vol 6.No.5,March 1986, pp.28-33

8.Microsoft Corporation,"Microsoft Windows Software Development Kits,Programming Learning Guide" ,Microsoft , 404 p.,1987

9.Microsoft Corporation,"Microsoft Windows Software Development Kits,Programmer Reference" ,Microsoft , 700 p.,1987

10.Microsoft Corporation,"Microsoft Windows Software Development Kits,Programmer Tools" ,Microsoft , 597 p.,1987

11.James W.Coffron,"Z-80 Application",SYBEX Inc. ,259 p. , 1983

12.John Uffenbeck,"Microcomputers and Microprocessors" , Prentice - hall, 670 p. , 1985

13.Peter Norton,"Norton Guide",Peter Norton (Software)

14.พิชัย ปรัชญานุสรณ์,"ทฤษฎีดนตรี 1", 58 หน้า,2532

14.พิชัย ปรัชญานุสรณ์,"ทฤษฎีดนตรี 2", 68 หน้า,2532

15.ยืน ภู่วรวรรณ ,ดร.ชัยยงค์ วงค์ชัยสุวัฒน์,ดร.ไพศาล สงวนหมู่,"เทคโนโลยีไมโครคอมพิวเตอร์ 16 บิท",ซีเอ็ดยูเคชั่น,342 หน้า,2530