ปีการศึกษา 2530

ปริญญานิพนธ์ เรื่อง

SPEECH SYNTHESIS

โดย

นายสังวรณ์ อยู่ยง 29.6320

นายพงเดช ชลวิไล 29.6310

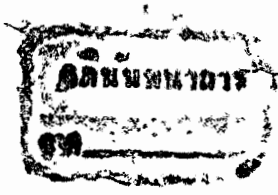
อาจารย์ที่ปรึกษา

อาจารย์ กิตินันท์ รัตนธำรงค์

ภาควิชาเทคโนโลยีวัดคุมอุสาหกรรม

คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้า ลาดกระบัง



ปริญญานิพนธ์ปีการศึกษา 2530

เรื่อง SPEECH SYNTHESIS

ผู้จัดทำ

1. นายสังวรณ์ อยู่ยง
2. นายพงเดช ชลวิไล

.....อาจารย์ที่ปรึกษา

(อาจารย์ กิตินันท์ รัตนธำรงค์)

SPEECH SYNTHESIS

นายสังวรณ์ อยู่ยง
นายพงเดช ชลวิไล

กิตตินันท์ รัตนธำรงค์ อาจารย์ที่ปรึกษา
ปีการศึกษา 2530

บทคัดย่อ

โครงงาน SPEECH SYNTHESIS เป็นการประยุกต์เครื่องไมโครคอมพิวเตอร์ IBM PC/XT มาสร้างเป็นส่วนจัดเก็บสัญญาณเสียง และหาข้อมูลที่จำเป็นอื่นๆ เช่นจำนวนไบท์ต่อคำ ตำแหน่งของหน่วยความจำ แล้วจึงนำข้อมูลที่ได้ไปบันทึกหน่วยความจำ เพื่อนำไปใช้ในบอร์ด Z-80 ที่มีวงจรสร้างเสียง ซึ่งควบคุมการสร้างเสียงจากบอร์ดภายนอก

SPEECH SYNTHESIS

SUNGVON YUYONG

PHONGDET CHONVILAI

KITTINUN RATTANATUMRONG ADVISOR

ACADEMIOS YEAR 1987

ABSTRACT

"Speech synthesis" project is apply IBM PC/XT Computer for manages audio signal and other information. These information are about number of byte in one speech-word, position of speech-word and use these information for program memories. These memory are used in Z-80 processor board which has "Speech synthesis circuit".

สารบัญ

บทที่ 1 บทนำ

ปริญญานิพนธ์ชุดนี้ เป็นการนำเอาวงจรอิเลคทรอนิคส์มาทำการบันทึกและถ่ายทอดเสียงแทนเทป เพื่อให้ความสะดวกและคล่องตัวในการใช้งานมากกว่า ถึงแม้วงจรอิเลคทรอนิคส์จะบันทึกหรือเก็บเสียงได้ไม่เท่ากับเทป แต่ก็ให้ประโยชน์ในการใช้งานในโครงงานนี้อย่างคุ้มค่า โครงงานนี้แบ่งการสร้างเป็น 2ส่วน คือส่วนที่ใช้ไมโครคอมพิวเตอร์ IBM PC/XT เป็นส่วนเก็บสัญญาณเสียง ตัดต่อคำ และหาขนาดของคำว่ามีจำนวนกี่ไบท์ และส่วนที่ใช้ Z-80. ควบคุมการถ่ายทอดเสียงตามความต้องการของบอร์ดแม่ โดยมีส่วนอินเตอร์เฟสกับบอร์ดแม่ที่สั่งให้โครงงานนี้ถ่ายทอดเสียงตามรูปแบบที่กำหนดไว้ก่อนแล้ว สำหรับรายละเอียดแต่ละส่วนของโครงงานนี้อยู่ในบทที่ 3 แล้ว

บทที่ 2 หลักการพื้นฐานและการทำงานเบื้องต้น

มีวิธีพื้นฐานอยู่ 3 วิธี ที่จะกำเหนิดเสียง โดยใช้วิทยาศาสตร์ และคอมพิวเตอร์ ส่วนบุคคล คือ

1 wave*from* endecode

2 analog formant synthic for photic speech

3 linear predictive code

ในบทนี้จะได้อธิบายถึงรายละเอียด ความสามารถและข้อจำกัดของแต่ละวิธี และเข้าใจถึงความแตกต่างระหว่าง speech synthise ในปัจจุบัน เทคนิคเหล่านี้จะทำให้เลือกแนวทางที่เหมาะสมถ้าดูความเชื่อถือและความหมายของเสียงจากเอ้าพุทของคอมพิวเตอร์

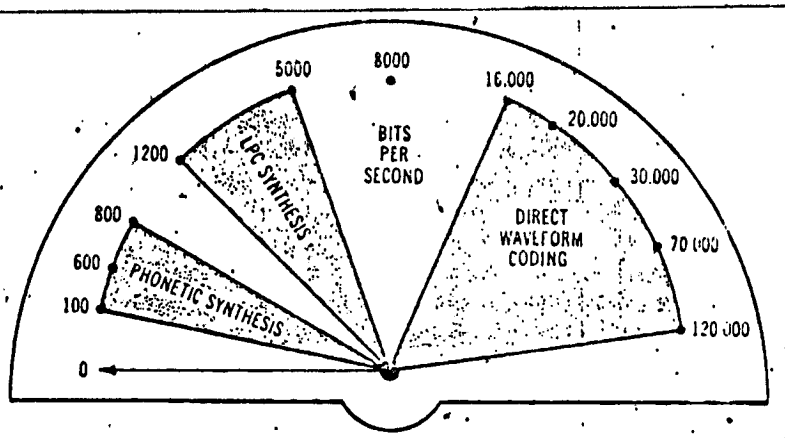
เราได้กล่าวมาแล้วว่ามีวิธีการพื้นฐาน 3 วิธี อาจจะเปรียบเทียบวิธี speech synthic ได้กับวิธีการขนส่ง เช่น รถยนต์ขนส่งมี 3 ชนิด ได้แก่ รถธรรมดา , รถสปอท, รถบรรทุก รถแต่ละชนิดมีจุดประสงค์การใช้ต่างกัน แต่ถึงที่หมายเหมือนกัน สิ่งที่แตกต่างกันคือราคา , ความเร็ว แต่ผลสุดท้ายถึงที่หมายเหมือนกัน

เทคโนโลยี่ในการสังเคราะเสียงก็เหมือนกัน เป็นสิ่งไม่มีตัวตนที่จะเปรียบเทียบได้ แต่ทั้ง 3 วิธี ก็มีจุดประสงค์หนึ่งคือ ให้เอ้าพุทจากคอมพิวเตอร์ของคุณ นั่นหมายความว่า รูปคลื่นของเสียง อาจได้จากการ encoding/reconstruction, Phenetic format หรือ matchtical ขึ้นอยู่กับความต้องการของ ราคาของอุปกรณ์ และความสามารถของระบบคอมพิวเตอร์ แต่ละชนิดมีคุณสมบัติแตกต่างกัน

SPEECH SYNTHESIS "TACHOMETER"

เพื่อให้เห็นภาพการเปรียบเทียบแต่ละวิธี ขอให้ดูในรูป 2-1 "tachometer" จะให้ความสำคัญของความเร็วคอมพิวเตอร์ที่ต้องการ กับชนิดของการสังเคราะห์เสียง ในปัจจุบันความเร็วไม่เป็นปัญหาอีกแล้วสำหรับคอมพิวเตอร์ส่วนบุคคล ความเร็วของการสุ่มสัญญาณในหนึ่งวินาที สัมพันธ์กันโดยตรงกับจำนวนหน่วยความจำ ที่ต้องใช้เก็บสัญญาณเสียง ถ้าเปรียบเทียบกันโดยอ้อมก็คือ เปรียบเทียบจำนวนหน่วยความจำที่ใช้ ดังนั้น "techometer" จึงแสดงความสัมพันธ์ของปัญหาการใช้หน่วยความจำเก็บสัญญาณเสียง

ยกตัวอย่าง สมมุติว่าคอมพิวเตอร์ มีการจัดหน่วยความจำ แบบ 8-บิท/ไบท์ แต่ "techometer" มีการจัดหน่วยความจำ เป็นแบบบิทต่อวินาที ขณะนี้เราสมมุติ ว่า แต่ละเทคนิคมีประสิทธิภาพในการเก็บคำพูด เช่น คำว่า "hello" จำนวนเวลาของการพูด คำว่า "hello" แปรเปลี่ยนได้แล้วแต่ การพูดของ



รูป 2-1 A - speech synthesis tachometer

แต่ละคน แต่ประมาณได้ 0.3 วินาที ดังแสดงในตาราง 2-1 ซึ่งเปรียบเทียบวิธีพื้นฐานทั้ง 3 วิธี ในแบบจำนวนหน่วยความจำที่ต้องการเก็บ ประสิทธิภาพที่แตกต่างกันมากมาย แต่ละวิธีมีจำนวนอัตราบิท ที่แตกต่างกัน จำนวนอัตราบิทที่ต่ำ ๆ จะให้คุณภาพของเสียงต่ำด้วย เนื่องจากการ loss ของข่าวสาร จำนวนอัตราบิทที่สูง ๆ จะให้สัญญาณที่ออกมาชัดเจนเหมือนเสียงคนปกติ ที่คอลัมที่ 3 แสดงความต้องการหน่วยความจำในการเก็บสัญญาณ (ในจำนวนไบท์) ที่จะพูดคำว่า "hello" มีตั้งแต่ 4 ไบท์ สำหรับเสียงคุณภาพต่ำ จนถึง 4500 ไบท์ ในแบบ direct wave from endecode คุณภาพสูง สำหรับเสียงคุณภาพต่ำ สัญญาณเสียงก็พอเข้าใจได้ แต่มีขีดจำกัดด้านความสมบูรณ์ของคำ

Synthesis Technique	Bit Rate Per Second	Storage Required In Bytes for "Hello"	Amount of Total Speech In 48K Byte Computer
Phonetic	100-800	4 to 30	1 hour 4 min. to 8 min.
LPC	1200-5000	45 to 188	5 min. 20 sec. to 1 min. 17 sec.
Waveform Coding	16,000-120,000	600 to 4500	24 sec. to 3.2 sec.

ตาราง 2-1 การเปรียบเทียบเทคนิคสังเคราะห์เสียง

การจัดเก็บสัญญาณเสียง

คอลัมภ์ในตาราง 2-1 แสดงผลกระทบต่อจำนวนความสามารถในการเก็บสัญญาณเสียงของคอมพิวเตอร์ส่วนบุคคลขนาด 48K จำนวนสูงสุดสำหรับเก็บในแบบ phonetic speech จะเก็บได้ 4 ชั่วโมง 4 นาที นี้ยังไม่รวมถึงพื้นที่ของโปรแกรม ที่จะจัดการนำข้อมูลออกมาเป็นสัญญาณเสียง ถ้าเราดูที่ waveform coding ความเร็วสูง ประสิทธิภาพในการเก็บต่ำสุด จะเก็บสัญญาณได้เพียง 3.2 วินาที ในหน่วยความจำของคอมพิวเตอร์ขนาด 48 K

จะต้องละลึกเสมอว่าหน่วยความจำเหล่านี้ ต้องการเก็บเฉพาะสัญญาณเท่านั้น ยังจะต้องรวมคำศัพท์ที่ต้องการให้พูดอีก ตัวอย่าง ถ้าต้องการคำศัพท์ 100 คำ จากการสร้างประโยคและถ้อยคำ ต้องสมมุติว่าแต่ละคำในกลุ่มคำศัพท์ กำหนดให้เหมาะสมประมาณ 1/2 วินาทีต่อคำ ดังนั้นเวลาที่น้อยที่สุดที่ต้องเก็บคือ 50วินาที ก่อนที่จะเริ่มสร้างประโยค ถ้าเราดูตาราง 2-1 เราจะเห็นว่า waveform coding จะเก็บได้ 48 คำ (24 วินาที) จะเห็นว่าไม่พอที่จะทำการเก็บ ต้องใช้เทคนิคอื่นๆ ช่วย แทนที่จะเก็บเป็นคำ ก็เก็บเป็นวลี และใช้การเรียกแต่ละวลี ตามความต้องการแทนการเรียกแบบเป็นคำอิสระ ทำให้ขนาดหน่วยความจำมีขนาดเล็ก มีที่ว่างพอที่จะเก็บโปรแกรมมอนนิเตอร์รวมอยู่ได้

คอมพิวเตอร์สังเคราะห์เสียงพูด

(computer synthetic speech)

จุดสุดท้ายที่จะพิจารณา ในการเปรียบเทียบทั้ง 3 วิธี สำหรับสังเคราะห์เสียงให้คำจำกัดความได้ว่าเป็น "synthetic speech" จากเหตุผล 2 ข้อ

เหตุผลแรก คอมพิวเตอร์ ที่สร้างเสียงจากการสังเคราะห์ ซึ่งเปล่งเสียงมาจากคอมพิวเตอร์มีข้อจำกัดในเรื่องหน่วยความจำ เทคโนโลยีทางดิจิตอลที่ใช้ในการเก็บสัญญาณ ในความจริงคำที่เปล่งมาจากคอมพิวเตอร์ที่ทำขึ้นมาเอง หรือจากวงจร IC เพื่อสร้างสัญญาณขึ้นมาใหม่ มี 2 วิธีที่จะสร้างสัญญาณขึ้นมาใหม่ ในรูป the speech tachometer คือdirect wavefrom coding methode และLPC synthesis methode ขณะที่วิธีทั้งสองแตกต่างกันมาก ทั้งสองวิธีต้องการ ผสมคำศัพท์เป็นขั้นสุดท้าย phonetic speech synthic methode เป็นวิธีสังเคราะห์เสียงที่แท้จริงเท่านั้น เหตุนี้เสียงจาก speech synthesis methode มีคุณภาพแย่ ในความจริงเทคโนโลยีของทุกวันนี้ สามารถสร้าง "syntheticz" speech จาก phoneme synthesizer ได้เสียงคล้ายหุ่นยนต์ ที่ฟังพอเข้าใจได้

เหตุผลที่2 ความหมายของการสังเคราะห์เสียงคือ ความสัมพันธ์กับทฤษฎีพื้นฐานการสุ่มสัญญาณหลักการพื้นฐานของการสุ่มสัญญาณ ความถี่สุ่มต่ำสุดต้องมากกว่า 2 เท่าของความถี่สัญญาณสูงสุด ตัวอย่าง ของทฤษฎีนี้ก็คือ ถ้าต้องการสุ่มสัญญาณเสียงที่ความถี่เสียงสูงสุด 4 KHz ดังนั้นอัตราการสุ่มสัญญาณ ต้องใช้ประมาณ 8 KHz หรือสูงกว่านี้ และการสุ่มแต่ละครั้ง ค่าของสัญญาณหรือ แอมปลิจูดของสัญญาณ จะต้องเก็บเอาไว้ วิธีนี้ต้องการจำนวนบิท 4 บิท แทนแอมปลิจูด ดังนั้นผลก็คือ 4 บิท คูณ 8 KHz (ค่าการสุ่มสัญญาณ) จะได้จำนวนบิทของการสุ่มประมาณ 32,000 บิท/วินาที แต่คำพูดที่พอจะฟังได้ คือสัญญาณเสียงที่มีความถี่ประมาณ 2 KHz นี้จะต้องใช้อัตราการสุ่มสัญญาณ ประมาณ 16,000 บิท/วินาที ดังแสดงใน speech thehometer ในรูป 2 -1

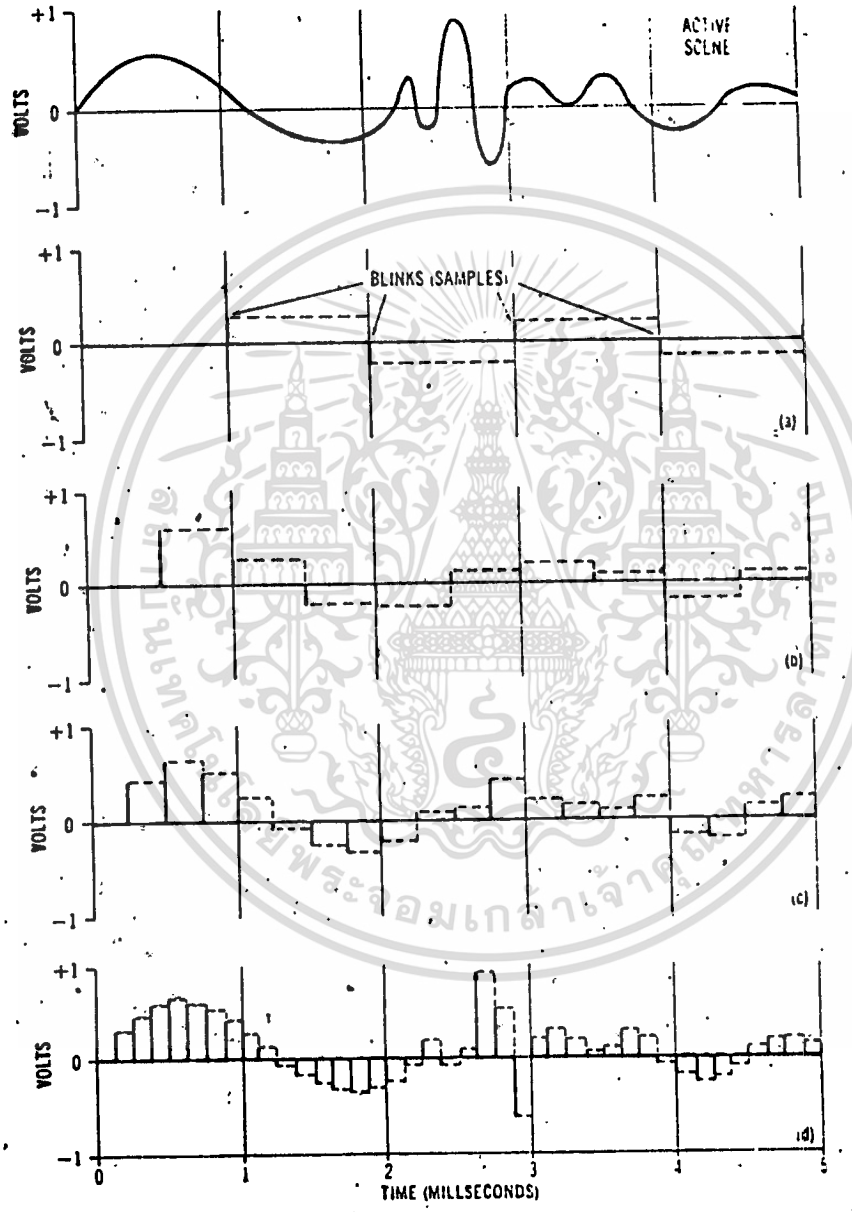
ดังนั้นเราจะสร้างสัญญาณดิจิตอล ขึ้นมาใหม่ที่ลดจำนวน บิท/วินาที ได้อีก วิธีที่สองของการสังเคราะห์เสียง ให้คำจำกัดความว่าphonotic synthesis และLPC synthesis methode เป็นการสังเคราะห์เสียงที่แท้จริง ทั้งสองวิธีใช้จำนวนบิทน้อยกว่า 16,000 บิท/วินาที

The Waveform Encoding/Reconstruction Tecnique

วิธีนี้เป็นวิธีที่ง่ายที่สุดของการกำเหนิดเสียง จากการบันทึกแบบดิจิตอล หลักการคือการสุ่มสัญญาณอนาล็อก สิ่งสำคัญก็คือ จะต้องมีความเข้าใจเทคนิคการสุ่มสัญญาณเพื่อที่จะเข้าใจ encoding/reconstruction เราจะอธิบายให้เข้าใจถึง ทฤษฎีการสุ่มสัญญาณ

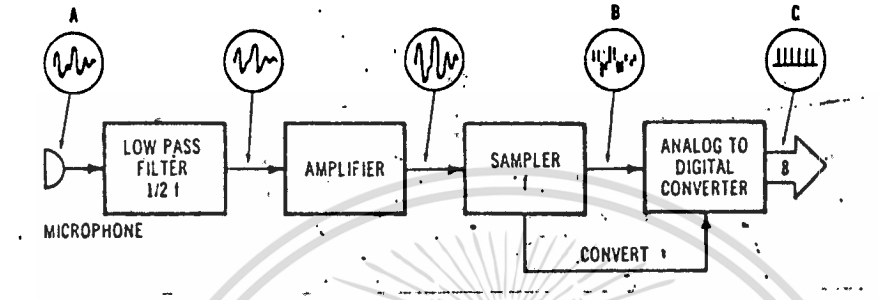
สิ่งแรกสัญญาณเสียง จะถูกเปลี่ยนเป็นสัญญาณอิเลคทรอนิคส์ด้วยไมโครโฟน เวฟฟอรมมีทั้งความถี่สูงและความถี่ต่ำ ถ้าเราสุ่มสัญญาณที่ช้า ผลที่ได้รับที่ดีที่สุด จากเวฟฟอรมที่เคลื่อนที่อย่างช้า ๆ เวฟฟรอมที่มีการเปลี่ยนแปลงอย่างรวดเร็วจะสูญเสียไปจากกการสุ่มของเรา เราจะเริ่มต้นสุ่มสัญญาณด้วยอัตราที่เร็วขึ้นและเร็วขึ้น เราจะเห็นสัญญาณความถี่สูงปรากฏขึ้นที่สัญญาณที่สุ่มได้ พิจารณาการเปรียบเทียบของความถี่การสุ่มจากรูป 2-2

สัญญาณรูปบนสุดแทนรูปคลื่นของอนาล็อก ที่ถูกสุ่มด้วยระบบ ดิจิตอลแซมปลิ้ง ซึ่งประกอบด้วย complex wavefrom เหมือนที่พบในสัญญาณเสียง เราจะพิจราณาสัญญาณส่วนที่เปลี่ยนแปลงอย่างช้าๆ และส่วนที่เปลี่ยนปลงอย่างรวดเร็ว ส่วนที่เปลี่ยนแปลงอย่างช้าๆ จะประกอบไปด้วยความถี่ต่ำ ส่วนที่เปลี่ยนแปลงอย่างเร็ว จะประกอบไปด้วยความถี่สูง ถ้าเราเริ่มต้นสุ่มด้วย 5 millisec ที่อัตราสุ่ม 1sampling/1 mS ผลของสัญญาณ pulse amplitude modulation จะได้ดังรูป 2-2A ซึ่งไม่เหมือนเวฟฟรอมอินพุทเลย ถ้าเราสร้างสัญญาณจาก pulse 5 ลูกให้เป็นสัญญาณเสียง เราก็จะสร้างได้เฉพาะส่วนความถี่ต่ำเท่านั้น ถ้าเราเพิ่มอัตราการสุ่มขึ้นอีกเท่าเป็น 2 sampling/mS เราจะได้สัญญาณที่เริ่มคล้ายสัญญาณอินพุทดังรูป 2-2B ถ้าเราเพิ่มอัตราสุ่มอีก 1 เท่าดังรูป 2-2C และเพิ่มอีกดังรูป 2-2D pulse amplitude modulation signal ที่ได้จากการสุ่มจะมีลักษณะตรงกับความจริง เราจะทำซ้ำในลักษณะนี้ จนกระทั่งได้สัญญาณที่มีความเทียงตรงส่วนที่เป็นความถี่สูงเกิดปรากฏที่ 1/2 mS ทฤษฎี Nyquist sampling เราจะได้รูปคลื่นที่ดีจากอัตราการสุ่ม 4 เท่า/mS ดังรูป 2-2C แต่เวฟฟรอมที่ได้สัญญาณที่ความถี่สูง ก็ยังไม่ได้รับการ modulat ดังนั้นจึงเป็นความถี่การสุ่มเป็น 8 ครั้งต่อ mS ดังรูป 2-2D สัญญาณที่ได้จึงมีรายละเอียดที่ไม่มีการผิดเพี้ยน



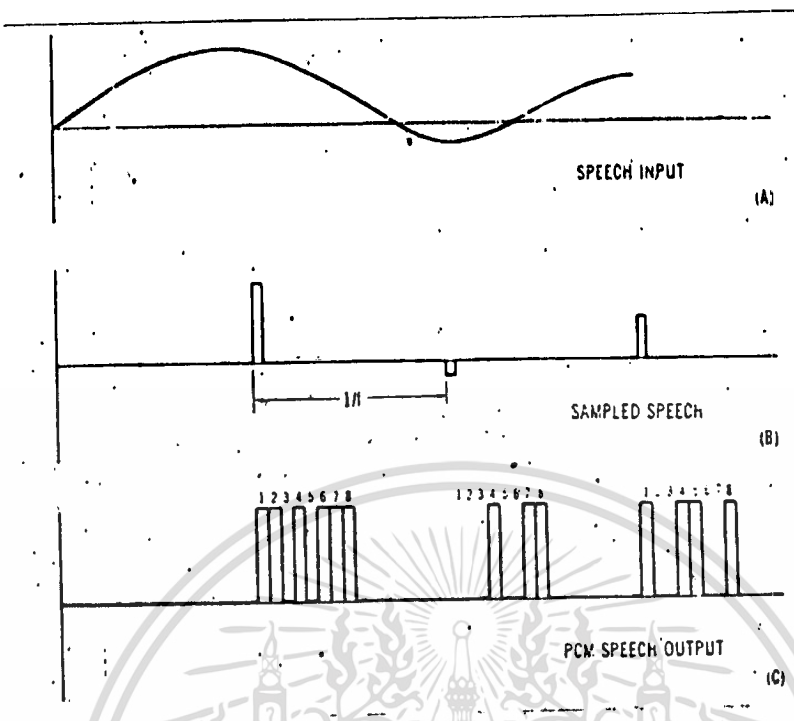
รูป 2-2 แสดงการเพิ่มเข้าด้วยสัญญาณ

กระบวนการสุ่มสัญญาณที่จะแสดงต่อไปนี้ แสดงการสุ่มข้อมูลของ analog pulse ที่ได้ความถี่สูงของสัญญาณเวลาสุ่มและความกว้างของสัญญาณที่สุ่ม แอมปลิจูดของข้อมูลแบบนี้ยากลำบากที่จะเก็บไว้ในหน่วยความจำของเครื่องคอมพิวเตอร์ เพื่อที่จะเก็บค่าของข้อมูลที่สุ่มได้ เราจำเป็นต้องแปลงรูปแบบของข้อมูล ตัวอย่างนี้แสดงในรูป 2-3



การสุ่มสัญญาณเสียงในรูป 2-3 เรานำสัญญาณเสียงจาก ไมโครโฟน และผ่านวงจรกรองความถี่ต่ำผ่าน เพื่อที่จะขจัดความถี่ที่มากกว่า 1/2 ของความถี่การสุ่มสัญญาณ การทำแแบนี้เพื่อป้องกันให้ข้อมูลที่สุ่มได้ มีความถี่น้อยกว่าความถี่การสุ่ม 2 เท่า ภาคต่อไปคือ วงจรยกระดับสัญญาณ การสุ่มสัญญาณก็กระทำเช่นเดียวกันกับรูป 2-2 เอาท์พุทที่ได้จะเป็นกลุ่มของ amplitude modulation pluse ภาคสุดท้ายของระบบคือแปลงสัญญาณอนาล็อกเป็นสัญญาณดิจิตอล จะสังเกตุพบว่าภาคนี้มีอินพุท 1 อินพุท รับสัญญาณอนาล็อกที่ได้จากสุ่ม และมีเอ้าท์พุท 8 เส้น วงจรแปลงอนาล็อกเป็นดิจิตอล มีเอาท์พุทถึง 16 หรือ 18 บิท แต่ในระบบของเราใช้ 8 บิท ก็เพียงพอ เอาท์พุท 8 บิท ที่ได้จะถูกนำไปเป็นอินพุทของคอมพิวเตอร์ ที่จุดนี้ สัญญาณอนาล็อกจะถูกแปรเปลี่ยนไปเป็นสัญญาณดิจิตอล ในลักษณะของ pulse code modulation (PCM) จาก A/D converter

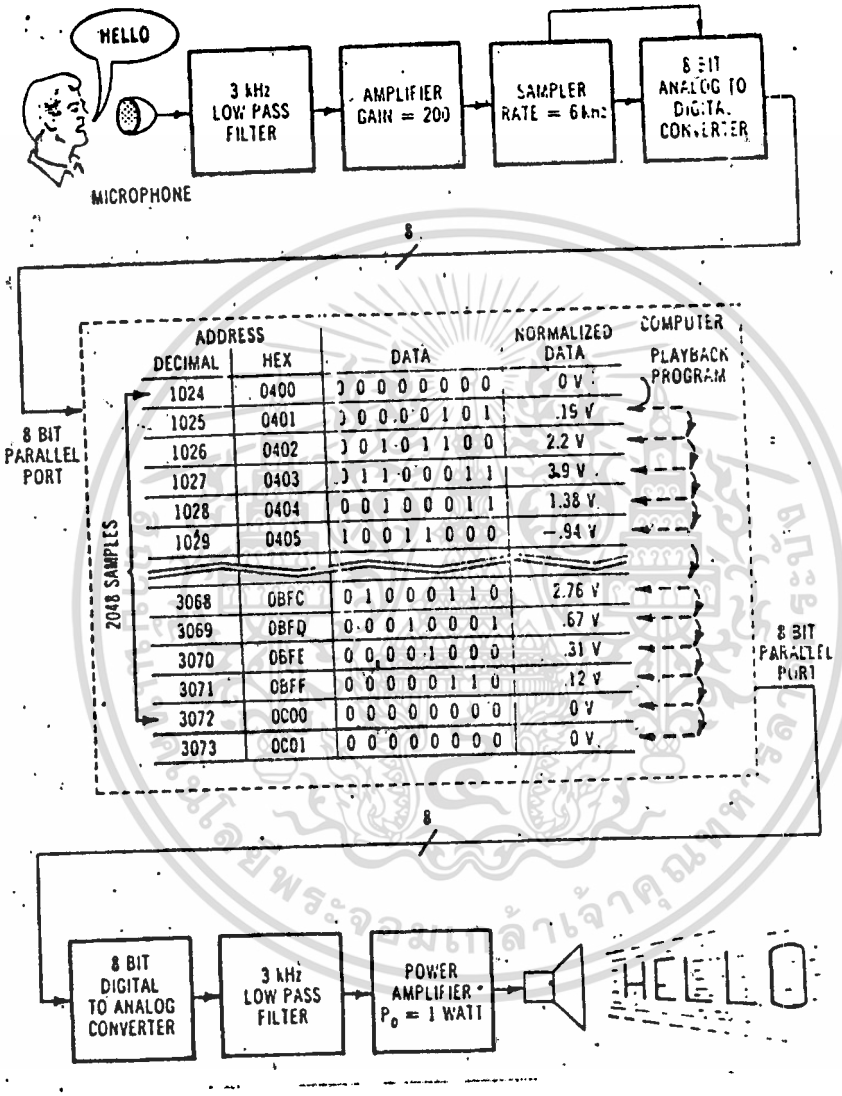
เวฟฟรอมทั้ง 3 รูป ของรูป 2-3 ในรูปย่อย A,B,C นำมาเน้นให้เห็นชัดอีกครั้งในรูป 2-4 เป็นตัวอย่างของ pulse code modulation processนี้เป็นตัวอย่างหนึ่งของการเปลี่ยนสัญญาณเสียงเพื่อให้สามารถเก็บใน computer ได้ ดูจากรูป A ถึงรูป C อินพุทเวฟฟรอมถูกสุ่มแปลงเป็น pulse code modulation ลักษณะสำคัญของรูป 2-4C นั้นไม่มีการเปลี่ยนแปลงของแอมปลิจูด แต่มีความคิดของการเก็บในหน่วยความจำของคอมพิวเตอร์ เอาท์พุทPCM 8 บิท ถูกนำไปเป็นอินพุทแบบอนุกรมหรืออินพุทแบบขนานเก็บในหน่วยความจำ หลังจากข้อมูลเก็บไว้ในหน่วยความจำของคอมพิวเตอร์แล้ว เสมือนว่าได้เก็บแอมปลิจูดของสัญญาณเสียงในคอมพิวเตอร์แล้ว



รูป 2-4 แปลงเอนาล็อกเป็น PCM

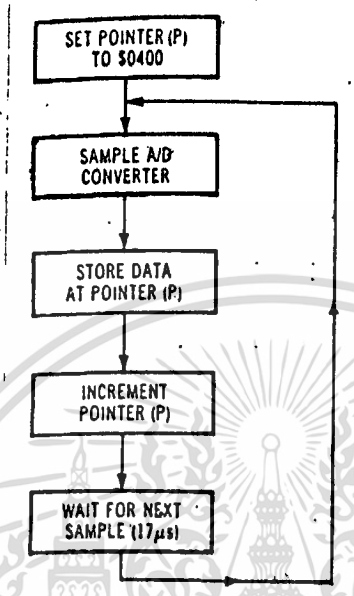
ขณะนี้เรารู้จักวิธีการสุ่มสัญญาณ และเก็บสัญญาณอนาล็อกไว้ในหน่วยความจำของคอมพิวเตอร์แล้ว ต่อไปเราจะต้องทำอะไร ทางหนึ่งที่เป็นไปได้ คือนำสัญญาณที่สุ่มได้แล้วมาสร้างสัญญาณขึ้นมาใหม่ ตัวอย่างต่อไปนี้แสดงถึงการสร้างสัญญาณเสียง สำหรับคอมพิวเตอร์ ดังรูป 2-5 รูปส่วนบนเราจะเห็นว่าเป็นระบบการสุ่มสัญญาณที่เรากล่าวมาแล้ว ซึ่งสุ่มสัญญาณจากไมโครโฟน ด้วยอัตราสุ่ม 6 KHz นี้จะเกิดการสุ่ม 6000 ครั้ง เราเคยกล่าวผ่านมาแล้วว่า คำว่า "hello" ใช้เวลาประมาณ 0.3 s ที่อัตราสุ่ม 6000 ครั้งต่อวินาที คำว่า "hello" จะใช้หน่วยความจำประมาณ 2000 ไบท์ ซึ่งรวมถึงช่วงการเริ่มและหยุดคำพูด บล็อกในรูปกล่างของรูป 2-6 แทนระบบคอมพิวเตอร์ ที่เป็นแบบ 8 บิทอินพุทขนาน และ8 บิทเอ้าพุทพอร์ททั้งสองนี้อาจจะเป็นพอร์ทเดียวกัน ซึ่งเป็นพอร์ทสองทิศทาง

ข้อมูลจะถูกรวบรวมด้วยคอมพิวเตอร์ (แสดงด้วย block) โปรแกรมของคอมพิวเตอร์จะนำข้อมูล 8 บิท ที่ได้จาก A/D converter ไปเก็บไว้ในหน่วยความจำของคอมพิวเตอร์ในรูปนี้แสดงถึงการกินเนื้อที่หน่วยความจำของคำว่า "hello" จาก address 1024 ฐานสิบ ถึงประมาณ 3073 ฐานสิบ ข้อมูลนี้จะเป็นข้อมูลไบนารี่แบบเต็มเสกล 5 โวลท์ คอลัมภ์ "Normalized data" เป็น voltage ที่สมมูลย์ของข้อมูลที่เก็บในหน่วยความจำ



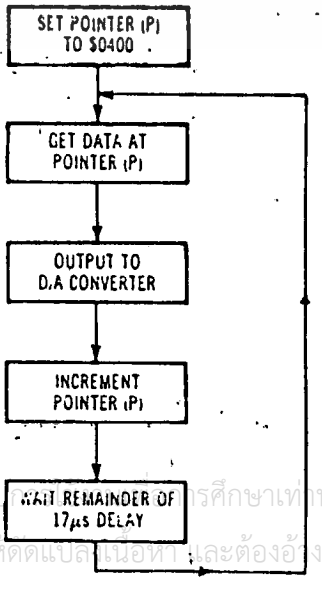
รูปที่ 2-5 WAVE FROM ENCODED SPEECH SUNTHESIS DIAGRAM

ที่จุดนี้แสดงถึงการเสร็จสมบูรณ์ของการเก็บสัญญาณเป็นดิจิตอลซึ่งสามารถเก็บลง disk หรือ cassette tape สำหรับเล่นกลับ



รูป 2-6 แสดงผังโปรแกรมส่วนอินพุท

ส่วนการ playback ข้อมูลที่เก็บไว้จะแทนเป็นส่วนของไดอะแกรม โปรแกรม playback จะถูกเก็บไว้ในคอมพิวเตอร์ โปรแกรมจะนำข้อมูลที่เก็บไว้ออกมาแบบอนุกรมครั้งละ 1 ไบท์ เข้ามายัง 8 บิท digital/analog (D/A) converterที่ถูกออกแบบอย่างเหมาะสม ข้อมูลที่ได้ออกมาจะเหมือนสัญญาณอินพุท นั่นหมายความว่า อินพุทที่เราสุ่มมีเต็มเสกล5 โวลท์ และ 8 บิท d/a จะมีย่าน (range) 5 โวลท์ด้วยเช่นกันเอาท์พุทที่ได้จาก d/a converter จะถูกส่งเข้าวงจรความถี่ต่ำผ่าน 3 KHz เพื่อเอาสัญญาณความถี่สูงที่เกิดขึ้นตอนสุ่มสัญญาณทิ้งไป (ดังรูป2-2D) เอาท์พุทสุดท้ายของวงจรฟิลเตอร์ถูกนำไปขยายด้วยแอมปลิไฟล์ ก่อนเข้าสู่ลำโพง



รูป 2-7 แสดงผังโปรแกรมส่วนเอาท์พุท

สิ่งที่พบในวงจร speech syntheszer ในรูป 2-5 เป็นสัญญาณเสียงที่เก็บในคอมพิวเตอร์และใช้โปรแกรม play back สร้างสัญญาณขึ้นมาใหม่ เพราะว่าในวงจรสุ่มสัญญาณความถี่สูง 6 KHz คอมพิวเตอร์ใช้เวลาประมาณ 17mS ในการเก็บข้อมูล 1 ไบท์ ลงในหน่วยความจำคอมพิวเตอร์มีความเร็วพอที่จะใช้ภาษา เบสิคหรือภาษาสูงอื่น ๆ ได้ ทั้งในการเก็บข้อมูล และ play back ถ้าจะใช้ภาษา assembly ก็ได้ จะเป็นโฟล์ชาดดังรูป 2-6 และรูป 2-7 เป็นกระบวนการง่าย ๆ

ในรูป 2-6 เพื่อให้มีความเหมาะสมของเอาท์พุทกับอินพุท ดังนั้นโปรแกรมทั้งสองจะต้องถูกปรับแต่งให้มีช่วงเวลาของการเก็บข้อมูลและ play back ที่เท่ากัน ถ้า play back มีอัตราสุ่มแตกต่างไปจากการสุ่มสัญญาณ เสียงที่ได้ยินก็จะแตกต่างไปจากเสียงจากแหล่งกำเนิด คล้ายกับการ playing ที่ 45 rpm ของระบบที่บันทึกจาก 33 1/2 rpm

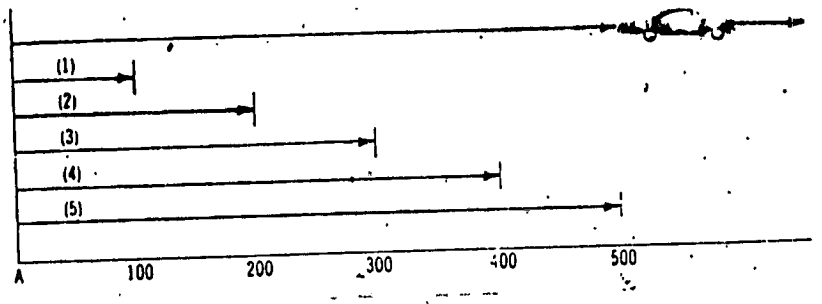
ทางหนึ่งที่จะสร้างเสียงคล้าย sound effect ได้จากคอมพิวเตอร์คือ เปลี่ยนแปลงเวลาหน่วงในบล็อกสุดท้ายของรูป 2-7 ให้ช้าลงหรือเร็วขึ้น ถ้าโปรแกรมวนลูปน้อยกว่า 17 mS แล้วเสียงที่ออกมาจะคล้ายเสียงของ Donald Duck (โดนัลดัก) ถ้าโปรแกรมวนลูป เร็วกว่า 17 mS เสียงที่ได้จะเหมือนกับแผ่นเสียงที่หมุนช้า คุณจะพบผลลัพธ์ได้ตอนเราพยายามปรับการชดเชยเวลาหน่วงของโปรแกรมเพื่อให้ได้เอาท์พุทเหมือนอินพุทปกติ

Delta Modulation

wavefrom encoded/reconstructed speech synthesis ไม่ได้หมายถึงแต่ชนิดแต่กล่าวมาแล้วเท่านั้น ยังมีอีกหลายวิธีที่มีประสิทธิภาพในการจัดเก็บสัญญาณเสียง วิธีหนึ่งซึ่งมีประสิทธิภาพในงาน telephone industry เรียกว่า Delta modulation ความแตกต่างของวิธีนี้กับในรูป 2-5 คือวิธีนี้การเปลี่ยนแปลงของแอมปลิจูด เป็นแบบค่าสมบูรณ์ ถ้าเรากลับไปกล่าวถึงเรื่องรถยนต์ที่เดินทางไกลอีกครั้ง เราจะเข้าใจเทคนิค delta modulation ดีขึ้น เราจะอธิบายได้ด้วยการเดินทางจากจุด A ถึงจุด B ดังนี้

1. รถเคลื่อนที่ได้ระยะทาง 100 ฟุต จากจุด A
2. " " 200 " "
3. " " 300 " "
4. " " 400 " "
5. " " 500 " "

จะเห็นว่าเราบอกระยะทางการเคลื่อนที่ของรถยนต์ที่เวลาหนึ่งด้วยระยะทางห่างจากจุดเริ่มต้นเสมอแบบนี้เป็นแบบของการสุ่มค่าสมบูรณ์ ดังแสดงแบบการเดินทางในรูป 2-8 ถ้าเราแทนการเดินทางเวฟฟอรม จะได้ระยะทางแทนแรงดันสัญญาณ



รูป 2-8 ตัวอย่างการสุ่มแบบสม่ำเสมอ

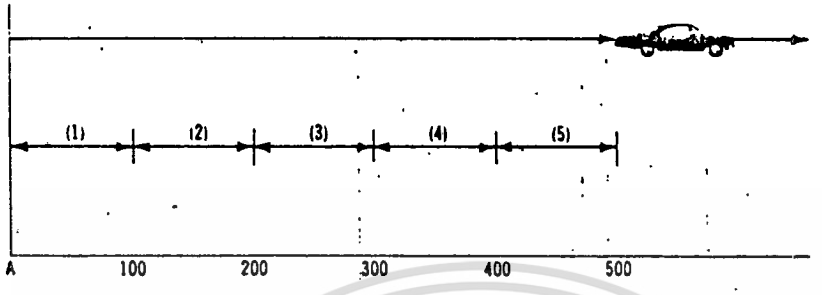
การสุ่มสัญญาณของแบบ delta modulation เป็นแบบสนใจเฉพาะส่วนที่มีการเปลี่ยนแปลงไปอย่างมากหรือที่น่าสนใจเท่านั้น ดังนั้นคุณภาพข้อมูลจะลดลง เราจะแยกสัญญาณที่ซ้ำซากกันออกไป (สัญญาณติดกันที่เหมือนกัน) ก็จะมีประสิทธิภาพในการเก็บข้อมูลได้มากขึ้น แบบเดียวกับการเคลื่อนที่ของรถยนต์ต่อไปนี้แทนแบบ delta modulation

1. รถยนต์เดินทางได้ 100 ฟุต จากจุด A
2. " " 100 " จากจุดที่เช็คครั้งสุดท้าย
3. " " 100 " " "
4. " " 100 " " "
5. " " 100 " " "

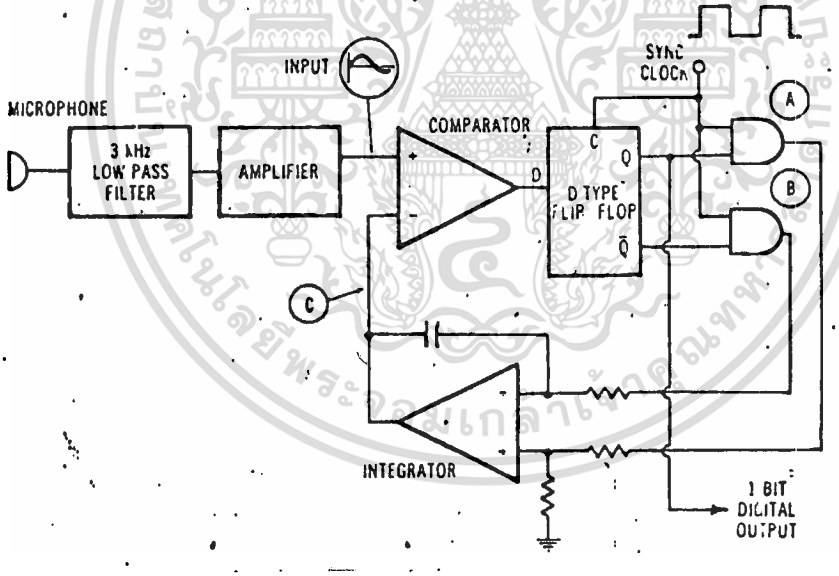
ขณะนี้สมมุติว่าเราเดินทาง 1000 ไมล์ โดยการใช้การ sampling ดังรูป 2-9 เราแบ่งย่านออกเป็น 500,000 พิต หรือมากกว่า ที่จุดสุ่มเราจะรู้ระยะทางได้จากความสัมพันธ์กับจุดเริ่มต้น (จุด A) ตัวอย่าง deta modulation ในรูป 2-9 อธิบายการเดินทางในเทอมที่ระยะแตกต่างกัน (หรือ deta) ระหว่างการสุ่ม แต่ละครั้ง ดังนั้นแม้จะเดินทางถึง 100 ไมล์ แต่จำนวนการเดินทางที่ไกลสุดแต่ละครั้งไม่เกิน100 พิต ประมาณ 500,000 ครั้ง นั่นก็คือ ประสิทธิภาพของวิธีนี้เพิ่มขึ้น

ขบวนการ deta modulation เปรียบได้เหมือนกับ incremental encoding methods เพราะจะ encode เฉพาะช่วงที่สุ่มสัญญาณ แล้วมีการเปลี่ยนแปลงของสัญญาณอินพุท แต่เราจะทำลักษณะนี้กับสัญญาณอิเล็กทรอนิกส์จากไมโครโฟนได้อย่างไร? วิธีหนึ่งคือ digital delta modulation ดังแสดงในรูป 2-10 จากไดอะแกรมแสดงถึงสัญญาณเสียงรับได้จากไมโครโฟน ผ่านวงจรฟิลเตอร์และแอมปลิไฟเออร์หลังจากนั้นจึงแปลงสัญญาณเป็น delta modulation ด้วยวงจรง่าย ๆ ที่ประกอบด้วยอนาล็อกออปแอมป์กับส่วนการป้อนกลับ ที่ประกอบด้วยวงจรanalog integrator , loop ของ

D flip-flop ซึ่งจัดให้สัญญาณ delta ถูก synchronize (หรือการเปลี่ยน slope) กับ digitizing clock rate



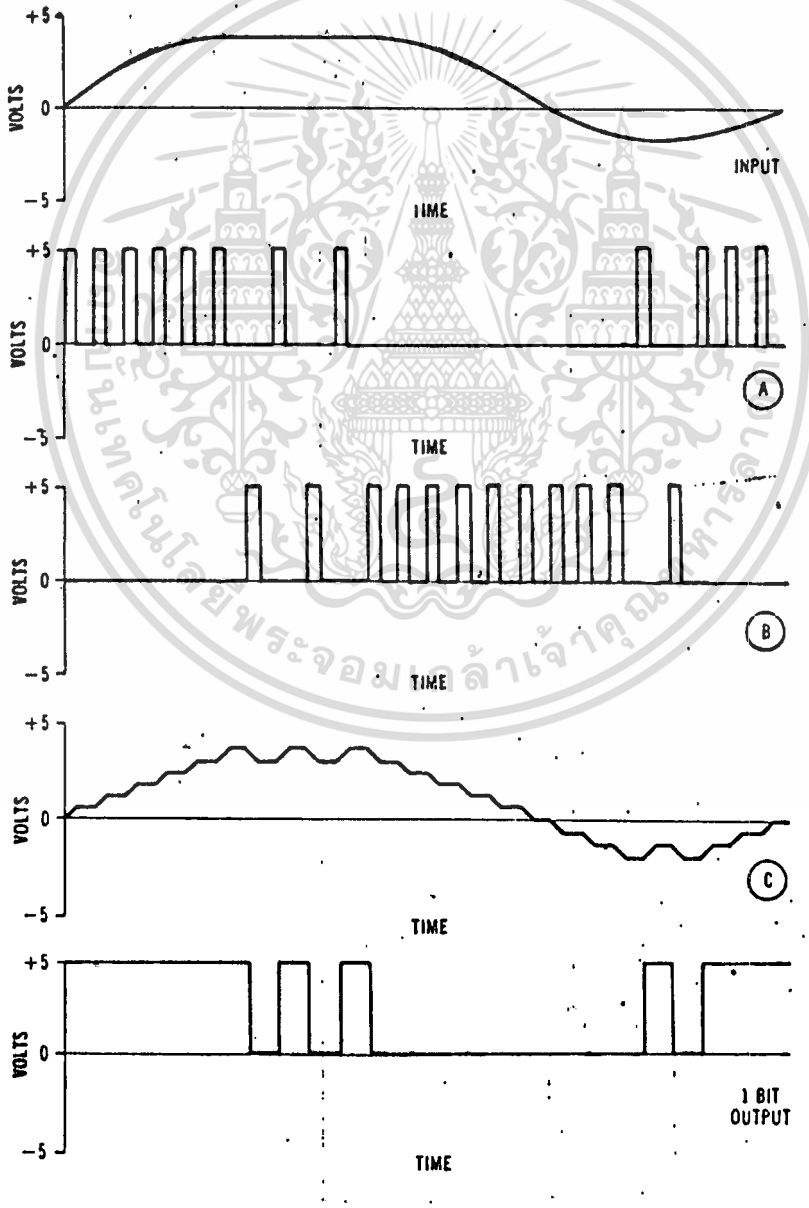
รูป 2-9 แสดงตัวอย่างของ delta modulation encoding



รูป 2-10 electronic delta encoding method

ให้ดูสัญญาณที่เกิดขึ้นระหว่างขบวนการdelta encoding ในรูป 2-10 สังเกตุที่รูปคลื่นแรกที่เป็นสัญญาณอินพุท เป็นรูปคลื่นที่ oscillating ซึ่งมีทั้ง rising slope และ folling slope และค่าบเวลาของสัญญาณที่เป็น signal stability เอาท์พุทของ D-flip-flop ในรูป 2-10 แทนด้วยสัญญาณ A และ B ในรูป 2-11 ซึ่งเป็นพัลส์ขนาด 5 โวลท์ ถูกส่งให้วงจร Integrator มีสัญญาณเอ้าพุท C เมื่อมีการ

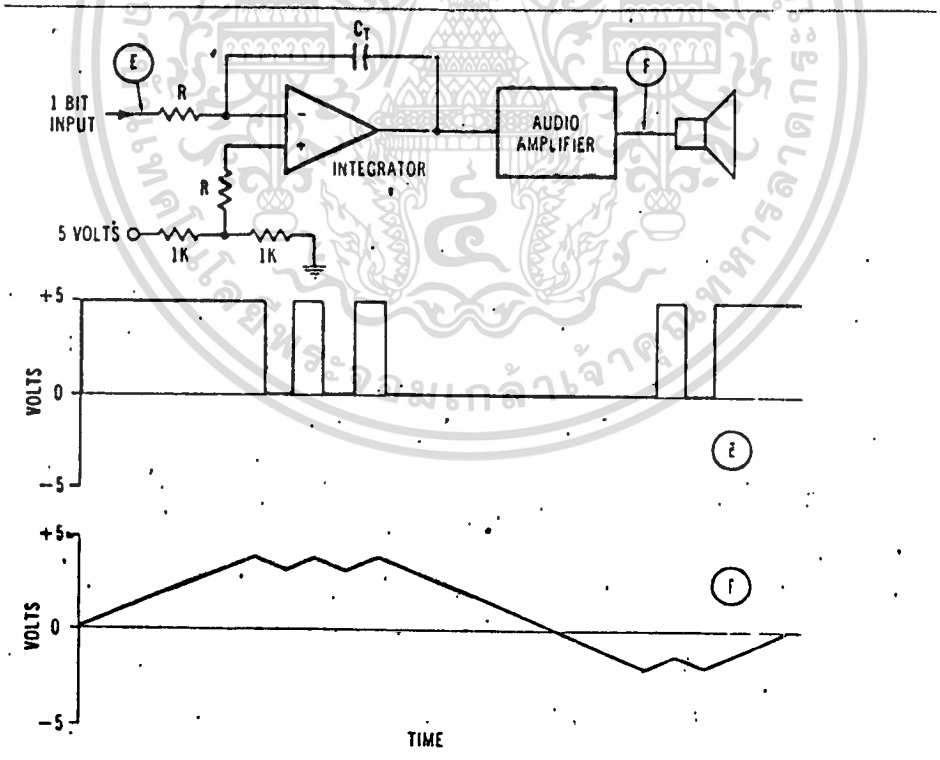
เปลี่ยนแปลงของสัญญาณอินพุทที่สุ่ม จะเกิดรูปคลื่นที่มีคาบเวลา T ในรูป A ถ้าสัญญาณอินพุทค่อย ๆ เพิ่มขึ้น พัลส์ที่เกิดขึ้นที่จุด A แสดงถึงการที่สัญญาณอินพุทเพิ่มขึ้นรูปคลื่น ที่จุด B แสดงถึงสัญญาณอินพุทมีการเปลี่ยนแปลงที่ลดลง (falling input signal) ช่วงสัญญาณที่อินพุทมีค่าคงที่ (fixed valve) ดังนั้น delta modulation จะทำให้เอาท์พุทพัลส์ที่จุด A,B oscillation สลับกันไปมา สัญญาณเอาท์พุทที่จุด C จะถูกป้อนกลับกลับมาเปรียบเทียบกับสัญญาณอินพุท จากวงจร integrator รูปคลื่นสุดท้ายในรูป2-11



รูป 2-11 สัญญาณของ delta encoding

เป็นสัญญาณดิจิตอลเอาท์พุท ไปยังเครื่องคอมพิวเตอร์ บิทสัญญาณนี้ถูกเก็บในหน่วยความจำของคอมพิวเตอร์ ด้วยอัตราเดียวกับอัตราสุ่มสัญญาณ ในdelta encoder ในรูป 2-10

การสร้างสัญญาณกลับมา (Reconstruction) ของ delta encoed ต้องการวงจรที่มี hardware ง่าย ๆ ที่สามารถผลิตสัญญาณเสียงได้ ดังไดอะแกรมในรูป 2-12 เป็นวงจร simple integrator ที่คล้ายกับที่ใช้ในขบวนการ encoding ที่อินพุท E ไปยัง integrator เป็นแบบบิทสัญญาณจากเครื่องคอมพิวเตอร์ที่อัตราเดียวกับการสุ่มสัญญาณ ค่าเวลาวงจร integrator นี้ ได้จากผลคูณของ RC ซึ่งต้อง match กับวงจร integrator ในระบบการสุ่มสัญญาณ ถ้าอินพุทที่เข้ามาที่จุด E มีค่าลอจิกเป็น 1 วงจร integrator จะสร้าง constance slope ในทิศทางหนึ่ง ถ้าอินพุทเปลี่ยนเป็นø วงจร integerator นี้จะให้เอาท์พุทที่ slpoe กลับขั้วกับโลจิค 1 ผลอันนี้จะได้สัญญาณที่สมมูลย์ดังจุด F เวฟฟอรมดังรูป 2-12 ซึ่งเป็นผลมาจากสัญญาณอินพุทผ่าน delta modulation เราจะเห็นว่ารูปคลื่นนี้ดีพอใช้ มีการผิดเพี้ยน (distrotion)

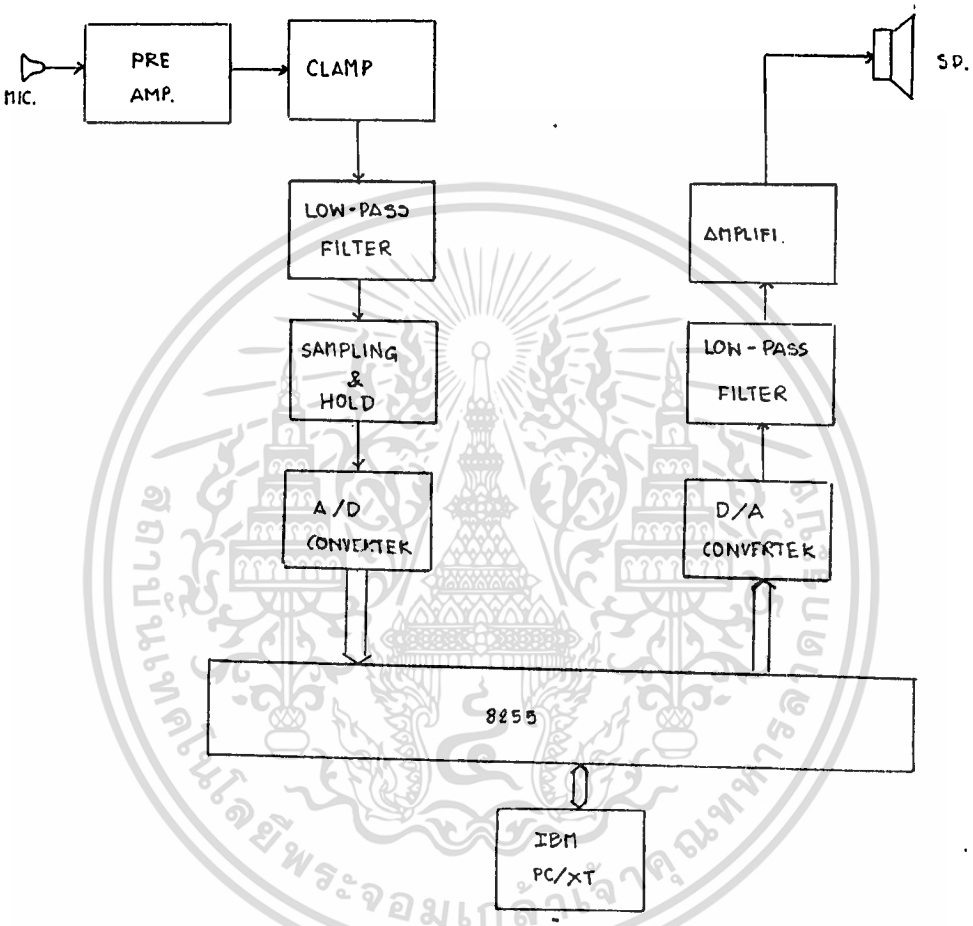


รูป 2-12 Delta mooding speech playback circuit

เล็กน้อย distrotion นี้เกิดขึ้นระหว่าง encoding หรือที่เรียกว่าslope overload ถ้าอินพุท slope เปลี่ยนแปลงอย่างรวดเร็วกว่าIntegratorจะตามสัญญาณได้ทัน(trak) เอาท์พุทที่ได้จะไม่เหมือนกับอินพุท ปัญหานี้เป็นธรรมดาของวงจรLinear delta modulation การแก้ปัญหาทำได้โดยใช้เทคนิค adaptive chip รวมทั้ง National digital speech synthih system (เครื่องหมายการค้าของบริษัท National semiconductor) ระบบADPCM คล้ายกับ linear system ต่างกันที่ ไม่เก็บslope ที่เปลี่ยนแปลงครั้งสุดท้ายเท่านั้น แต่ยังเก็บจำนวนครั้งของการเปลี่ยนแปลง วิธีนี้จึงมีเป็น ปัญหาslope overlode จะใช้ 4 บิทแทนการเปลี่ยนแปลงของ slope ที่ขอบขาขึ้น หรือขาลง

ชุดเก็บสัญญาณเสียง

ชุดเก็บสัญญาณเสียงมีโครงสร้างดังรูป



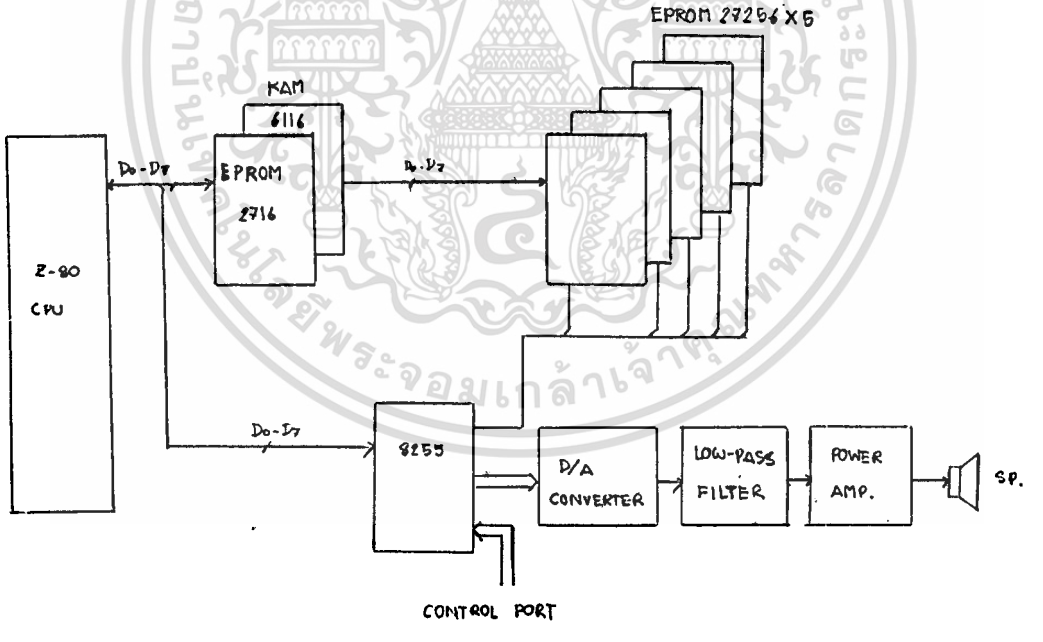
รูป 2-13 บล็อกไดอะแกรมของชุดเก็บเสียง

จากรูปเครื่องคอมพิวเตอร์เริ่มทำงาน จะทำโปรแกรมพอร์ท 8255 เราจะเก็บสัญญาณเสียงจากโมโครโฟน มีสัญญาณเอ้าพุทออกเพียง 2-3 mV. จึงต้องขยายสัญญาณเพิ่ม 300 เท่า เพื่อให้ได้สัญญาณจริงประมาณ 4 V. สัญญาณจากปรีแอมป์เป็นสัญญาณที่แกว่งในช่วงบวกและลบ เราจะยกระดับสัญญาณให้แกว่งในช่วงบวกเท่านั้น โดยนำสัญญาณมาผ่านวงจรแคลมป์ เอ้าพุทที่ได้จะนำมาผ่านวงจรโลพาส ฟิลเตอร์ เพื่อตัดความถี่สูงออกไปก่อนเข้าวงจรสุ่มสัญญาณ เพื่อทำการสุ่มแอมปลิจูดของสัญญาณ และใช้วงจรA/D แปลงสัญญาณอนาล็อกเป็นสัญญาณดิจิตอล ก่อนเก็บไว้ในหน่วยความจำของเครื่องไมโครคอมพิวเตอร์IBM PC/XT สัญญาณเสียงที่เก็บไว้ในไมโครคอมพิวเตอร์ จะถูกนำออกมาใช้โดยผ่านวงจรD/A เพื่อแปลงสัญญาณดิจิตอลเป็นสัญญาณอนาล็อก แล้วจึงนำมาผ่านวงจรlow pass filter

เพื่อกำจัดความถี่สูงที่เกิดจากA/D ที่ไม่ต้องการออกไป และนำมาผ่านวงจรขยายก่อนเข้าลำโพง สัญญาณเสียงที่เก็บอยู่ในหน่วยความจำของ IBM PC/XT เราจะเก็บไว้เป็นไฟล์ข้อมูล เพื่อที่เราจะได้ตัดต่อสัญญาณเสียงออกเป็นคำๆ และทราบตำแหน่งของคำ และจำนวนไบท์ของคำ เพื่อว่าเมื่อเราเบรินสัญญาณเสียงลง EPROMเราจะได้นำเอาข้อมูลมาใช้กับบอดร์สร้างเสียงพูด

บอดร์สร้างเสียงพูดนี้จะใช้ Z-80 CPU ในการจัดการสัญญาณเสียง โดยมีพอร์ทแบบ hand shank ในการติดต่อเมนบอร์ทเพื่อส่ง codeและ data ที่สั่งให้บอดร์นี้สร้างเสียงพูดตามที่ต้องการ เช่น ไฟเลี้ยวซ้ายเปิด, ความดันเกิน ฯ

บอดร์นี้ประกอบด้วย ROM เก็บเสียงพูดขนาด 32KByte เบอร์27256 จำนวน 5 ตัว โดยใช้ 8255 เป็นตัวDecode ROM ที่เก็บเสียง และเป็นพอร์ทแบบhand shack โดยมีโปรแกรม monitor เก็บอยู่ใน ROM 2716 ขนาด 2 KByte ดังแสดงในบล็อกไดอะแกรม



รูป 2-14 บล็อกไดอะแกรมของบอร์ดสร้างเสียง Z-80

บทที่ 3 รายละเอียดของวงจร

แอคทีฟโลพาสฟิลเตอร์(Ative Low-pass fillter)

บัตเตอร์เวิดส์ด(Butterword)

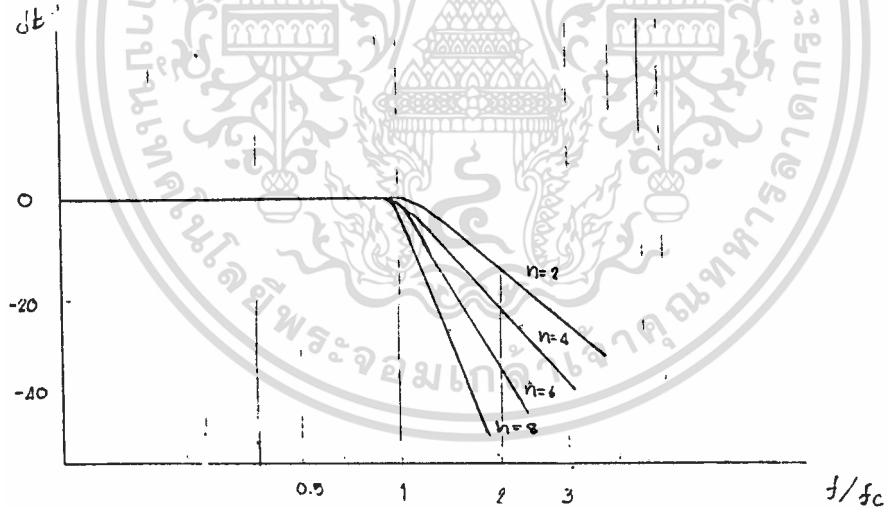
บัตเตอร์เวิส์ดฟิลเตอร์ให้ผลตอบสนองของแอมปลิจูดราบเรียบมากที่สุด ฟิลเตอร์นี้ใช้สำหรับกรองสัญญาณในระบบจัดการข้อมูล เพื่อป้องกันข้อผิดพลาดในการสุ่มสัญญาณ และจุดประสงค์ทั่ว ๆ ไป ในวงจรกรองสัญญาณความถี่ต่ำผ่าน

ความถี่คัทออฟ fc เป็นความถี่ซึ่งแอมปลิจูดของสัญญาณลดลง 3 dB อัตราการตัดทอนของความถี่ถัดไปคือ -n6 dB ต่อออปเตทของความถี่ของสัญญาณซึ่งวงจรมี n อันดับ(จำนวนของโพล)

คุณลักษณะของวงจรบัตเตอร์เวิส์ด:

-ผลตอบสนองของแอมปลิจูดราบเรียบที่สุด

-เกณฑ์การขยายของวงจรเที่ยงตรงมากตั้งแต่ความถี่ต่ำสุดถึงความถี่ปลายสุดของย่านความถี่



รูป3-1 ผลตอบสนองของวงจร

ตารางข้างล่างนี้แสดงถึงการเกิดโอเวอร์ชูท และเซ็ทติ้งไทม์ของวงจรกรองความถี่ต่ำผ่านต่อสัญญาณอินพุทแบบเสตป

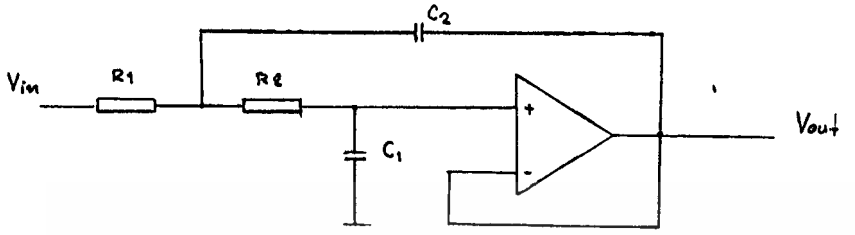
ตาราง 3-1

NUMBER OF POLE	PEAK OVERSHOOT	SETTING TIME (% OF SIGNAL VALUE)		
	% OVERSHOOT	± 1 %	± 0.1%	± 0.01%
2	0	1.1/fc sec	1.7/fc sec	1.9/fc sec
4	11	1.7/fc	2.8/fc	3.8/fc
6	14	2.0/fc	3.9/fc	5.0/fc
8	16	3.1/fc	5.1/fc	7.1/fc

การออกแบบวงจรความถี่ต่ำผ่านแบบอันดับสอง

(เกณฑ์การขยายของออปแอมป์เท่ากับ 1)

รูปโครงสร้างของวงจร



$$\frac{V_o}{V_i} = \frac{1}{1 + 2\xi\frac{s}{\omega_c} + \frac{s^2}{\omega_c^2}}$$

$\omega_c = 2\pi f_c$

f_c = cut off frequency

ξ = damping factor

รูปที่ 3-2 วงจรโลพาสฟิลเตอร์แบบบัตเตอร์เวิร์ท

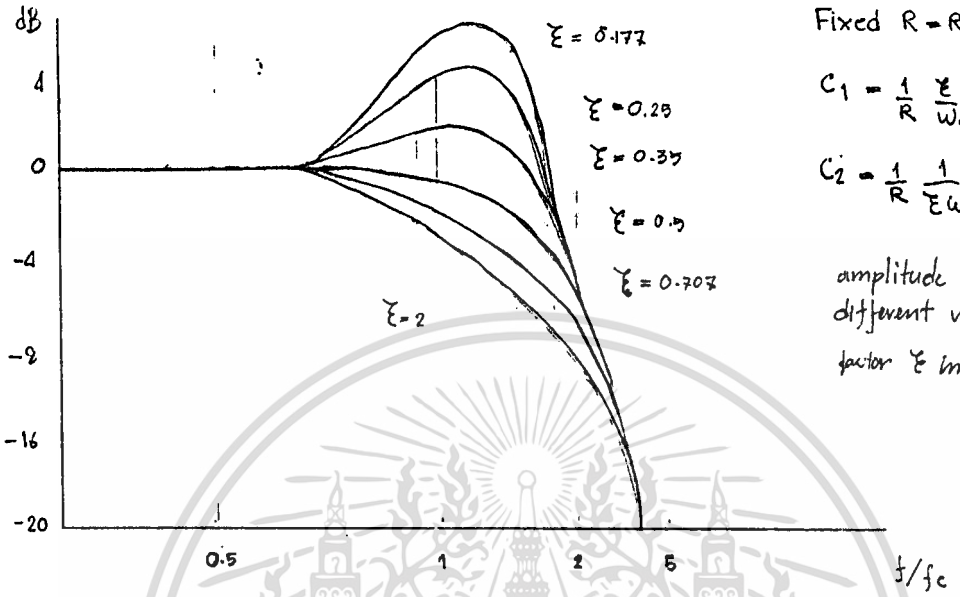
ค่าพารามิเตอร์เหล่านี้เป็นตัวกำหนดคุณสมบัติของความถี่และเฟส ที่ได้รับจากวงจรกรองความถี่ต่ำแบบอันดับสอง: อัตราขยาย(Gv), แดมปิ้งแฟกเตอร์(ξ) หรือค่า Q ($Q=2\xi)^{-1}$) และความถี่คัทออฟ

อันดับสูงสุดของการตอบสนองที่ได้รับจากวงจรอันดับสอง ประกอบด้วยส่วนของ RC แบบง่ายๆ ที่ประกอบกันเป็นวงจรฟิลเตอร์ การเลือกค่า (หรือ Q แฟกเตอร์) ได้จากผลตอบสนองของวงจรฟิลเตอร์(ดูในตาราง)

FILTER RESPONCE	ξ	Q	CUTOFF FREQUENCY f_c
BUTTERWORTH	$\frac{\sqrt{2}}{2}$	$\frac{1}{\sqrt{2}}$	FREQUENCY OF WHICH $G_V = -3$ dB

ตาราง 3-2

ผลตอบสนองของฟิลเตอร์ และแดมปิงแฟกเตอร์



Fixed $R = R_1 = R_2$ (Fig 3-2)

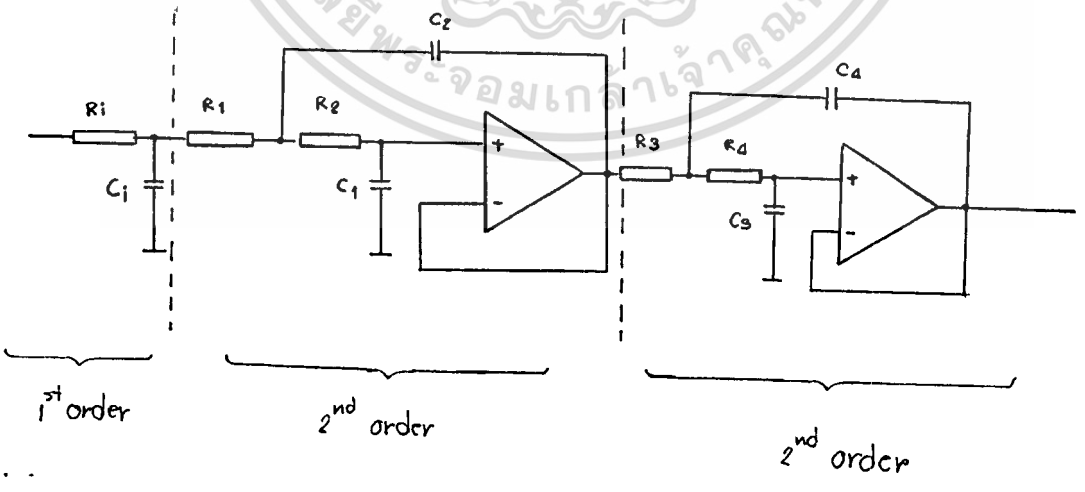
$C_1 = \frac{1}{R}\frac{\xi}{\omega_c}$

$C_2 = \frac{1}{R}\frac{1}{\xi\omega_c}$

amplitude responce for different value of damping factor ξ in 2nd order filter

ตัวอย่างของวงจรกรองความถี่ต่ำผ่านอันดับ 5 (บัสเตอร์เวิร์ธ) มีค่าเกณฑ์เท่ากับหนึ่ง

(รูป)



รูปที่ 3-3 วงจรความถี่ต่ำผ่านอันดับ 5

ในวงจรถ้าต้องการความถี่คัทออฟ fc= 3.4 kHz และ $R_i = R_1 = R_2 = R_3 = 10k$ เราจะหาค่า C ได้ดังนี้

$C_i = 1.354 \cdot \frac{1}{R} \cdot \frac{1}{2\Pi fc} = 6.33\ nF$

$C_1 = 0.421 \cdot \frac{1}{R} \cdot \frac{1}{2\Pi fc} = 1.97\ nF$

$C_2 = 1.753 \cdot \frac{1}{R} \cdot \frac{1}{2\Pi fc} = 8.20\ nF$

$C_3 = 0.309 \cdot \frac{1}{R} \cdot \frac{1}{2\Pi fc} = 1.45\ nF$

$C_4 = 3.325 \cdot \frac{1}{R} \cdot \frac{1}{2\Pi fc} = 15.14\ nF$

The attenuator of the filter is 30 dB. at 6.8 kHz and better than 60dB. at 15 kHz

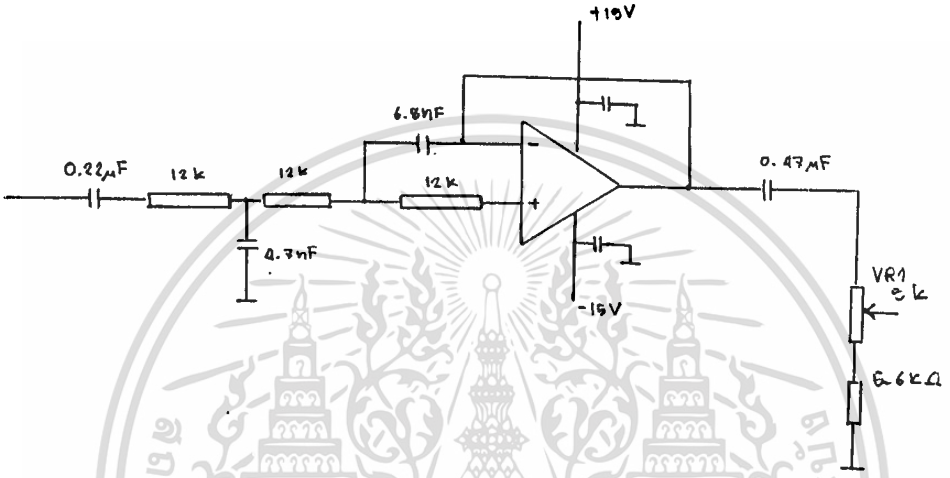
ค่าการตัดทอนของวงจรฟิลเตอร์นี้มีค่า 30 dB ที่ 6.8 kHz และดีกว่า 60 dB ที่ 15 kHz

ตารางที่ 3-3
แดมปิ้งแฟกเตอร์สำหรับวงจรกรองความถี่ต่ำบัสเตอร์เวิส์ด

Order	C_i	C_1	C_2	C_3	C_4	C_5	C_6	C_7	C_8
1		0.707	1.41						
3	1.392	0.202	3.54						
4		0.92	1.09	0.38	2.61				
5	1.354	0.421	1.75	0.309	3.235				
6		0.966	1.035	0.707	1.414	0.259	3.86		
7	1.336	0.488	1.53	0.623	1.604	0.222	4.49		
8		0.98	1.02	0.83	1.20	0.556	1.80	0.195	5.125

วงจรโลพาสฟิลเตอร์ 2.8 kHz อันดับ 3

(วงจร)



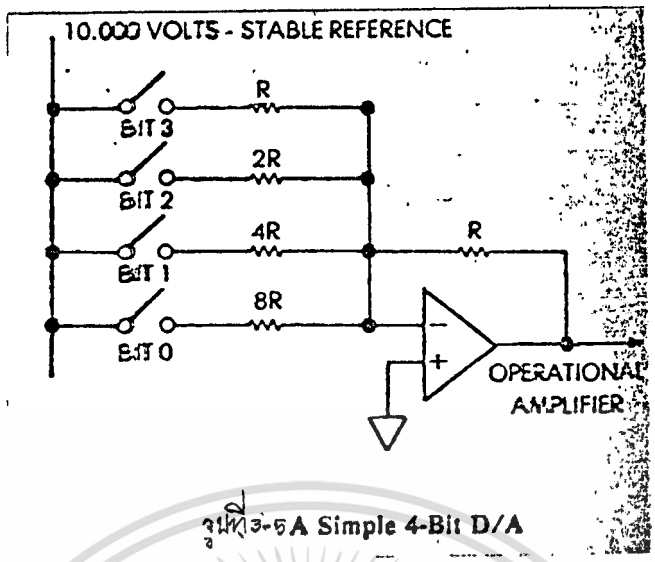
รูปที่ 3-4 วงจรโลพาสฟิลเตอร์ 2.8 kHz

ส่วนการแปลงสัญญาณดิจิตอลเป็นสัญญาณอนาล็อก

การเปลี่ยนจำนวนไบนารีให้เป็นอนาล็อกโวลท์เตจ โดยวิธีการง่ายๆ ดังต่อไปนี้ โวลท์เตจสร้างมาจากตำแหน่งบิทของแต่ละจำนวนไบนารี ค่าโวลท์เตจเป็นสัดส่วนกับค่าประจำหลักของแต่ละบิท

สำหรับตัวอย่างนี้ บิท0 จะสร้างโวลท์เตจ $V(2^0)$: บิท1 สร้างโวลท์เตจ $2V(2^1)$: บิท2 จะสร้างโวลท์เตจ $4V(2^2)$: และบิท n ละสร้างโวลท์เตจ $2^n * V$ ผลลัพท์ของโวลท์เตจนี้เป็นสัดส่วนกับจำนวนไบนารี

. วงจร4 บิท D/A แบบง่ายๆ ในรูป ประกอบด้วยสวิทช์ 4 ตัว, รีซิสเตอร์สำหรับบวกค่าแรงดัน 4 ตัว, ออปเปอร์เรชั่นแลแอมปลิไฟล์, ฟีดแบ็ครีซิสเตอร์ ค่าของรีซิสเตอร์เป็นอัตราส่วน 1,2,4,8 ซึ่งทำให้เกิดเกณฑ์การขยาย-⅛,-¼ , -½ และ -1 ตามฟังชั่นของวงจร

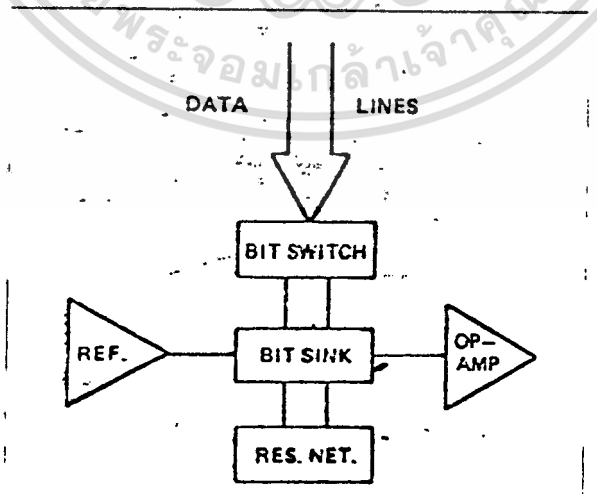


รูปที่ 3-5 A Simple 4-Bit D/A

สมมุติเริ่มต้นสวิทช์ทั้งหมดอยู่ในตำแหน่งเปิดวงจร ก็คือไม่มีอินพุทให้แก่ออปแอมป์ เอ้าพุทจะเป็น "0" เลือกบิทสวิทช์ 0 ติอกับไฟ -10 V. เป็นอินพุทของออปแอมป์ผ่านทางรีซิสเตอร์ 8R ทำให้เกิดเอ้าพุท 1.25 V. เลือกสวิทช์บิท 1 จะเป็นการบวกค่าแรงดันเข้ากับค่า 1.25 V. เอ้าพุทจะได้ 3.75 V. ถ้าสวิทช์ทั้งหมดปิด ผลลัพท์ของเอ้าพุทคือ 10+5+2.5+1.25 = 18.75 V.

วงจรในทางปฏิบัติของ D/A

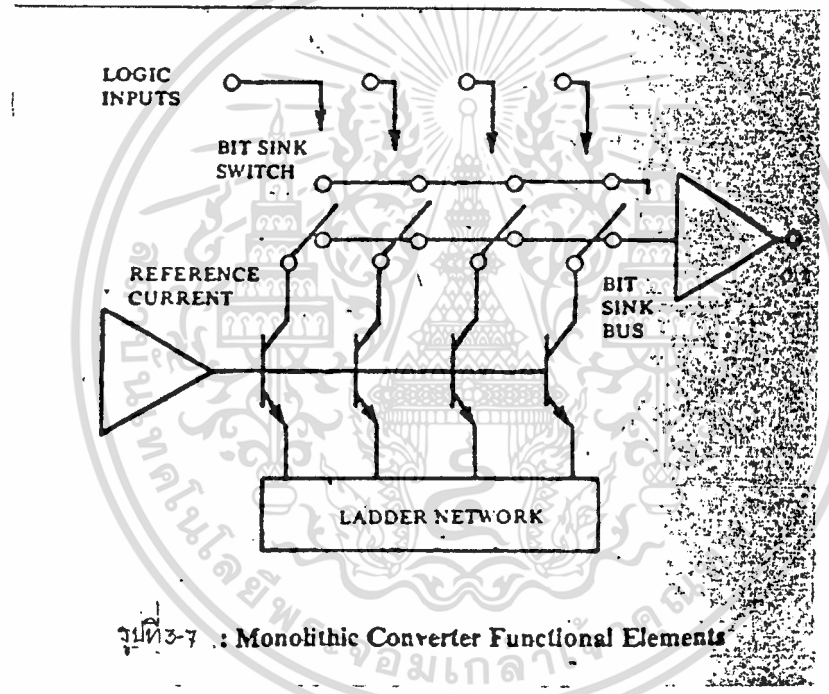
การออกแบบทางปฏิบัติ ในรูป 3-6 เป็นตัวอย่างที่ออกแบบในวงจรรวม D/A ในทางปฏิบัติจะใช้การรวมของกระแสแทนแรงดัน ทำให้การสวิทช์ออน และออฟ ทำได้ง่ายและเที่ยงตรง เพื่อให้ได้แรงดันเอ้าพุทในส่วนสุดท้ายจะใช้วงจรแปลงกระแสเป็นแรงดัน ซึ่งทำได้ง่ายๆ โดยใช้ออปแอมป์



รูปที่ 3-6 A Practical Converter Diagram

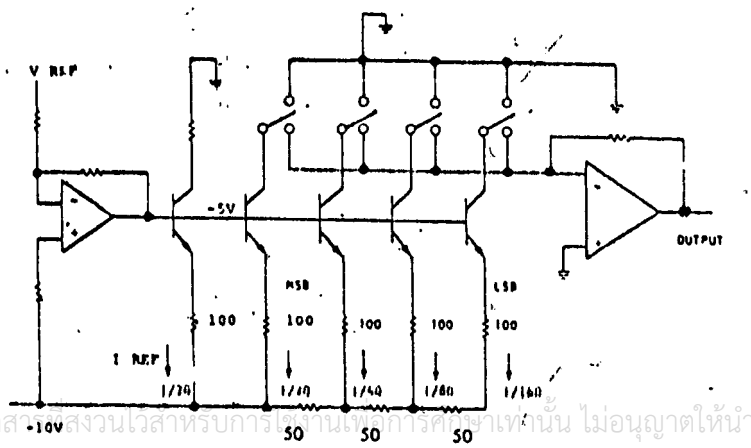
รูป3-7 เป็นตัวอย่างของการแปลงสัญญาณ ประกอบด้วย: แหล่งจ่ายกระแสอ้างอิง, บิท-ซิ้งค์ทรานซิสเตอร์, แลดเดอร์รีซิสเตอร์เน็กเวิค, บิทซิ้งค์สวิทช์ และส่วนการแปลงกระแสเป็นแรงดัน

กระแสบิท-ซิ้งค์ ได้จากแหล่งจ่ายกระแสที่เสถียร ซึ่งจ่ายกระแสซอส เป็นสัดส่วนกับกระแสอ้างอิงเหล่านี้ กระแสของบิท-ซิ้ง ทรานซิสเตอร์แต่ละตัวได้จากส่วนของ R-2R แลดเดอร์-รีซิสเตอร์เน็กเวิค เน็กเวิค R-2R จะทำให้เกิดกระแส 2^n ตามลำดับ ไหลผ่าน บิท-ซิ้ง ทรานซิสเตอร์ สวิทช์นี้จะทำให้กระแสไหลผ่านเข้า บิท-ซิ้ง บัส ซึ่งต่อกับกระแสเป็นแรงดัน หรือต่อลงกราวด์ ในตัวอย่างนี้มีกระแสคือ $\frac{1}{2}$, $\frac{1}{4}$, $\frac{1}{8}$, และ $\frac{1}{16}$ แอมแปร์



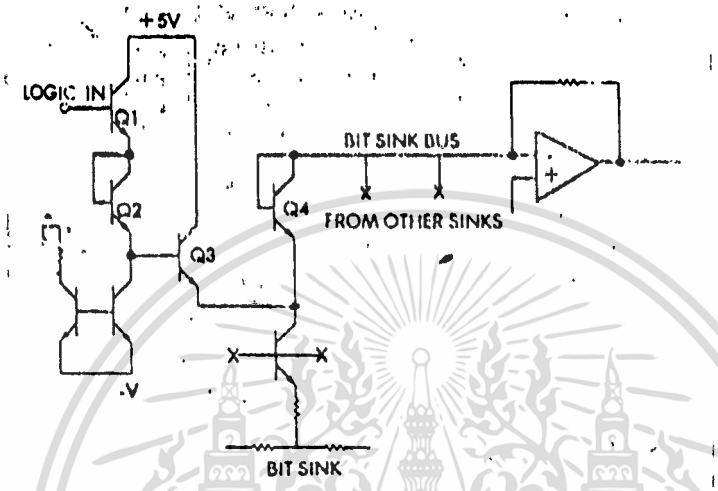
รูปที่3-7 : Monolithic Converter Functional Elements

ในวงจรโมโนลิติค จริงๆ ใช้ทรานซิสเตอร์แทนสวิทช์ในการตัดต่อกระแส ในรูป3-8 แสดงถึงสัญญาณลอจิก ต่อกับ บิท-คูเรนท์-ซิ้งสวิทช์



รูปที่3-8: Completed Monolithic Converter

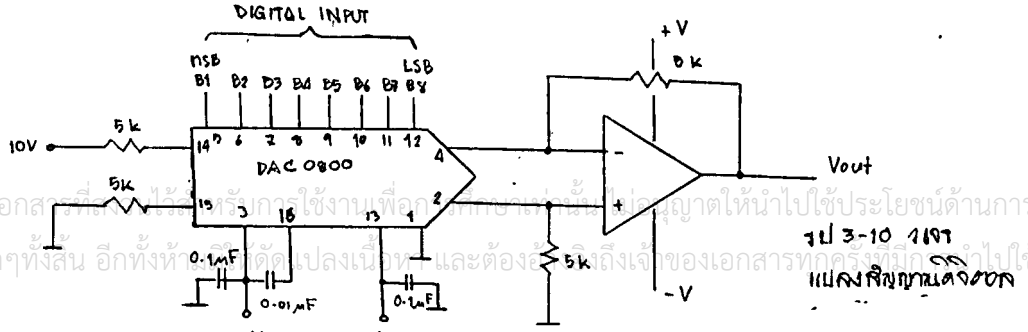
เมื่ออินพุทเป็นลอจิก"0" ที่ระดับ 0V. บิทซิ้งผ่าน Q 4 ไปยังบิท-ซิ้ง บัส เมื่ออินพุท เป็นลอจิก"1" ที่ระดับโวลท์เตจมากกว่า 2 V. กระแสจะผ่าน Q3 แทน Q4



รูปที่3-9: **Detail—The Bit Switches**

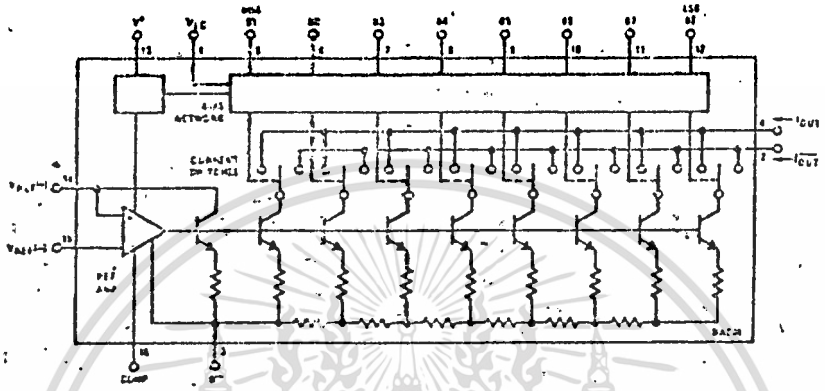
DAC 0800:

DAC 0800 มีขนาด 8 บิทเป็นแบบ high speed digital-to-analog converter (DAC) มีค่า setting time 100 nS. มีเอ้าพุทเป็นแบบ differential 20Vp-p มีกระแสอ้างอิง full-scale matting ดีกว่า + 1 LSB สำหรับรายละเอียด ดูภาคผนวก

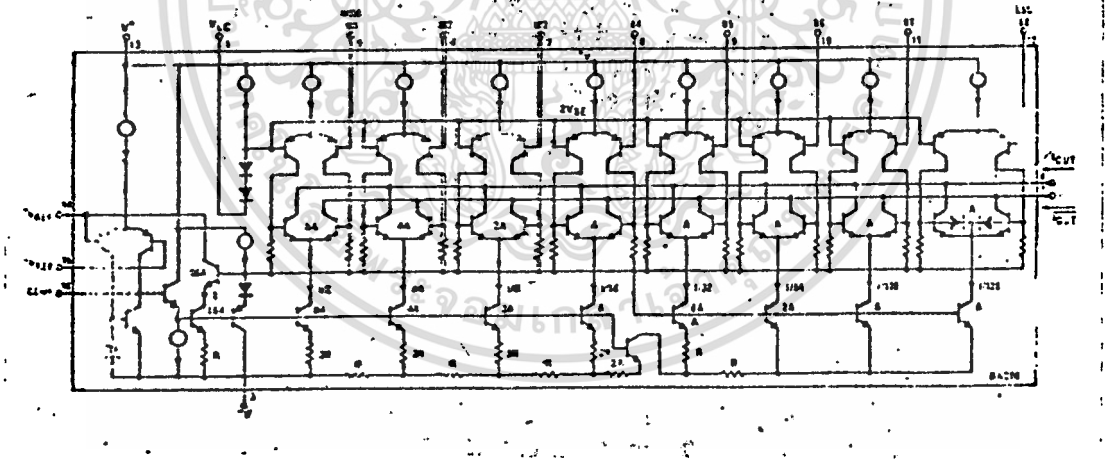


รูป 3-10 วงจร
แปลงสัญญาณดิจิตอล

block diagram



equivalent circuit



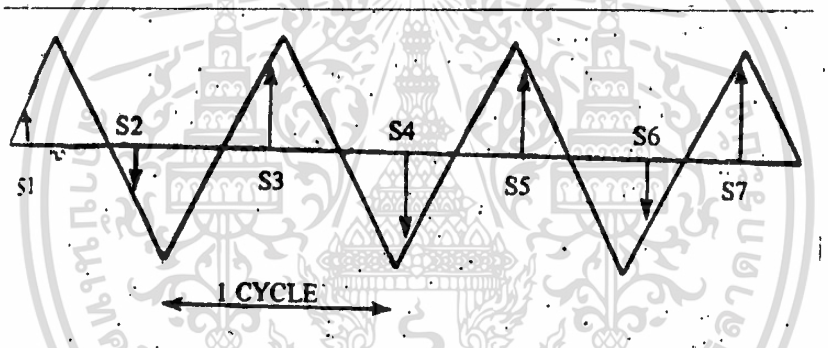
รูปที่3-11 block diagram of DAC 0800

ส่วนของการแปลงสัญญาณอนาล็อกเป็นสัญญาณดิจิตอล

เราจะใช้ข้อมูลไบนารี แทนขนาดของสัญญาณอนาล็อก การทำก็คือเราจะวัดสัญญาณอนาล็อกและเปลี่ยนเป็นข้อมูลไบนารี มีวิธีการแปลงพื้นฐานอยู่ 3 วิธี: successive approcimation, intregration, และ direct comparision ก่อนที่จะกล่าวถึงแต่ละวิธี จะกล่าวถึงทฤษฎีการสุ่มสัญญาณก่อน

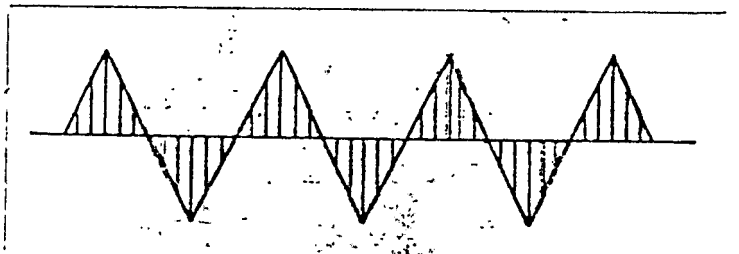
การสุ่มสัญญาณ(Sampling)

จำนวนไบนารีที่แทนสัญญาณอนาล็อกจะแทนจุด ที่เวลาหนึ่งๆ ที่เรียกว่าการสุ่มสัญญาณ (sampling) เวฟฟอร์มในรูป3-12 แสดงถึงตัวอย่างการสุ่มสัญญาณ ค่าที่สุ่มได้ไม่ได้แสดงถึงรูปร่างที่แท้จริงของสัญญาณอนาล็อก เราจะรวบรวมค่าที่สุ่มแทนสัญญาณ ความถี่ของการสุ่มเรียกว่า sampling rate



รูป3-12 **Infrequent Sampling**

เราจะต้องสุ่มสัญญาณด้วยอัตราอย่างน้อย 2 เท่าของสัญญาณที่เกิดขึ้นในระบบของเรา ในรูป3-13 แสดงถึงการสุ่มสัญญาณ



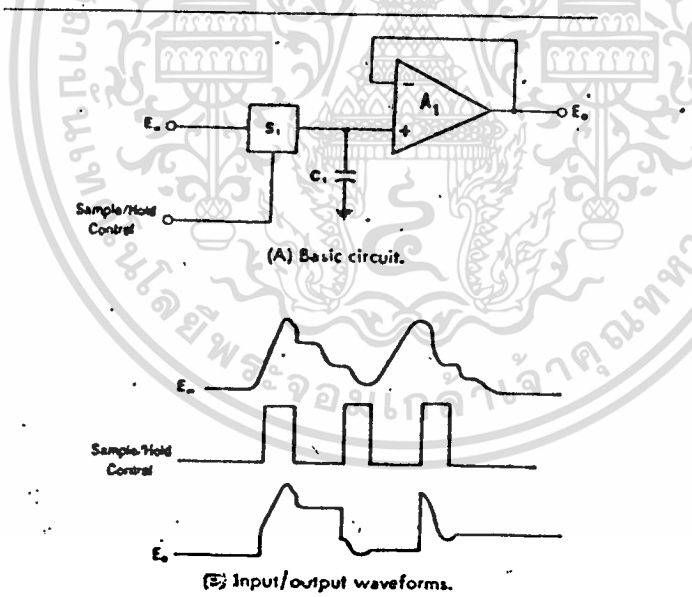
รูป3-13 : **Frequent Sampling**

วงจรSAMPLE-AND-HOLD :

วงจร sample and hold มีการทำงาน 2 ส่วนคือ ส่วนที่ 1 สุ่มสัญญาณอินพุทและส่งออกเอ้าพุท ส่วนที่ 2 ค่าสัญญาณที่สุ่มได้จะถูกคงระดับไว้ จนกว่าจะถึงการสุ่มครั้งใหม่

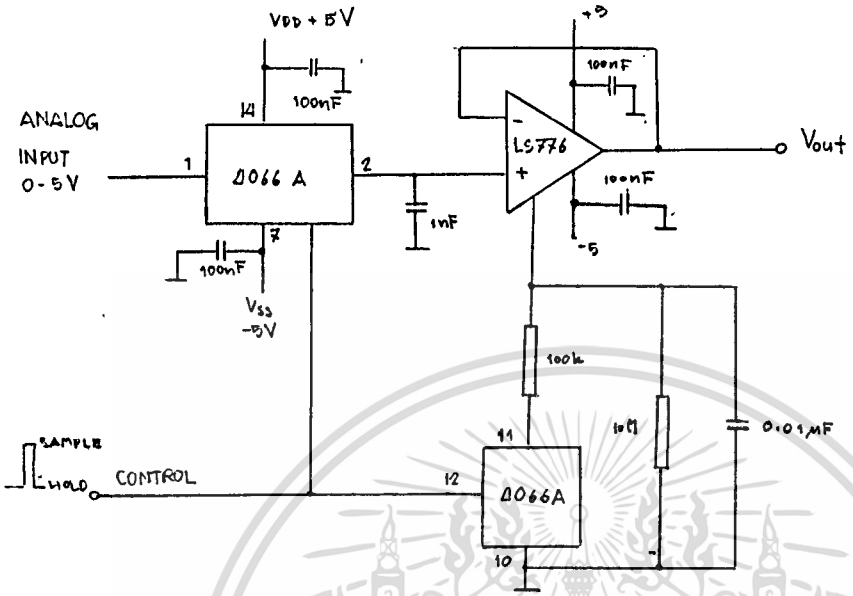
การทำงานพื้นฐาน

การทำงานเบื้องต้นของวงจรแสดงดังในรูป3-14 ในรูปa สัญญาณอินพุท(Ein) ถูกจ่ายให้กับสวิทช์อิเลคทรอนิคส์ S1 และขึ้นอยู่กับสถานะของ S1 และสัญญาณจะถูกส่งมายังC1 สถานะของสวิทช์S1 ถูกควบคุมจากขาควบคุม sample/hold เมื่อ S1 ปิด สัญญาณอินพุทจะไปตกคล่อม C1 ซึ่งมี A1 เป็น บัฟเฟอร์ ถ้า S1 ถูกต่อในช่วงเวลาหนึ่ง ขณะที่ Ein เปลี่ยนแปลงด้วยค่า ac การทำงานของวงจรจะกล่าวได้ว่า มีการ tracting Ein เมื่อสวิทช์ S1 เปิด ค่าสุดท้ายของ Ein จะถูกรักษาไว้ด้วย C1 และ A1 จะอ่านค่าแรงดัน C1 ออกไปตลอดเวลาจนกระทั่งมีการสุ่มครั้งใหม่ ดังแสดงในรูป



รูป3-14 Sample/hold operation.

High Accuracy Sample and Hold:

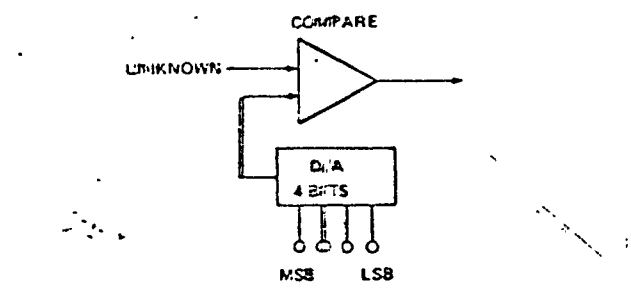


รูป3-15 high accuracy sample and hold

เทคนิคการแปลงสัญญาณอนาล็อกเป็นสัญญาณดิจิตอล
แบบSuccessive approximation

เทคนิคนี้ถูกนำมาใช้กับโมโครโปรเซสเซอร์ ซึ่งมีคุณสมบัติที่มีความเร็วสูง เที่ยงตรง และราคาถูก

หลักการของวิธีนี้จะสร้างค่าสุ่มเริ่มต้นเพื่อเดาค่าอินพุท แล้วแปลงให้เป็นสัญญาณอนาล็อก และเปรียบเทียบกับค่าอินพุทที่แท้จริง ผลลัพท์ของการเปรียบเทียบขึ้นอยู่กับค่าที่เดาเริ่มต้น ซึ่งอาจจะต้องลดหรือเพิ่มค่าที่สุ่มขึ้นมา วงจรแสดงดังรูป



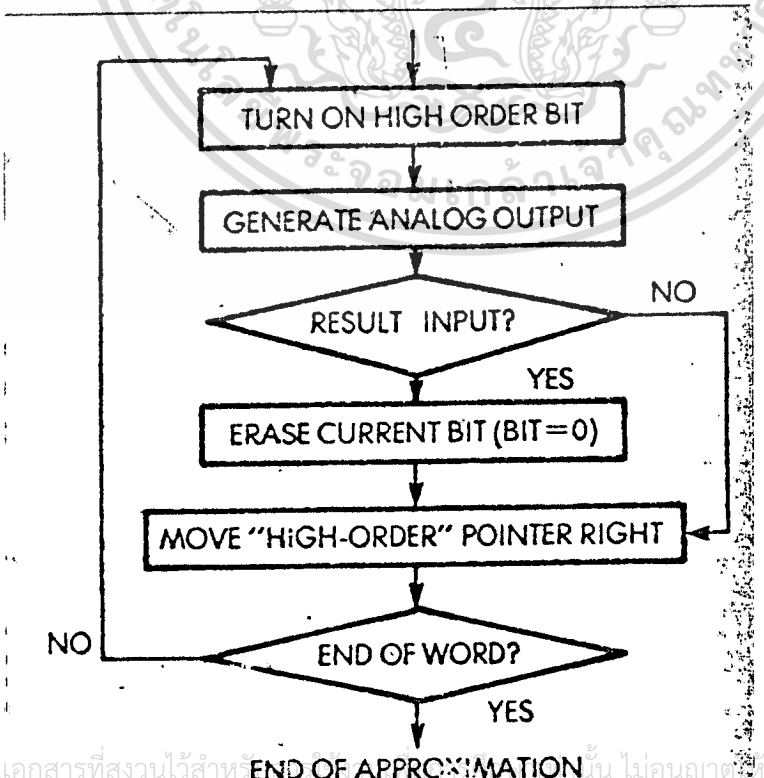
รูป3-16 Successive Approximation Hardware

สำหรับตัวอย่างนี้ ใช้ขนาด 8 บิท ค่าที่เดาเริ่มต้นเป็น "10000000" ถ้าค่าอินพุทที่แท้จริงมากว่าค่าอนาล็อกสมมูลย์ของ "10000000" ให้บิทถัดไปมีค่า"1" ค่าที่เดาถัดไปคือ "11000000" ถ้าค่านี้ยังเล็กอีก ค่าที่เดาถัดไปเป็น "11100000" ถ้าขณะนี้อินพุทที่แท้จริงน้อยกว่าค่าประมาณ บิทที่พึ่งเซ็ทเป็น "1" ที่ผ่านมา จะถูกรีเซ็ทเป็น"0" และบิทถัดไปจะถูกเซ็ทเป็น "1" ค่าที่เดาถัดไปคือ "11010000" ดั้งนี้เป็นต้น

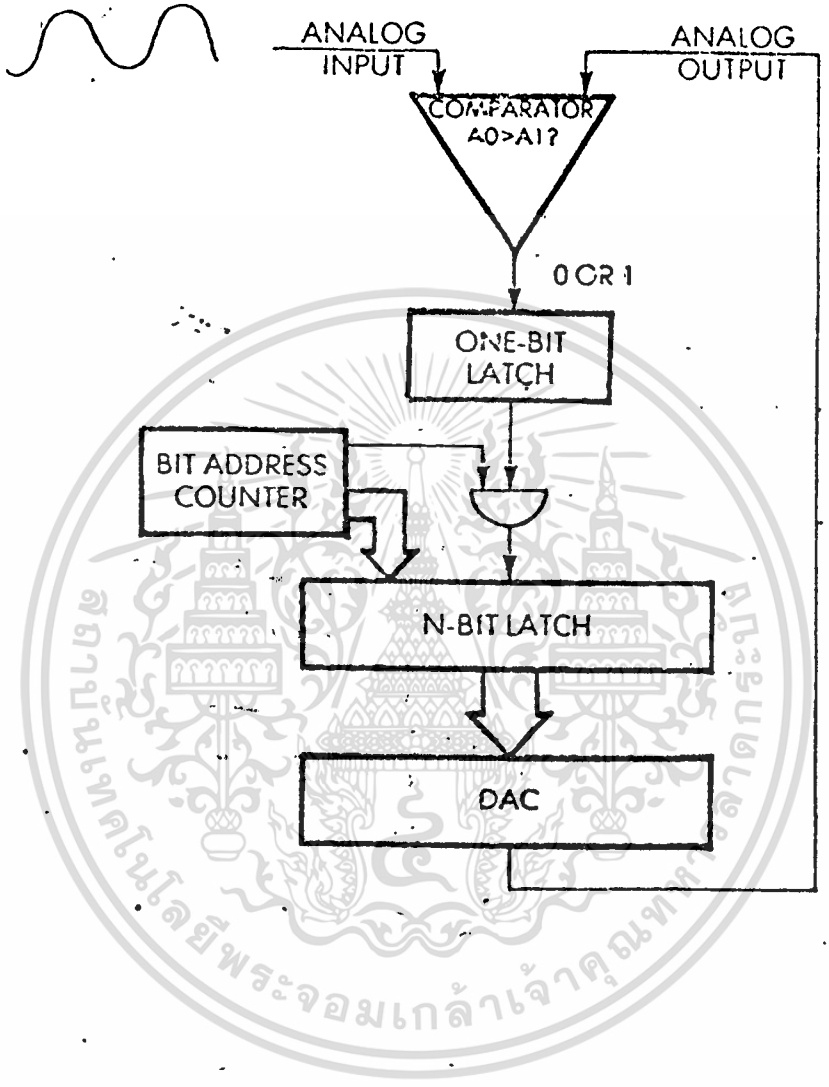
	Too low	Exact	Too high
Guess 1: 10000000	✓	—	—
Guess 2: 11000000	✓	—	—
Guess 3: 11100000	—	—	✓
Guess 4: 11010000	—	—	—

ตาราง 3-4 การประมาณค่า

เมื่อไรก็ตามที่ค่าอิทพุทที่แท้จริงมากกว่าค่าประมาณ บิทปัจจุบันจะถูกเซ็ทเป็น "1" และบิทถัดไปทางขวา จะถูกรีเซ็ทเป็น "0" ดังโฟลชาท์จ ดังรูป3-17 และรูป3-18 แสดงถึงส่วนของฮาดแวร์



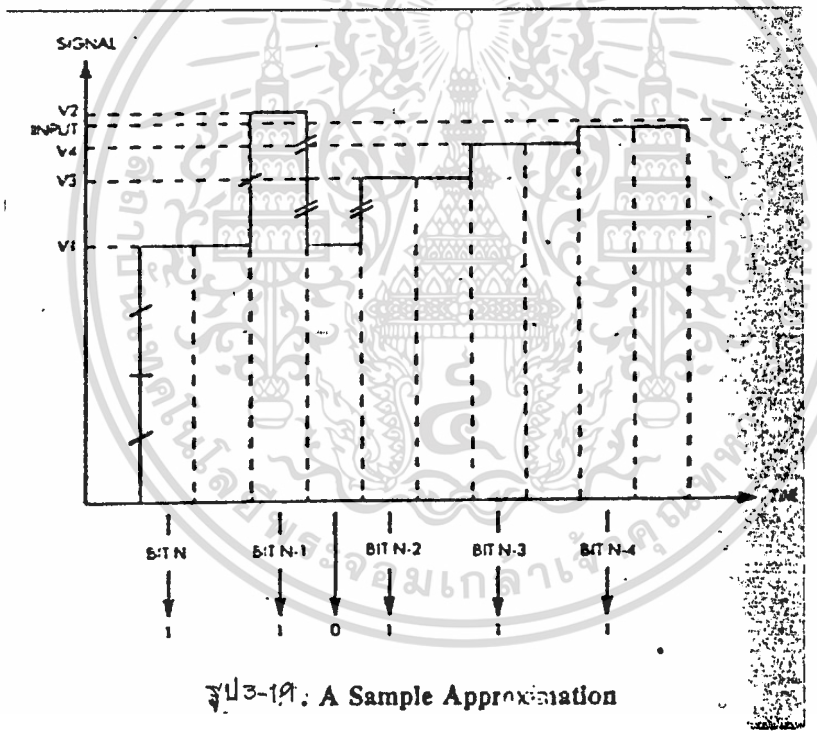
รูป3-17 : Successive Approximation ADC



รูป3-1๙ **Detailed Successive Approximation Hardware**

ตัวอย่างการสุ่มค่าประมาณแสดงดังรูป3-19

บิทสูงสุด(บิท n) เป็นบิทแรกที่ถูกเซ็ทเป็น "1" ผลของV1 มีขนาดเล็กกว่าอินพุท บิทที่อยู่ถัดไปอีกหนึ่งบิทจะถูกเซ็ทเป็น "1" ดังแสดงในรูป อย่างไรก็ตามเวลานี้ ผลลัพท์ประมาณ V2 มีขนาดใหญ่กว่าค่าอินพุท บิท n-1 ซึ่งได้รับการเซ็ทให้เป็น"1" ขณะนี้จะถูกรีเซ็ทให้เป็น "0" การประมาณค่าถัดไปจะเป็น 101 ตามด้วย 0 คือ V3 ค่านี้ยังน้อยกว่าค่าอิทพุท และบิท n-2 จะถูกเซ็ทเป็น "1" ประมาณครั้งนี้คือ 1011 ตามด้วย 0 คือ V4 ดังนี้เป็นต้น เพื่อให้สิ้นเข้า ในรูป จะใช้ 5 บิท ประมาณค่า 10111 ซึ่งจำนวนบิทที่ใช้นี้นขึ้นอยู่กับค่าความละเอียดที่ต้องการ



รูป3-19 : A Sample Approximation

ADC 0804:

ADC 0804 มีโครงสร้างเป็น CMOS 8 บิท successive approcimation A/D converters ซึ่งใช้ค่าความต้านทานที่มีระดับแตกต่างกัน 256 ระดับ ซึ่งมีบัสควบคุม, เอ้าพุทแบบ TRI-STATE ที่แล็ทได้ ไว้ต่อกับระบบบัสของ CPU โดยตรง CPU จะมอง A/D นี้เหมือนกับตำแหน่งหน่วยความจำหรือ I/O พอร์ท มีอิทพุทที่มีcommon -mode rejection สูง และoffset มีค่าศูนย์ และแรงดันอ้างอิงสามารถปรับตามอินพุทที่มีขนาดต่ำ เพื่อให้ผลตอบสนองเอ้าพุทเต็มย่าน 8 บิท

รายละเอียดของADC 0804

-มีช่วงเวลา access time 135 nS

-อินพุท/เอ้าพุท สามารถต่อกับ TTL และ MOS ได้

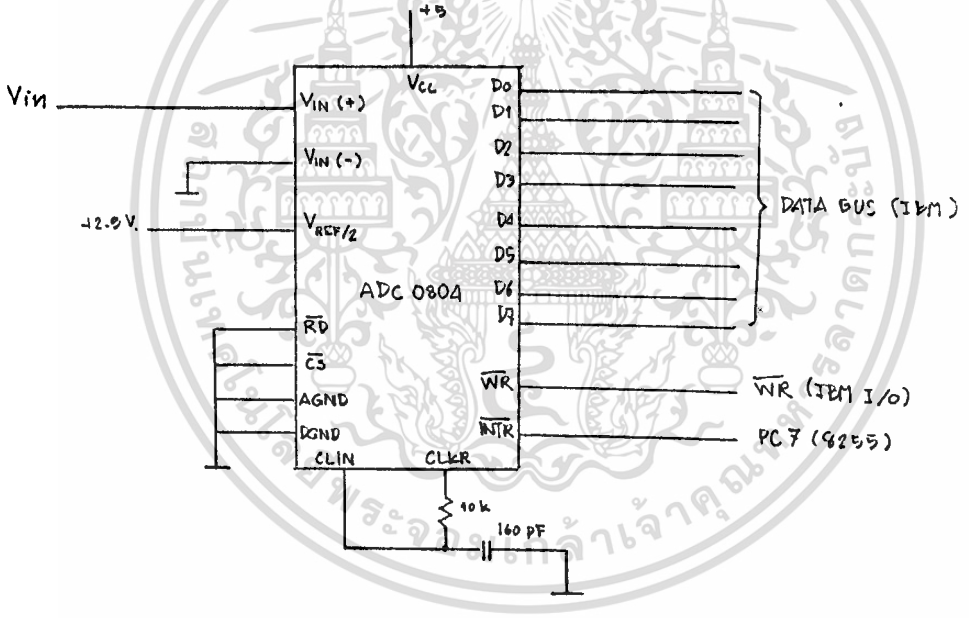
-ใช้ได้กับICสร้างแรงดันอ้างอิง 2.5V. (LM 336)

-อินพุทมีย่าน 0-5 V. ที่แหล่งจ่ายไฟ 5 V.

-ไม่ต้องปรับ zero adjust

-error ± 1 LSB

-conversion time 100 uS



รูป 3-20 วงจรแปลงสัญญาณอนาล็อกเป็นดิจิตอล

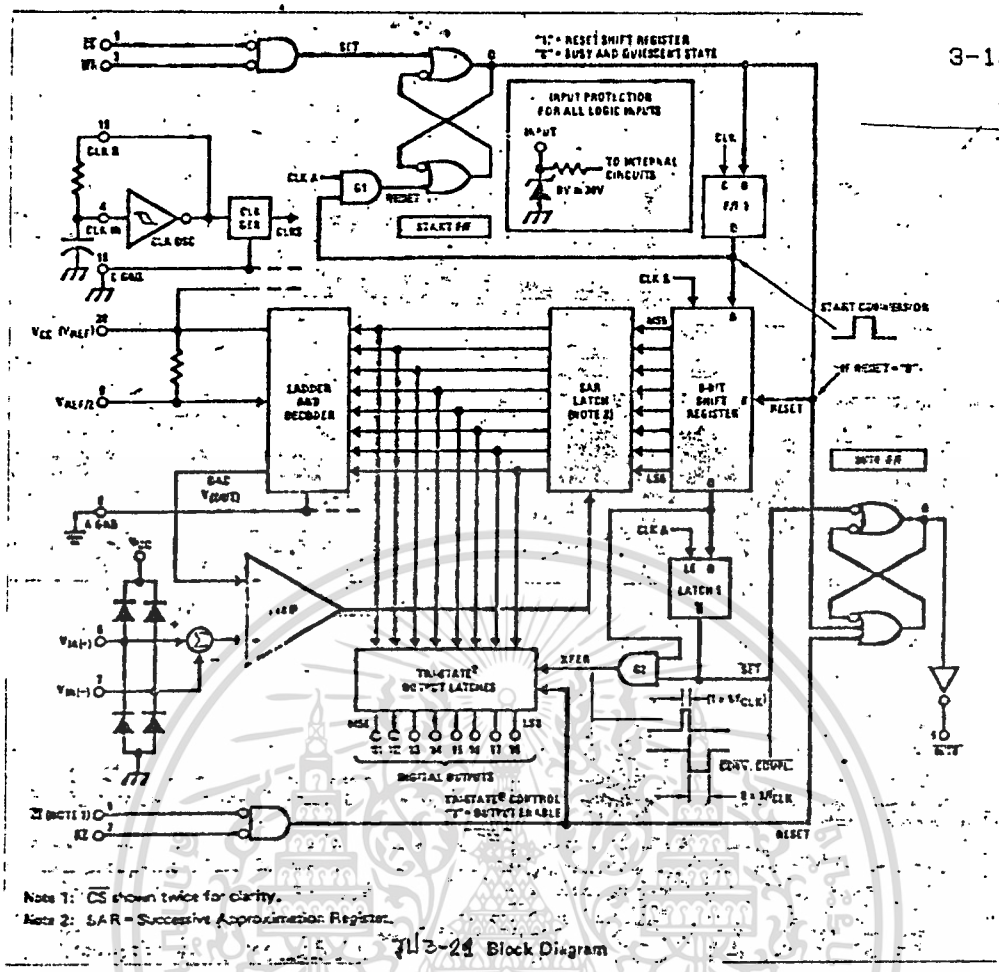
อธิบายการทำงาน

ADC 0801 series ประกอบด้วยวงจร equivalent 256R network อนาล็อกสวิทช์จัดเรียงลำดับตาม successive approcimation logic เพื่อให้ match กับอินพุทโวลท์เตจ [Vin(+)- Vin(-)] ตามลักษณะการ tap บน R network บิทสูงสุดจะถูกทดสอบก่อนบิทอื่นๆ หลังจากเปรียบเทียบครบ8 บิท (64clock cycle) สัญญาณดิจิตอล 8 บิท จะถูกส่งออกไปแล็ทที่เอ้าพุท แล้วจึงส่งสัญญาณอินเตอร์รัพท์($\overline{\text{INTR}}$ แอคทีฟ high-to-low) การแปลงสัญญาณครั้งต่อไปสามารถนำเอาสัญญาณอินเตอรัพท์มาใช้ได้ โดยให้ทำงานในโหมด free-running โดยต่อสัญญาณ $\overline{\text{INTR}}$ เข้ากับอินพุทWR และ CS=0 เพื่อให้แน่ใจว่าจะสามารถทำงานใน free-runnigได้ ขาWRจะต้องมี start pulse เกิดขึ้นขณะเริ่มจ่ายแรงดันให้วงจร

การเปลี่ยนสถานะจาก high-to-low ของขาอินพุท WR วงจรภายในคือ SAR latch และshift register จะถูกรีเซ็ท ลักษณะเดียวกับเมื่อ อินพุทCSและWR เป็นลอจิกlow A/D จะถูกรีเซ็ท การแปลงสัญญาณจะใช้เวลา 8 clock period หลังจากCSและWR เปลี่ยนสถานะจาก low-to-hight

diagram ของ A/D converter แสดงดังรูป ส่วนของlogic control แสดงด้วยเส้นหนัก

การทำงานเริ่มต้น เริ่มจากCSและWR เป็นลอจิก low ทำให้เกิดการ start flip-flop(F/F) มีเอ้าพุท "1" ไปรีเซ็ท 8 bit shift register รีเซ็ทอินเตอรัพท์($\overline{\text{INTR}}$) และมีอินพุท "1" ไปยัง D ฟลิฟฟลอฟ F/F1 ซึ่งเป็นอินพุทบิทสุดท้ายของ 8 bit shift register สัญญาณ clock ภายใน ส่งอินพุทไปยังเอ้าพุท Q ของ F/F1 และ and gate G1 จะรวมเอ้าพุท"1" นี้กับ clock signal เพื่อให้เกิดสัญญาณรีเซ็ท และ 8 bit shift register จะมีสัญญาณclock "1" เข้ามา ซึ่งทำให้เริ่มขบวณการแปลงสัญญาณ ถ้าสัญญาณรีเซ็ทยังคงอยู่ รีเซ็ทพัลซ์นี้ไม่มีผลต่อระบบ และ 8 bit shift register อยู่ในโหมดรีเซ็ท



รูป 3-21 Block Diagram

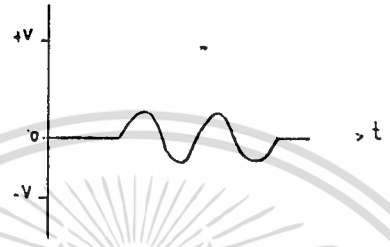
หลังจากclok ผ่าน 8 bit shift register เอ้าพุท"1" ที่มาจาก8 bit shit register and gate G2จะทำให้ข้อมูลดิจิตอลชุดใหม่ ส่งไปยังTRI-STATEเอ้าพุทแล็ท เอ้าพุทQ มีสถานะเป็น high-to-low ซึ่งทำให้ INTR F/F ถูกเซ็ท และจะทำให้เกิดสัญญาณเอ้าพุท $\overline{\text{INTR}}$

รายละเอียดขอให้ดูภาคผนวก

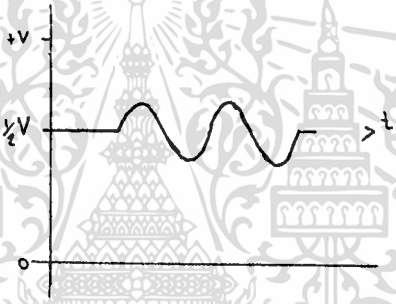
วงจรยกระดับสัญญาณ(Clamp circuit)

เนื่องจากวงจร ADC ทำงานในย่านอินพุทที่เปลี่ยนแปลงจาก 0-5 Vdc. ดังนั้น สัญญาณเสียงที่มีการแกว่งของสัญญาณในย่านระดับแรงดันบวกและลบเมื่อเทียบกับกราวด์ จะต้องทำสัญญาณให้แกว่งอยู่ในช่วงแรงดันบวกและกราวด์เท่านั้น นั่นคือจะต้องยกระดับสัญญาณขึ้นไป

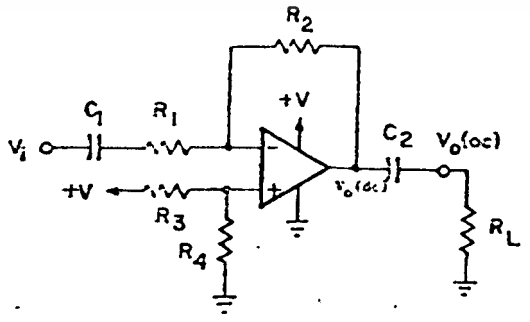
สัญญาณ ac. ทั่วๆ ไป



สัญญาณ ac. หลังจากการยกระดับ



เราจะนำเอาวงจร inverting amplifier แบบ single supply มายกระดับสัญญาณ



รูป3-22 inverting amplifier using single supply

การทำงานของวงจรนี้ จะคงค่าเอ้าพุทของวงจรให้อยู่ที่ระดับ 1/2 ของแหล่งจ่ายแรงดันเมื่อไม่มีสัญญาณอินพุท วงจรนี้ทำงานในคลาส A เมื่อมีอินพุท sine wave เข้ามาในวงจร เอ้าพุทของวงจรจะแกว่งอยู่ในช่วง 1/2 Supply ซึ่งประกอบด้วย ac component และ dc component โดยมี C2 เป็นตัว block สัญญาณ dc ไม่ให้ออกมาที่เอ้าพุท โวลท์เตจดิไวเดอร์ R3, R4 มีไว้เพื่อเซ็ทเอ้าพุทโวลท์เตจอยู่ที่1/2 supply เมื่อไม่มีอินพุท ตัวเก็บประจุ C1,C2 เป็นตัว coupling input และ output ตามลำดับ

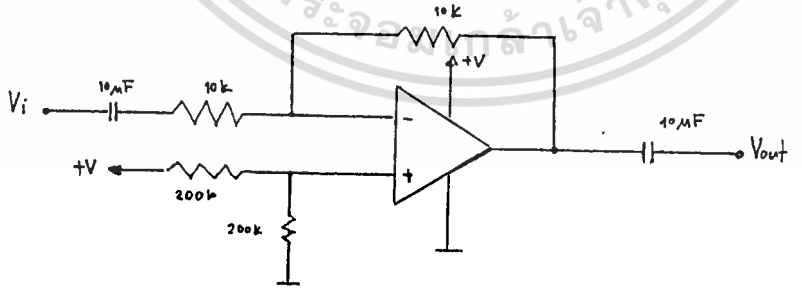
$$Vo(dc) = \frac{R_4}{R_3 + R_4} (+V)$$

ค่าของ C1 และ C2 หาได้จากการออกแบบ low-frequence responceของวงจร

$$C1 = \frac{1}{2\pi f_c R_1}$$

$$C2 = \frac{1}{2\pi f_c R_L}$$

fc is lowest frequence



รูป 3-23 Clamp amp.

การInterface กับ IBM PC/XT

ตามรูป ส่วนอินพุทและเอ้าพุทของ PIA 8255 มีพอร์ท I/O อยู่ 3 พอร์ท ซึ่งควบคุมการทำงานด้วยซอฟท์แวร์ ชิฟนี้ปรกติออกแบบให้ทำงานร่วมกับไอซีเมโครตระกูล 8080 ขา 8255 มี 40 ขา ทำงานในระบบ 8 บิท พอร์ททั้งสามเรียกใช้งานโดยการdecode แอสเดรส I/O

การสร้างส่วนอินพุทและเอ้าพุท(I/O)

การสร้างส่วนI/O สำหรับคอมพิวเตอร์ ใช้ไอซี 74LS138 จะเป็นตัวกำหนดแอดเดสเลือกการทำงานของPIA 8255 ซึ่งจะแอคทีฟในสถานะlow แอดเดสสำหรับ8255 คือ 0780H ตารางที่1 ให้I/O แอดเดสสำหรับสายสโตป 8 เส้นจาก 74LS138

สำหรับแอดเดสที่สูงกว่า 3FFFH คือแอดเดส A14 และ A15 จะไม่ถูกใช้ในการเชื่อมต่อกับIBM PC/XT เพราะแอดเดสที่สูงกว่า 3FFFH ถูกใช้ในระบบคอมพิวเตอร์ ดังนั้นแอดเดสที่เหมาะสมสำหรับการเชื่อมต่อควรเริ่มจากแอดเดส 03FFH ลงมา

โหมดการทำงานพื้นฐานสำหรับ 8255 มีสามโหมดซึ่งเลือกการทำงานได้ด้วยซอฟท์แว โหมด0 เป็นอินพุท เอ้าพุทพื้นฐาน โหมด1 คือสโตป อินพุท เอ้าพุท และโหมด2คือบัสสองทิศทาง ตัวชิพมีพอร์ทข้อมูลสามพอร์ทแยกจากกันคือ พอร์ทเอ พอร์ทบี พอร์ทซี การเลือกโหมดสำหรับพอร์ทเอ และบี เลือกแตกต่างกันได้ แต่พอร์ทซีต้องสอดคล้องตามโหมดที่เลือกไว้สำหรับเอและบี ครึ่งส่วนบนของพอร์ทซีเลือกโหมดแบบเดียวกับพอร์ทเอ ครึ่งส่วนล่างของพอร์ทซีเลือกโหมดเดียวกับพอร์ทบี

รีจิสเตอร์ภายในมี 4 ตัวในชิพ 8255 สามตัวแรกสำหรับพอร์ทข้อมูล และตัวที่ 4 คือรีจิสเตอร์ควบคุม ดังที่กล่าวมาตามรูปแบบการดีโค้ดลอจิก พอร์ทเอ มีแอดเดส I/O เท่ากับ 0780H พอร์ทบี0781H พอร์ทซี0782H และพอร์ทควบคุม0783H

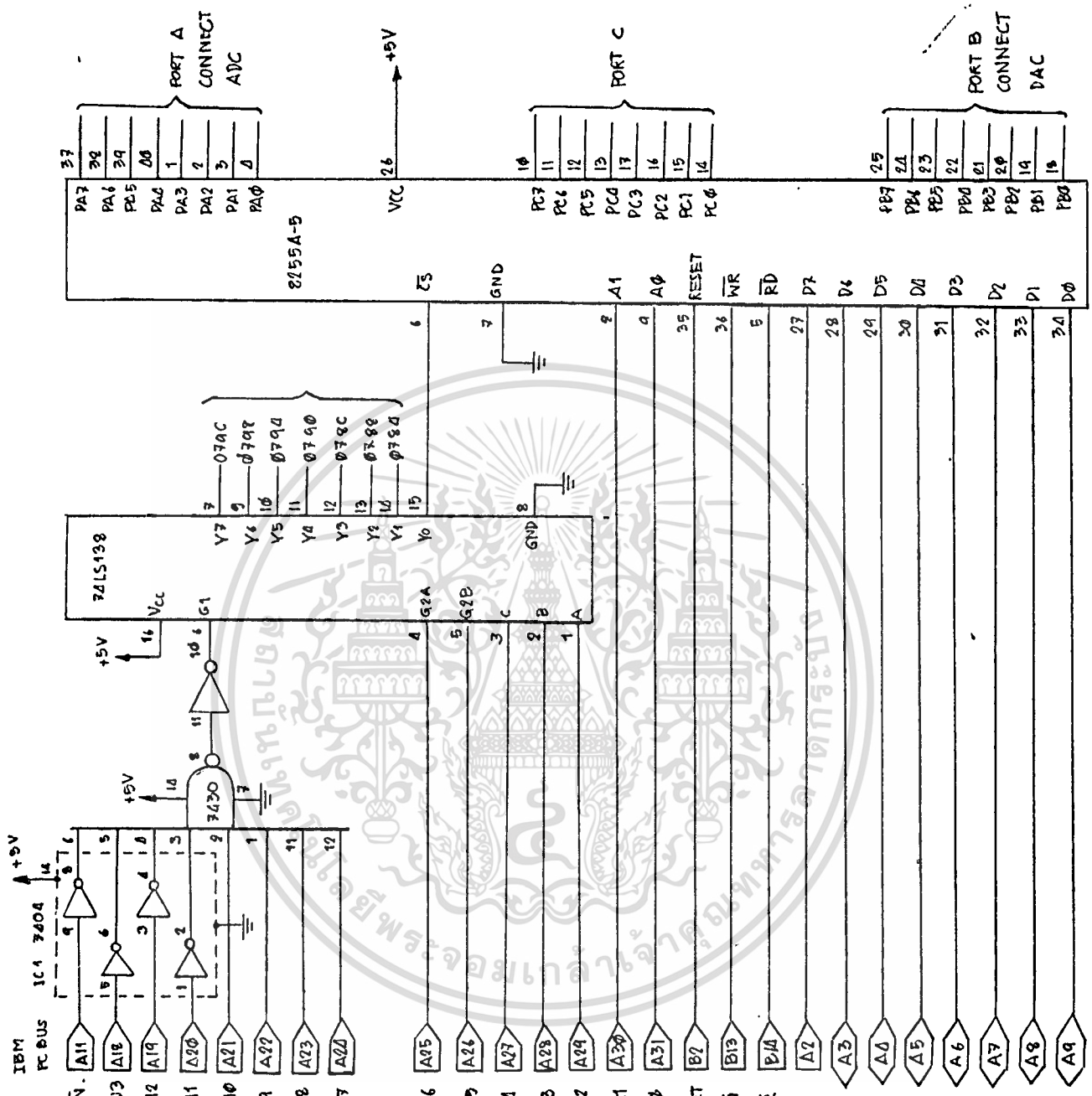
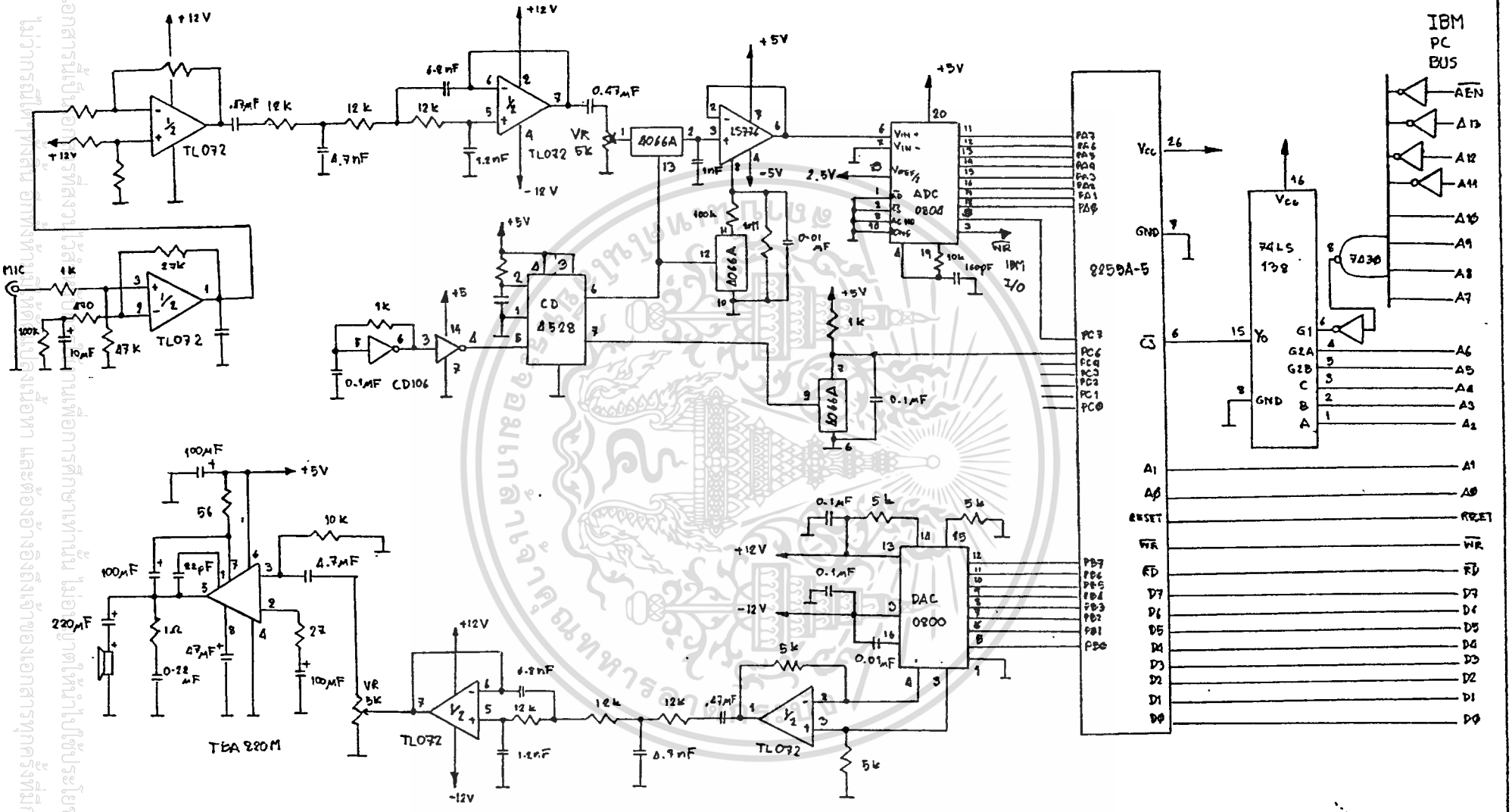
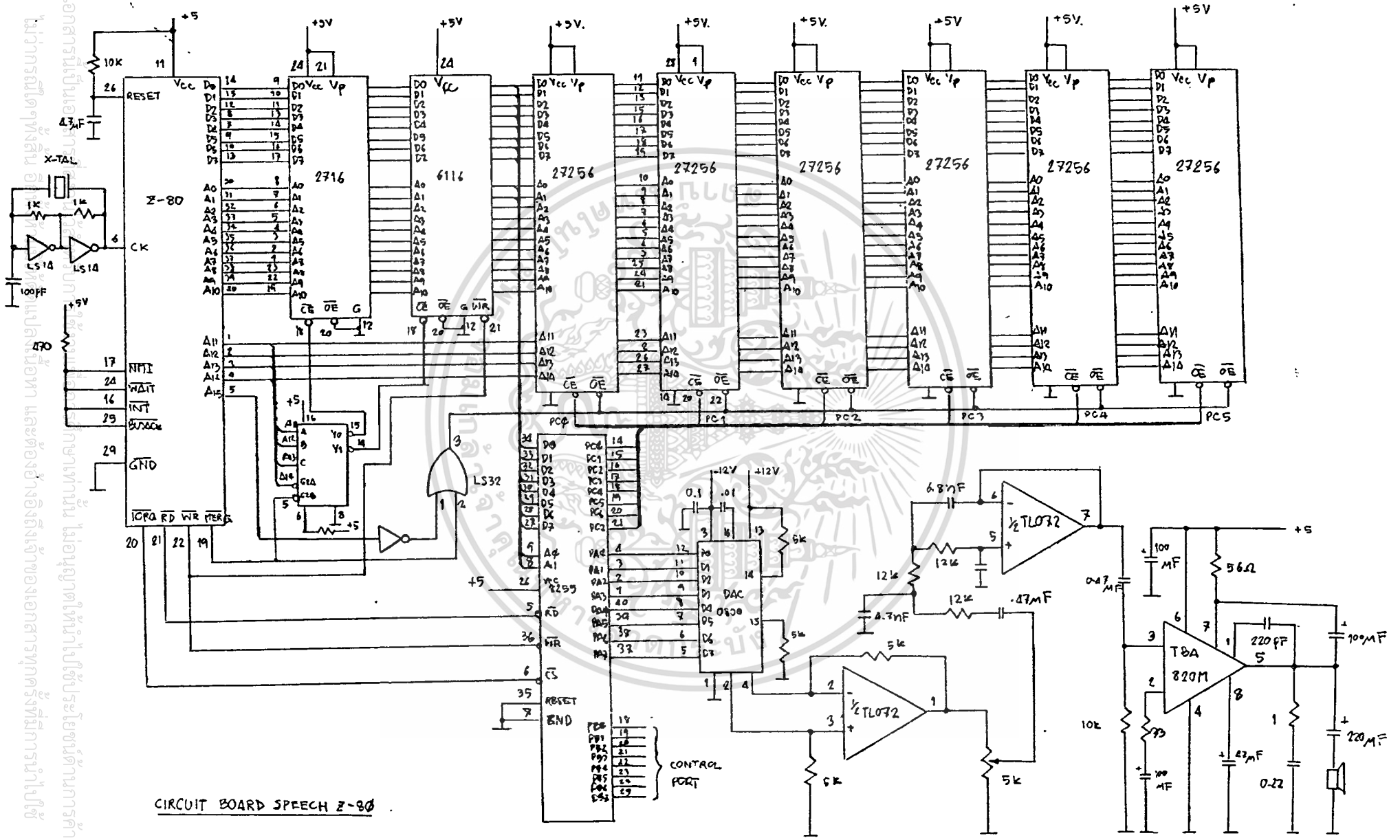


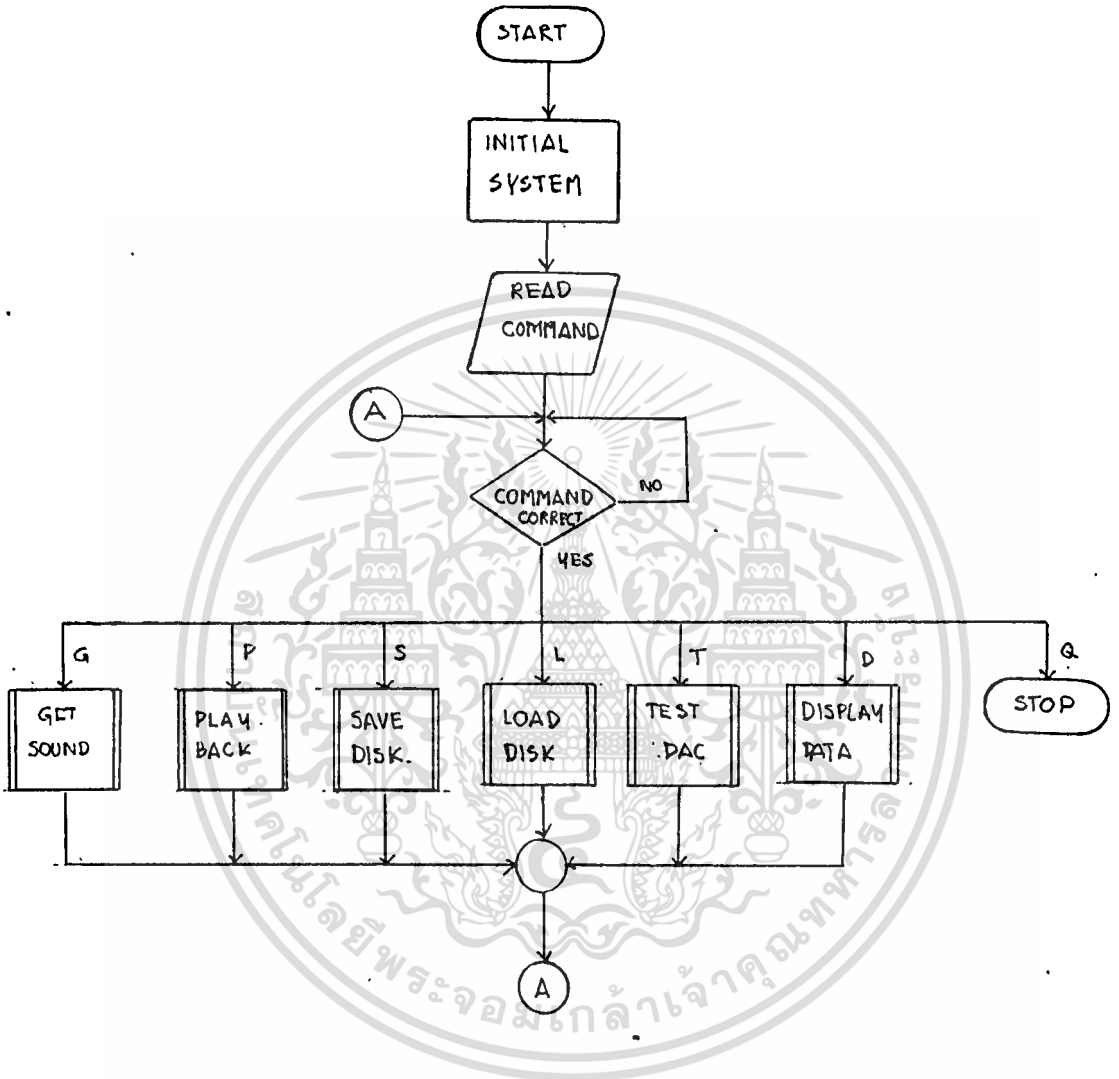
DIAGRAM OF I/O PART

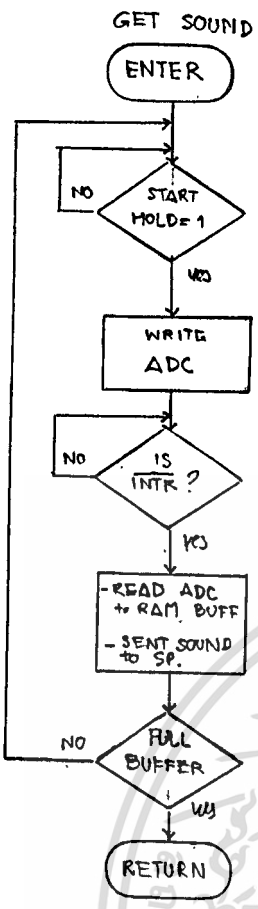


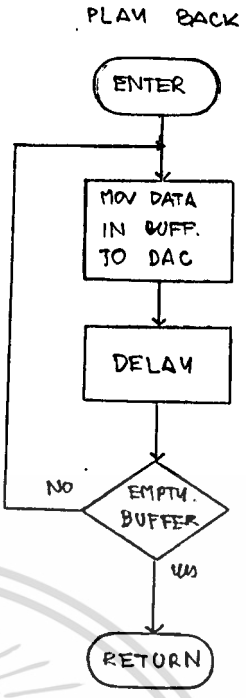
CIRCUIT OF SPEECH SYNTHESIS

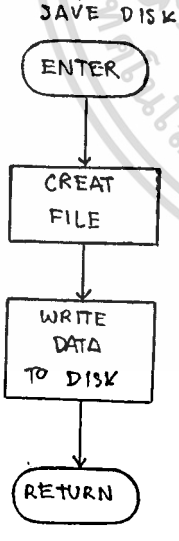

CIRCUIT BOARD SPEECH Z-80
Z-80
RESET
CK
GND
NMI
WAIT
INT
BUSRQ
IORQ RD WR MERQ
X-TAL
10K
4.7MF
1K
1K
LS14
LS14
100PF
470
2716
6116
27256
74LS138
LS32
8255
RD
WR
CS
RESET
GND
PC0
PC1
PC2
PC3
PC4
PC5
CONTROL PORT
DAC 0830
+12V
-12V
0.1
.01
5K
1/2 TL072
6.8NF
12K
.47MF
.047MF
100 MF
56Ω
TBA 820M
220 PF
10MF
220MF
10K
0.22
+5
+5V

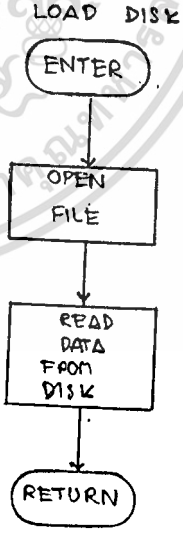
โฟล์ชาตแก็บสัญญาณเสียง.

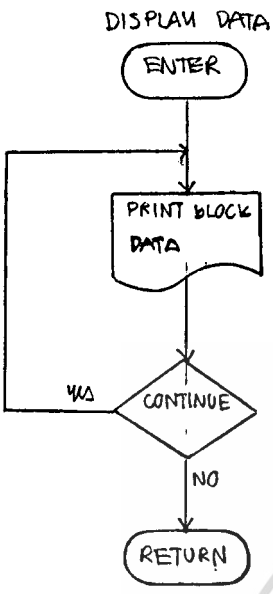


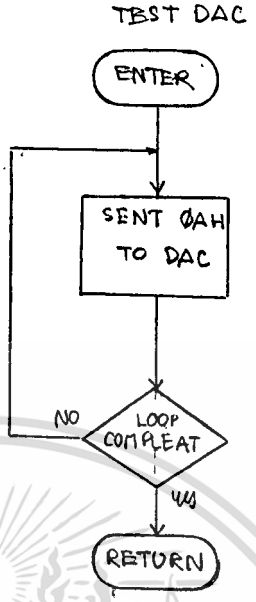




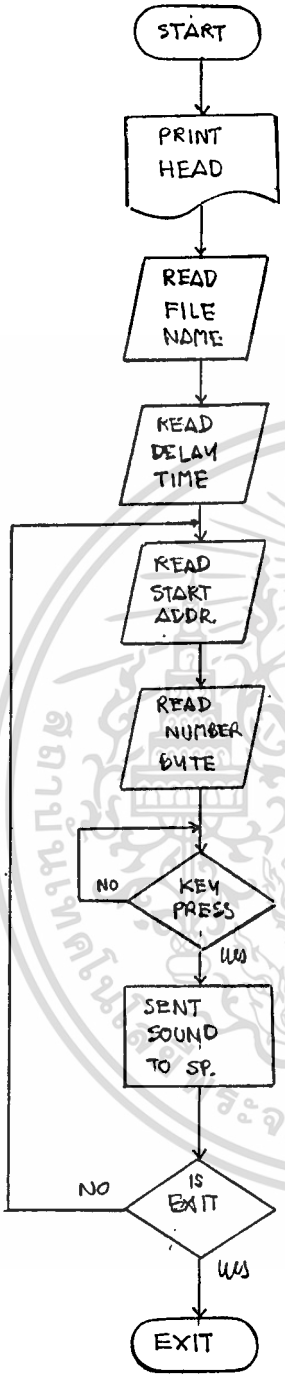




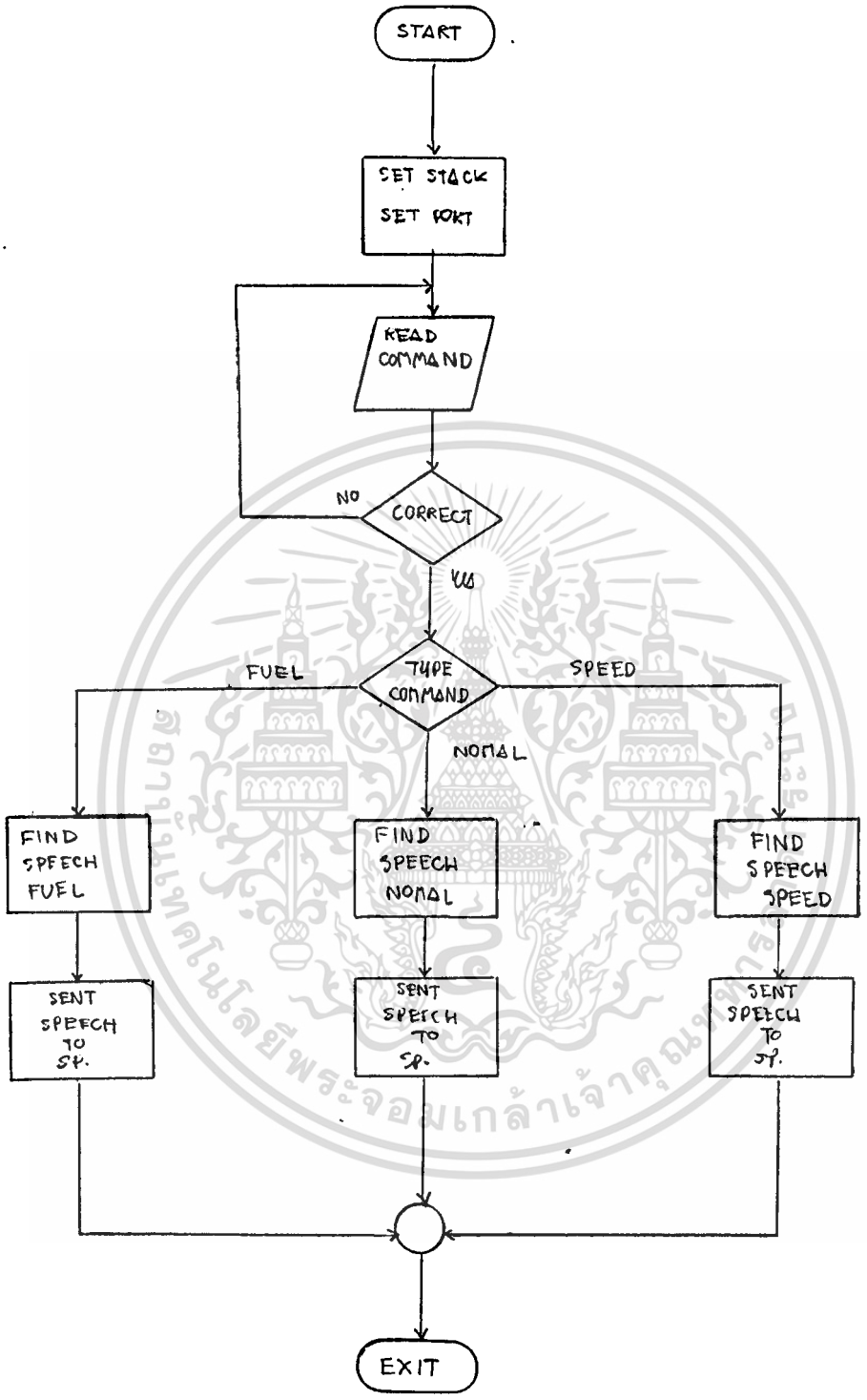




โฟลชาตหาแอดเดส



โฟลชาตสทั้งโสยนนอร์ด Z-80



```
;    I-----------------------------------------------I
;    I                                               I
;    I       PROGRAM  : PROCESS  SOUND ON IBM PC/XT  I
;    I       WRITE BY : MR.SUNGVON YUYONG            I
;    I       WRITE DATE: 18/10/30                    I
;    I-----------------------------------------------I

    PAGE  54,132
    TITLE PRO_SOUND.ASM
    INCLUDE B:MASM.MCR

    DSEG    SEGMENT

    head        db 'SAVE SOUND TO MEMORY    ==>  M ',10,13,'$'
    head1       db 'SOUND  TO  SPECKER      ==>  P ',10,13,'$'
    head2       db 'SAVE SOUND TO DISK      ==>  D ',10,13,'$'
    head3       db 'LOAD SOUND FORM DISK    ==>  L ',10,13,'$'
    head4       db 'EXIT FORM THIS PROGRAM  ==>  Q ',10,13,'$'
    head5       db 'DISPLAY  DATA IN MEMORY ==>  R ',10,13,'$'
    head6       db 'FILL  0Ah  IN  MEMORY   ==>  T ',10,13,'$'
    headsound   db 'SOUND and SOUND ',10,13,'$'
    show        db 'CREATION THE FILE DATA.DTA ! ',13,10
                db 'ENTER YOUR  FILE NAME..... ','$'
    showload    db 'LOAD THE FILE DATA.DTA !',13,10,
                db 'ENTER YOUR FILE NAME..... ','$'
    notfound    db 'FILE  NOT  FOUND',10,13,'$'
    notcreate   db 'CAN  NOT  CREATES FILE',10,13,'$'
    headsbr0    db 'SAVE SOUND  PRESSED  ANY KEY ',10,13,'$'
    FILE_NAME   DB  12 DUP(?)
                DB  25 DUP (?)
    COUNT       dw  0
    ADDRESS     DB  128 DUP(?)
    OK          DB  'FILE CREATION  OK.',13,10,'$'
    OKLOAD      DB  'LOAD  FILE  OK.',10,13,'$'
    er          DB  'DISK ERROR',10,13,'$'
    str         db  50 dup(?)
    test        db 'TEST ',10,13,'$'
    case        db ?
    handle      dw ?
    char        db ?
    blank       db ' ','$'
    newadd1     db 'ES:',13,'$'
    newadd2     db 'DI:',13,'$'


    TABLE       dw  SAVE        ;save  sound on  RAM
                dw  SOUND       ;sound to  SP.
                dw  DISK        ;save  data  to disk
                dw  LOAD        ;load  data  fore disk
                dw  QUIT        ;goto  DOS
                dw  DISPLAYM    ;display  data in RAM
                dw  TESTDAC     ;test  dac

  DSEG          ENDS

;...............SYSTEM EQUATES
    portA       equ  0780h
    portB       equ  0781h
    portC       equ  0782h
    control     equ  0783h
    portADC     equ  0784h
    startADDR   equ  6000h
    numBYTE     equ  7FFFh
    size        equ  32768
    max         equ  52
```

```
;.................START  PROGRAM.....

CSEG    SEGMENT  public 'CODE'
        ASSUME   CS:CSEG,DS:DSEG

MAIN    PROC    FAR
start:  clrscr                  ;clear screen
        mov     ax,dseg         ;DS-->data segment
        mov     ds,ax
        gotoXY 20,5
        writemssg  head
        gotoXY 20,6
        writemssg  head1
        gotoXY 20,7
        writemssg  head2
        gotoXY 20,8
        writemssg  head3
        gotoXY 20,9
        writemssg  head4
        gotoXY 20,10
        writemssg  head5
        gotoXY 20,11
        writemssg  head6
        mov     dx,control
        mov     al,98h          ; write control word 8255
        out     dx,al
rep:    readKBD  case           ; REPEAT
        mov     al,case         ;      read(KBD,ch)
        cmp     al,'m'          ;      case  ch in [M,P,D,L,Q,R] do
        jne     p               ;        M: Call save
        mov     di,0            ;        P: Call sound
        jmp     sub             ;        D: Call disk
p:      cmp     al,'p'          ;        L: Call load
        jne     d               ;        Q: Call quit
        mov     di,2            ;        R: call displaya
        jmp     sub
d:      cmp     al,'d'
        jne     l
        mov     di,4
        jmp     sub
l:      cmp     al,'l'
        jne     q
        mov     di,6
        jmp     sub
q:      cmp     al,'q'
        jne     dp
        mov     di,8
        jmp     sub
dp:     cmp     al,'r'
        jne     t
        mov     di,10
        jmp     sub
t:      cmp     al,'t'
        jne     rep
        mov     di,12
sub:    call    table[di]   ; UNTIL ch in [M,P,D,L,Q,R]
        jmp      start
MAIN    ENDP

;SAVE
;save sound to memory ON 32 K byte
;exit sound in memory buffer
SAVE    PROC    NEAR
        clrscr
        gotoXY  25,10
        writemssg  headsbr0
        mov     ax,startADDR
```

```
        mov     es,ax
        mov     di,0            ;ES : DI --> addr.  data
        mov     cx,numBYTE
        gotoXY  15,15
        call    showaddr        ;write ('ES:',segment,'DI:',offset)
        readKBD  char           ;wait key pressed

sound0:   mov     dx,portC      ; check start hold
ck:     in      al,dx           ;    repeat until start hold=1
        and     al,40h          ;    (PC 6 )
        jz      ck              ;
        mov     dx,portADC      ; write ADC
        out     dx,al           ;
        mov     dx,portC        ;
INT:    in      al,dx           ; test interrupt
        and     al,80h          ;   repeat until interrupt =0
        jnz     INT             ;   (PC 7)
        mov     dx,portADC      ;
        in       al,dx          ; read ADC  to RAM
        mov     es:[di],al      ;
        mov      dx,portB       ;
        out      dx,al          ; sent sound to SP.
        inc     di
        loop    sound0
        gotoXY  15,15
        call    showaddr
        nop
        readKBD  char
        ret
SAVE    ENDP

;SOUND
;Load  sound  to  specker
;no exit paraeeter
SOUND   PROC  NEAR
        clrscr
        gotoXY  25,10
        writemssg readsound
        readKBD  char
        mov     ax,startADDR
        mov     es,ax           ;ES:DI-->data in RAM
        mov     di,0
        mov     cx,numBYTE
sound1: mov     al,es:[di]      ;sent data to SP.
        mov     dx,portB
        out     dx,al
        inc     di
        call    delay
        loop    sound1
        nop
        ret
SOUND   ENDP

;DELAY
;delay sound
;fixt  time constance
DELAY   PROC
        push    cx
        mov     cx,1Eh          ;number delay in CX
Loop2:  dec     cx
        jnz     loop2
        pop     cx
        ret
DELAY   ENDP


;SHOWADDR
```

```
;show address on screen
;data ES, OFFSET
SHOWADDR PROC
         push   cx          ;save CX
         mov    ax,es       ;write ('ES: ',segment)
         writemssg newaddl
         mov    bl,ah       ;data in BL
         push   ax
         call   write
         pop    ax
         mov    bl,al
         call   write
         writemssg  blank
         writemssg newadd2   ;write ('DI: ',offset)
         mov    ax,di
         mov    bl,ah
         push   ax
         call   write
         pop    ax
         mov    bl,al
         call   write
         pop  cx
SHOWADDR ENDP


;WRITE
;write data in BL on screen
;data  is  binary
WRITE   PROC  NEAR
         mov     ah,bl     ;data in  BL
         mov     al,ah
         and     ah,0f0h   ;mark 4 high bit
         mov     cl,4
         shr     ah,cl     ;shift right 4 bit
         call    display   ;
         mov     ah,al
         and     ah,0fh    ;mark 4 low bit
         call     display
         ret
WRITE   ENDP

;DISPLAY
;convert binary to hext
;and disply
DISPLAY  PROC  NEAR    ;convert BINARY to HEX
         cmp   ah,0ah      ;if data >= 10 then  data + 7h
         jl    oneNINE     ;               else  nop
         add   ah,07h      ; data:=data+30h
oneNINE:  add   ah,30h
         writeCH  ah       ;write (data)
         ret
DISPLAY  ENDP

;DISK
;save sound to disk
;enter file name
DISK   PROC
       clrscr
       writemssg show
       ;read partname of file to be open
       readmssg  max,str
       nl
       ;insert zero in buffer following name
       mov  bl,str+1        ;get # of byte read
       mov  bh,0
       mov  [str+bx+2],0    ;zero in to byte
       ;creat file
       mov   dx,offset str+2;addr of name
       mov   cx,0           ;normal attribute
       mov   ah,3ch         ;creat file function
       int   21h
```

```
        mov   handle,ax       ;stor handle
        jc    error0          ;error  return
        ;write file to disk
        push  ds              ;save data segment
        mov   bx,handle       ;get  handle file
        mov   cx,numBYTE      ;get  number of byte
        mov   ax,startADDR    ;get  segment buffer to write
        mov   ds,ax
        mov   dx,0            ;offset 0
        mov   ah,40h          ;write file function
        int   21h
        pop   ds              ;return  data segment
        jc    error0          ;error  return
        ;close  file  and exit
        mov   bx,handle
        mov   ah,3eh
        int   21h
        jc    error0
        writemssg  OK
        readKBD char
        ret
error0: writemssg er        ;massage  file error
        readKBD  char
        ret
DISK    ENDP

;LOAD
;load sound in disk to buffer
;enter file name
LOAD    PROC
        clrscr
        writemssg showload
        ;readfile name
        readmssg  max,str       ;read file name
        nl
        ;insert zero in buffer following name
        mov  bl,str+1         ;get # of byte read
        mov  bh,0
        mov  [str+bx+2],0     ;zero in to byte
        ;open  file
        mov   dx,offset str+2;addr of name
        mov   al,0            ;file open for reading
        mov   ah,3dh          ;creat file function
        int   21h
        mov   handle,ax       ;stor handle
        jc    error1          ;error  return
        ;read  file
        push  ds              ;save data segment
        mov   bx,handle       ;get  handle file
        mov   cx,numBYTE      ;get  number of byte
        mov   ax,startADDR    ;get  segment buffer to read
        mov   ds,ax
        mov   dx,0            ;offset 0
        mov   ah,3fh          ;write file function
        int   21h
        pop   ds              ;return  data segment
        jc    error1
        cmp   ax,0            ;no at EOF
        je    error1
        ;close  file  and exit
        mov   bx,handle
        mov   ah,3eh
        int   21h
        jc    error1
        writemssg  OKload
        readKBD char
        ret

error1: writemssg er      ;massage  file error
        readKBD  char
```

```
        ret
LOAD    ENDP

;DISPLY
;disply data in memmory buffer
;
DISPLAYM PROC  NEAR
         clrscr
         gotoXY  20,2
         writemssg head5
         mov    ax,startADDR
         mov    es,ax
         mov    di,0
con:     nl
         mov    cl,0            ;for a := 0 to 25 do
         call   showaddr
         nl
forD1:   mov    ch,0          ;   for n:= 0 to 30 do
forD2:   mov    bl,es:[di] ;          write(data);
         push   cx
         call   write
         writemssg blank
         pop    cx
         inc    ch
         inc    di
         cmp    ch,10h
         jl     forD2
         nl                    ;writeln
         inc    cl
         cmp    cl,10h
         jl     forD1
         nl
         nl
         readKBD  char
         mov    al,char
         cmp    al,'q'
         jne    con
         ret
DISPLAYM  ENDP

;TESTDAC
;test  DAC 0800
TESTDAC PROC   NEAR
        clrscr
        writemssg  head6
        mov    ax,startADDR
        mov    es,ax
        mov    di,0
        mov    cx,numBYTE
looptst:  mov    al,0ah
        mov    es:[di],al
        inc    di
        loop   looptst
        ret
TESTDAC ENDP

;QUIT
;exit to DOS
QUIT    PROC  FAR
        mov     ax,4c00h
        int     21h
QUIT    ENDP

CSEG    ENDS

STACK   SEGMENT  stack 'STACK'
        dw    64 dup (?)
STACK   ENDS
        END
```

```
; I-------------------------------------------------I
; I                                                 I
; I         PROGRAM: PROJECT Z80                    I
; I         WRITER : MR.SUNGVON YUYONG              I
; I         DATE   : 18/1/31                        I
; I                                                 I
; I-------------------------------------------------I

;System  Equate

P8255:    EQU  03H   ;8255 I control port
PORT_C:   EQU  02H   ;8255 I port C
PORT_B:   EQU  01H   ;8255 I port B
PORT_A:   EQU  00H   ;8255 I port A
WORD:     EQU  82H   ;8255 I control word
STACK:    EQU  0FFFH ;Addr. stack
SOT:      EQU  1AH
EOT:      EQU  1BH
_A:       EQU  0AH
_B:       EQU  0BH
_C:       EQU  0CH
D:        EQU  0FH
CK:       EQU  0AH ;----
ROM_0:    EQU  11111110B
ROM_1:    EQU  11111101B
ROM_2:    EQU  11111011B
BUFFER_COMMAND:  EQU  0800H
BUFFER_SPEED:    EQU  0840H
BUFFER_FUEL:     EQU  0840H

;---------------------------------------
;Program MAIN
;Initial address 0000H

          ORG   0000H
          LD    B,0
          DJNZ  $          ;Power up delay

          LD   SP,STACK  ;Set stack
          LD   A,WORD
          OUT  (P8255),A ;Set port 8255
          ;This control word set port C and port A are output port.
          ;Port b is input port.

MAIN:     CALL  READ_COMMAND ;recive command form main board
          CALL  TYPE_COMMAND ;work command
          JR    MAIN

;READ_COMMAND
;read command form port8255 & save command in BUFFER COMMAND
;exit parameter BUFFER_COMMAND
READ_COMMAND:
          LD   HL,BUFFER_COMMAND   ;get addr. buffer
RED_COM1: CALL  READ_PORT          ;read data form master board
          CP    SOT                ;data = start of text ?
          RET   NZ                 ;no, data error
          LD    (HL),A             ;yes,
          INC   HL                 ;save 'SOT'
RED_COM2: CALL  READ_PORT          ;read data
          LD    (HL),A             ;save data
          INC   HL                 ;buffer + 1
          CP    EOT                ;data = end of text
          JR    NZ,RED_COM2        ;no,read again
          RET

;READ_PORT
;read port 8255  by handshacking
;Pc7 is sent ACK
```

```
;Pb0 in STOBE
;exit parameter
;          A = data command
READ_PORT: LD   A,8FH              ;sent ack =1
           OUT  (PORT_C),A
RD_P1:     IN   A,(PORT_B)         ;stobe =0 ?
           BIT  7,A
           JR   NZ,RD_P1
           LD   A,0FH              ;yes,
           OUT  (PORT_C),A         ;sent ack =0
RD_P2:     IN   A,(PORT_B)         ;stobe =1 ?
           BIT  7,A
           JR   Z,RD_P2
           IN   A,(PORT_B)         ;yes,reset Pb7
           RES  7,A
           LD   B,A                ;stor in B
           LD   A,8FH              ;sent ack = 1
           OUT  (PORT_C),A
           LD   A,B                ;value in A
           RET

;TYPE_COMMAND
;find type command and sent sound to SP.
TYPE_COMMAND:
           CALL  READ_TYPE         ;read type form buffer command
           CP    A
           JR    Z,TYPE_NORMAL     ;type 'A' is normal
           CP    B
           JR    Z,TYPE_SPEED      ;type 'B' is speed
           CP    C
           JR    Z,TYPE_FUEL       ;type 'C' is fuel
           RET                     ;any type return
TYPE_NORMAL:
           CALL  NORMAL_COMMAND
           RET
TYPE_SPEED:
           CALL  SPEED_COMMAND
           RET
TYPE_FUEL: CALL  FUEL_COMMAND
           RET

;READ_TYPE
;read type  command form buffer_command
;type in buffer +1
;exit parameter  in A
READ_TYPE: LD  HL,BUFFER_COMMAND
           LD  A,(HL)
           INC HL
           CP  SOT                  ;command is correct?
           JR  NZ,TYPE_C01          ;no, set type anything
           LD  A,(HL)               ;yes,type in A
           RET
TYPE_C01:  LD  A,0
           RET

;NORMAL_COMMAND
;sound is one statement
NORMAL_COMMAND:
           LD  HL,BUFFER_COMMAND    ;command in buffer +2
           INC HL
           INC HL
           LD  A,(HL)
           CP  0K                   ;command is correct ?
           CALL  C,SOUND_NORMAL     ;yes,
           RET                      ;no,
SOUND_NORMAL:
           ADD  A,A                 ;exten word
           LD   IX,TABLE            ;table
           LD   D,0
           LD   E,A
           ADD  IX,DE
```

```
            LD   L,(IX+0)           ;addr. low
            LD   H,(IX+1)           ;addr. high
            CALL SOUND
            RET

;SOUND
;sent sound to speaker
;HL=addr. of sound
SOUND:      PUSH HL
            POP  IX                 ;ld ix,hl
SOU_LOOP:   LD   A,(IX+0)           ;end of statment ?
            CP   0FFH
            RET  Z                  ;yes, ret
            OUT  (PORT_C),A         ;select ROM
            LD   L,(IX+1)           ;addr.  sound
            LD   H,(IX+2)
            LD   C,(IX+3)           ;number byte
            LD   B,(IX+4)
            LD   E,(IX+5)           ;delay time
SOU_WORD:   LD   A,(HL)
            OUT  (PORT_A),A
            LD   D,E
SOU_DELAY:
            DEC  D                  ;delay time
            JR   NZ,SOU_DELAY
            INC  HL
            DEC  BC                 ;end number byte
            LD   A,B
            OR   C
            JR   NZ,SOU_WORD
            LD   BC,06H             ;next word
            ADD  IX,BC
            JP   SOU_LOOP

;SPEED COMMAND
;statement is speed...km.
SPEED_COMMAND:
            LD   HL,BUFFER_COMMAND ;SOT
            INC  HL                 ;code or type
            INC  HL                 ;first value
            LD   B,0
SPH_COM1:   LD   A,(HL)             ;count number command
            CP   EOT
            JR   Z,SPH_FILL
            INC  B
            INC  HL
            JP   SPH_COM1
SPH_FILL:                           ;select type number
            LD   A,B
            CP   1
            JR   Z,SPEED_1
            CP   2
            JR   Z,SPEED_2
            CP   3
            JR   Z,SPEED_3
            RET
SPEED_1:    CALL  SPEED_ONE
            RET
SPEED_2:    CALL  SPEED_TWO
            RET
SPEED_3:    CALL  SPEED_TREE
            RET

;SPEED_ONE
;speed ...km.
SPEED_ONE:
            CALL  LOAD_SPEED
            LD    HL,BUFFER_COMMAND  ;SOT
            INC   HL                 ;code or type
```

```
            INC   HL                  ;first value
            LD    A,(HL)
            LD    BC,6                ;offset =6
            CALL  LOAD_VALUE
            LD    BC,12
            CALL  LOAD_KM
            LD    HL,BUFFER_SPEED
            CALL  SOUND
            RET

;SPEED_TWO
;speed...ten ...km
SPEED_TWO:
            CALL LOAD_SPEED
            LD   HL,BUFFER_COMMAND
            INC  HL
            INC  HL
            LD   A,(HL)
            LD   BC,6               ;offset= 6
            CALL LOAD_VALUE
            LD   BC,12              ;offset= 12
            CALL LOAD_TEN
            LD   BC,18              ;offset=18
            CALL LOAD_VALUE
            LD   BC,24
            CALL LOAD_KM
            LD   HL,BUFFER_SPEED
            CALL  SOUND
            RET
;SPEED_TREE
;speed...hundres ...ten...km
SPEED_TREE:
            CALL LOAD_SPEED
            LD   HL,BUFFER_COMMAND
            INC  HL
            INC  HL
            LD   A,(HL)
            LD   BC,6
            CALL LOAD_VALUE
            LD   BC,12
            CALL LOAD_HUNDRES
            LD   BC,18
            CALL LOAD_VALUE
            LD   BC,24
            CALL  LOAD_TEN
            LD   BC,30
            CALL  LOAD_VALUE
            LD   BC,36
            CALL LOAD_KM
            LD   HL,BUFFER_SPEED
            CALL  SOUND
            RET


;FUEL_COMMAND
;statement is fuel...km.
FUEL_COMMAND:
            LD    HL,BUFFER_COMMAND ;SOT
            INC   HL                ;code or type
            INC   HL                ;first value
            LD    B,0
FUE_COM1:   LD    A,(HL)            ;count number command
            CP    EOT
            JR    Z,FUE_FILL
            INC   B
            INC   HL
            JP    FUE_COM1
FUE_FILL:                           ;select type number
            LD   A,B
            CP   1
            JR   Z,FUEL_1
            CP   2
```

```
            JR   Z,FUEL_2
            CP   3
            JR   Z,FUEL_3
            RET
FUEL_1:     CALL  FUEL_ONE
            RET
FUEL_2:     CALL  FUEL_TWO
            RET
FUEL_3:     CALL  FUEL_TREE
            RET

;FUEL_ONE
;fuel ...km.
FUEL_ONE:
            CALL  LOAD_FUEL
            LD    HL,BUFFER_COMMAND  ;SOT
            INC   HL                 ;code or type
            INC   HL                 ;first value
            LD    A,(HL)
            LD    BC,6               ;offset =6
            CALL  LOAD_VALUE
            LD    BC,12
            CALL  LOAD_KM
            LD    HL,BUFFER_FUEL
            CALL  SOUND
            RET

;FUEL_TWO
;fuel ...ten ...km
FUEL_TWO:
            CALL LOAD_FUEL
            LD   HL,BUFFER_COMMAND
            INC  HL
            INC  HL
            LD   A,(HL)
            LD   BC,6               ;offset= 6
            CALL LOAD_VALUE
            LD   BC,12              ;offset= 12
            CALL LOAD_TEN
            LD   BC,18              ;offset=18
            CALL LOAD_VALUE
            LD   BC,24
            CALL LOAD_KM
            LD   HL,BUFFER_FUEL
            CALL  SOUND
            RET
;FUEL_TREE
;fuel ...hundres ...ten...km
FUEL_TREE:
            CALL LOAD_FUEL
            LD   HL,BUFFER_COMMAND
            INC  HL
            INC  HL
            LD   A,(HL)
            LD   BC,6
            CALL LOAD_VALUE
            LD   BC,12
            CALL LOAD_HUNDRES
            LD   BC,18
            CALL LOAD_VALUE
            LD   BC,24
            CALL  LOAD_TEN
            LD   BC,30
            CALL  LOAD_VALUE
            LD   BC,36
            CALL LOAD_KM
            LD   HL,BUFFER_FUEL
            CALL  SOUND
            RET

;LOAD_SPEED
;load 'speed' in RAM buffer_speed
LOAD_SPEED: LD HL,SPEED          ;in table
```

```
          LD   DE,BUFFER_SPEED    ;in RAM
          LD   BC,6
          LDIR                    ;load increment repeat
          RET

;LOAD_KM
;load 'kilometic' in RAM buffer_speed + offset
;input : offset in register BC
LOAD_KM:  LD   HL,BUFFER_SPEED
          ADD  HL,BC              ;buffer + offset
          LD   D,H
          LD   E,L
          LD   HL,KM              ; KM in table_2
          LD   BC,6
          LDIR
          LD   A,0FFH             ;end command
          LD   (DE),A
          RET

;LOAD_FUEL
;load 'fuel' in RAM buffer_fuel
LOAD_FUEL: LD HL,FUEL           ;in table
          LD   DE,BUFFER_FUEL    ;in RAM
          LD   BC,6
          LDIR                    ;load increment repeat
          RET

;LOAD_LIT
;load 'lit' in RAM buffer_fuel + offset
;input : offset in register BC
LOAD_LIT:   LD   HL,BUFFER_FUEL
          ADD  HL,BC              ;buffer + offset
          LD   D,H
          LD   E,L
          LD   HL,LIT             ;LIT in table_2
          LD   BC,6
          LDIR
          LD   A,0FFH             ;end command
          LD   (DE),A
          RET


;LOAD_VALUE
;load value in RAM buffer_speed
;input : offset in register BC
;input : value in  register A
LOAD_VALUE: ADD A,A
          LD  IX,TABLE_2          ;table value
          LD  D,0
          LD  E,A
          ADD IX,DE
          LD  L,(IX+0)
          LD  H,(IX+1)
          LD  IX,BUFFER_SPEED
          ADD IX,BC               ;buffer speed + offset
          PUSH  IX
          POP   DE
          LD  BC,6
          LDIR
          RET

;LOAD_TEN
;load 'ten' in RAM buffer speed+offset
;offset in register BC
LOAD_TEN:  LD  HL,BUFFER_SPEED
          ADD  HL,BC              ;buffer speed+ offset
          LD   D,H
          LD   E,L
          LD   HL,TEN             ;ten in table_2
          LD   BC,6
          LDIR
```

```
          RET

;LOAD_HUNDRES
;load 'hundres' in RAM buffer_speed + offset
LOAD_HUNDRES:
          LD  HL,BUFFER_SPEED
          ADD HL,BC
          LD  D,H
          LD  E,L
          LD  HL,WORD_HUNDRES
          LD  BC,6
          LDIR
          RET


;----------system define---------
TABLE     DEFW   WORD_1
          DEFW   WORD_2
          DEFW   WORD_3
          DEFW   WORD_4
          DEFW   WORD_5
          DEFW   WORD_6
          DEFW   WORD_7
          DEFW   WORD_8
          DEFW   WORD_9
          DEFW   WORD_A

WORD_1:   DEFB   ROM_1        ;turn light right on
          DEFW   8FFFH
          DEFW   0A00H
          DEFB   16H
          DEFB   ROM_1
          DEFW   9B00H
          DEFW   0B00H
          DEFB   16H
          DEFB   ROM_1
          DEFW   0A700H
          DEFW   0D00H
          DEFB   16H
          DEFB   ROM_1
          DEFW   0BCD0H
          DEFW   0D00H
          DEFB   16H
          DEFB   0FFH

WORD_2:   DEFB   ROM_1          ;turn light right off
          DEFW   8FFFH
          DEFW   1F00H
          DEFB   16H
          DEFB   ROM_1
          DEFW   0CBFFH
          DEFW   0B80H
          DEFB   16H
          DEFB   0FFH

WORD_3:   DEFB   ROM_1          ;turn light on
          DEFW   8FFFH
          DEFW   1300H
          DEFB   16H
          DEFB   ROM_1
          DEFW   0AEFFH
          DEFW   1200H
          DEFB   16H
          DEFB   ROM_1
          DEFW   0C0FFH
          DEFW   0B00H
          DEFB   16H
          DEFB   0FFH

WORD_4:   DEFB   ROM_1          ;turn light off
          DEFW   8FFFH
```

```
          DEFW   1300H
          DEFB   16H
          DEFB   ROM_1
          DEFW   0AEFFH
          DEFW   1200H
          DEFB   16H
          DEFB   ROM_1
          DEFW   0CBFFH
          DEFW   0B90H
          DEFB   16H
          DEFB   0FFH

WORD_5:   DEFB    ROM_1         ;high light on
          DEFW    8FFFH
          DEFW    0200H
          DEFB    16H
          DEFB    ROM_1
          DEFW    0D77FH
          DEFW    0B80H
          DEFB    16H
          DEFB    ROM_1
          DEFW    0C0FFH
          DEFW    0B00H
          DEFB    16H
          DEFB    0FFH

WORD_6:   DEFB    ROM_1         ;high light off
          DEFW    8FFFH
          DEFW    0800H
          DEFB    16H
          DEFB    ROM_1
          DEFW    0D77FH
          DEFW    0B80H
          DEFB    16H
          DEFB   ROM_1
          DEFW   0C8FFH
          DEFW   0B80H
          DEFB   16H
          DEFB   0FFH

WORD_7:   DEFB    ROM_1         ;back light on
          DEFW    8FFFH
          DEFW    0800H
          DEFB    16H
          DEFB    ROM_1
          DEFW    0E37FH
          DEFW    0900H
          DEFB    16H
          DEFB    ROM_1
          DEFW    0C0FFH
          DEFW    0B00H
          DEFB    16H
          DEFB    0FFH

WORD_8:   DEFB    ROM_1         ;back light off
          DEFW    8FFFH
          DEFW    0300H
          DEFB    16H
          DEFB    ROM_1
          DEFW    0E37FH
          DEFW    0900H
          DEFB    16H
          DEFB    ROM_1
          DEFW   0CBFFH
          DEFW   0B80H
          DEFB   16H
          DEFB   0FFH

WORD_9:   DEFB   ROM_2          ;break down temperature
```

```
          DEFW   9400H
          DEFW   2D00H
          DEFB   16H
          DEFB   0FFH

WORD_A:   DEFB   ROM_2     ;empty fuel
          DEFW   0B900H
          DEFW   0C00H
          DEFB   16H
          DEFB   ROM_2
          DEFW   0CF80H
          DEFW   0E00H
          DEFB   16H
          DEFB   0FFH

TABLE_2   DEFW   ZERO
          DEFW   ONE
          DEFW   TWO
          DEFW   TREE
          DEFW   FOUR
          DEFW   FIVE
          DEFW   SIX
          DEFW   SEVEN
          DEFW   EIGTH
          DEFW   NINE
          DEFW   TEN
          DEFW   ELEVEN
          DEFW   TWENTY

ZERO:     DEFB    ROM_0
          DEFW    8900H
          DEFW    0B00H
          DEFB    1BH

ONE:      DEFB    ROM_0
          DEFW    9300H
          DEFW    0600H
          DEFB    1BH

TWO:      DEFB    ROM_0
          DEFW    9900H
          DEFW    0370H
          DEFB    1BH

TREE:     DEFB    ROM_0
          DEFW    0A100H
          DEFW    0930H
          DEFB    1BH

FOUR:     DEFB    ROM_0
          DEFW    0AA30H
          DEFW    07D0H
          DEFB    1BH

FIVE:     DEFB    ROM_0
          DEFW    0B200H
          DEFW    0890H
          DEFB    1BH

SIX:      DEFB    ROM_0
          DEFW    0BA90H
          DEFW    0950H
          DEFB    1BH

SEVEN:    DEFB    ROM_0
          DEFW    0C3E0H
          DEFW    0600H
          DEFB    1BH

EIGTH:    D
```

```
            EFB     ROM 0
            DEFW    0C940H
            DEFW    0700H
            DEFB    1BH

NINE:       DEFB    ROM 0
            DEFW    0D040H
            DEFW    0850H
            DEFB    1BH

TEN:        DEFB     ROM 0
            DEFW     0DB90H
            DEFW     0900H
            DEFB     1BH

ELEVEN:     DEFB     ROM 0
            DEFW     0E290H
            DEFW     0650H
            DEFB     1BH

TWENTY:     DEFB    ROM 0
            DEFW    0EAE0H
            DEFW    0650H
            DEFB    1BH

FUEL:       DEFB    ROM 2
            DEFW    0B900H
            DEFW    0C00H
            DEFB    16H
            DEFB    ROM 2
            DEFW    0B100H
            DEFW    0800H
            DEFB    16H

LIT:        DEFB    ROM 2
            DEFW    0C500H
            DEFW    0A80H
            DEFB    16H

SPEED:      DEFB    ROM 1
            DEFW    0F770H
            DEFW    088FH
            DEFB    16H

KM:         DEFB    ROM 2
            DEFW    8000H
            DEFW    1400H
            DEFB    16H


WORD_HUNDRES: DEFB ROM 0
            DEFW    0EAE0H
            DEFW    1400H
            DEFB    14H

            END
```

```
;   I-------------------------------------------------I
;   I                                                 I
;   I     PROGRAM   :FIND ADDRESS SOUND & NUMBERBYTE  I
;   I     WTRITE BY :MR.SUNGVON YUYONG                I
;   I     COMPUTER  :IBM PC/XT                        I
;   I     WRITE DATE:17/12/30                         I
;   I                                                 I
;   I-------------------------------------------------I

    PAGE   54,132
    TITLE  ADD S.ASM
    INCLUDE MASM.MCR

    DSEG    SEGMENT

    _FIND      DB 'FIND ADDRESS SOUND',10,13,'$'
    _ENTER     DB 'ENTER FILE NAME: ','$'
    _STR_ADDR  DB 'START ADDRESS: ','$'
    _PRESS     DB 'PRESS KEY ENTER & STOP BY ANY KEY',10,13,'$'
    _END       DB 'END   ADDR: ','$'
    _CONTI     DB 'EXIT BY PRESS =>0','$'
    _VALUE     DB 'DELAY TIME: ','$'
    _NUM       DB 'NUMBER BYTE: ','$'
    NOT_HEX    DB 'NOT HEXDECIMAL','$'
    blank      DB '                                                ',10,13,'$'
    VALUE      DB  50 DUP(?)
    NUM        DB  50 DUP(?)
    DE_VALUE   DW  0
    OFF_SET    DW  0
    er         DB  'DISK ERROR',10,13,'$'
    str        db  50 dup(?)
    HANDLE     DW  ?
    char       db ?
    NUM_SOUND  DW  ?

    DSEG    ENDS

;.......SYSTEM EQUATES
    portA       equ  0780h
    portB       equ  0781h
    portC       equ  0782h
    control     equ  0783h
    portADC     equ  0784h
    startADDR   equ  9000h
    numBYTE     equ  7FFFh
    size        equ  32768
    max         equ  52

;.....START  PROGRAM.....

    CSEG    SEGMENT  public 'CODE'
            ASSUME   CS:CSEG, DS:DSEG

    MAIN    PROC     FAR
    start:  clrscr                  ;clear screen
            PUSH     DS
            SUB      AX,AX
            PUSH     AX

            mov      ax,dseg        ;DS-->data segment
            mov      ds,ax
            MOV      AL,80H         ;set 8255
            MOV      DX,CONTROL
            OUT      DX,AL

            GOTOXY   4,4
            WRITEMSG  _FIND         ; find addr sound

            GOTOXY    4,5
            WRITEMSG   _ENTER       ; enter file name
            READMSG   MAX,STR
```

```
            CALL      LOAD          ; load file to memory

            GOTOXY    4,6
            WRITEMSSG   VALUE       ; delay time
            READMSSG    MAX,VALUE
            MOV     SI,OFFSET  VALUE+2
            CALL    STR_HEX       ;read delay time
            MOV     DE_VALUE,AX

   REP_M:   GOTOXY    4,7
            WRITEMSSG   STR_ADDR    ; start address:....
            READMSSG    MAX,STR
            MOV     SI,OFFSET STR+2
            CALL    STR_HEX       ;read offset
            MOV     OFF_SET,AX

            GOTOXY    4,8
            WRITEMSSG   NUM         ;read number byte
            READMSSG    MAX,NUM
            MOV     SI,OFFSET NUM+2
            CALL    STR_HEX
            MOV     NUM_SOUND,AX

            GOTOXY    4,9
            WRITEMSSG   PRESS       ; pressed any key
            CALL      SOUND

            GOTOXY   4,12
            WRITEMSSG   CONTI       ; continue y/n
            READKBD     CHAR
            MOV       AL,CHAR
            CMP       AL,'Q'
            JE        RET_MAIN
            CMP       AL,'q'
            JNE       Y
RET_MAIN:   RET
Y:          gotoxy    4,7
            writemssg  blank
            writemssg  blank
            writemssg  blank
            writemssg  blank
            writemssg  blank
            writemssg  blank
            writemssg  blank
            writemssg  blank

            JMP       REP_M
MAIN        ENDP

;SOUND
;Load  sound  to lound specker

   SOUND    PROC
            readKBD  char
            mov     ax,startaddr
            mov     es,ax         ;ES:DI-->data in RAM
            mov     cx,NUM_SOUND
            mov     di,off_set
    sound1: mov     al,es:[di]    ;sent data to SP.
            mov     dx,port8
            out     dx,al
            inc     di
            call    delay
            loop    sound1
_STOP:      GOTOXY  4,11
            WRITEMSSG   END
            MOV     BX,DI
            CALL    BINIHEX
            ret
   SOUND    ENDP
```

```
;DELAY
;delay  sound
   DELAY     PROC    NEAR
             push    cx
             mov     cx,DE_VALUE       ;number delay in CX
    loop2:   dec     cx
             jnz     loop2
             pop     cx
             ret
    DELAY    ENDP


;STR_HEX
;procedure convert string  hex. TO binary
;input   SI->string STR+2 lest significant digit appearing first
;output  AX = binary or hex.
STR_HEX    PROC   NEAR
           mov   di,si
           push   si
           mov   cl,2
           mov   ch,0
   tr:     mov   al,[si+3]
           mov   ah,[di]
           mov   [si+3],ah
           mov   [di],al
           inc   di
           dec   si
           loop  tr
           pop   si
           MOV   CL,4        ;use cl as shift counter
           MOV   CH,CL       ;use ch as digit counter
           CLD               ;case string auto incleament
           SUB   AX,AX
           SUB   DX,DX
L_STR_H:   LODSB             ;load an ASCII code hex
           AND   AL,7FH      ;clear the parity bit
           CMP   AL,'0'      ;is it< 30h
           JL    INVALID
           CMP   AL,'9'      ;is it> 39h
           JG    A_TO_F
           SUB   AL,30H      ;convert to digit 0-9
           JMP   SHORT ROTATE
 A_TO_F:   CMP   AL,'A'      ;is it < 41h
           JL    INVALID
           CMP   AL,'F'      ;is it > 46h
           JG    INVALID
           SUB   AL,37H      ;convert to digit a-f
ROTATE:
           OR    DL,AL       ;combine with previous digit
           ROR   DX,CL       ;to form binary equivarent
           DEC   CH
           JNZ   L_STR_H     ;if not the last digit, repeat
           MOV   AX,DX       ;store the binary equivarent
           RET
INVALID:   WRITEMSSG  NOT_HEX
           RET
STR_HEX    ENDP


;BINIHEX
;procedure convert binary number in bx
;to hex on console screen
BINIHEX   PROC NEAR
          MOV  CH,4
ROT_BIH:  MOV  CL,4
          ROL  BX,CL
          MOV  AL,BL
          AND  AL,0FH
          ADD  AL,30H
          CMP  AL,3AH
          JL   PRINTIT
```

```
        ADD  AL,7H
PRINTIT:
        MOV  DL,AL
        MOV  AH,2
        INT  21H
        DEC  CH
        JNZ  ROT_BIH
        RET
BINIHEX  ENDP

;LOAD
;load data from disk to ram buffer
LOAD   PROC
       nl
       ;insert zero in buffer following name
       mov  bl,str+1        ;get # of byte read
       mov  bh,0
       mov  [str+bx+2],0    ;zero in to byte
       ;open  file
       mov   dx,offset str+2;addr of name
       mov   al,0           ;file open for reading
       mov   ah,3dh         ;creat file function
       int   21h
       mov   handle,ax      ;stor handle
       jc    error1         ;error  return
       ;read  file
       push   ds            ;save data segment
       mov    bx,handle     ;get  handle file
       mov    cx,numBYTE    ;get  number of byte
       mov    ax,startADDR  ;get  segment buffer to read
       mov    ds,ax
       mov    dx,0          ;offset 0
       mov    ah,3fh        ;write file function
       int    21h
       pop    ds            ;return  data segment
       jc     error1
       cmp    ax,0          ;no at EOF
       je     error1
       ret
       error1: writemssg er      ;massage  file error
       readKBD  char
       ret
LOAD   ENDP


;BINIBYTE
;procedure convert binary number in bL
;to hex on console screen
BINIBYTE   PROC NEAR
           PUSH  CX
          PUSH  BX
          PUSH  DX
          MOV  CH,2
ROT_BYTE:  MOV  CL,4
          ROL  BL,CL
          MOV  AL,BL
          AND  AL,0FH
          ADD  AL,30H
          CMP  AL,3AH
          JL   PRINBYTE
          ADD  AL,7H
PRINBYTE:
          MOV  DL,AL
          MOV  AH,2
          INT  21H
          DEC  CH
          JNZ  ROT_BYTE
          POP   DX
          POP   BX
          POP   CX
          RET
BINIBYTE  ENDP
```

```
CSEG    ENDS
        END
```

บทที่ 4 การทดลองและผลการทดลอง

ชุดสร้างเสียงพูดนี้ประกอบด้วยวงจรย่อยๆ หลายส่วน การประกอบวงจรใช้การ์ดอินเตอร์เฟส IBM เอนกประสงค์ การประกอบจึงมีทั้งใช้การพันสายในส่วนของดิจิตอล และการบักกรีในส่วนอนาล็อก เนื่องจากมีการต่อกับรีซิสเตอร์และคอนเดนเซอร์จำนวนมาก

เมื่อต่อวงจรส่วนฮาดแวร์เสร็จแล้ว จึงทำการเขียนโปรแกรมควบคุมวงจรโดยใช้ภาษาแอสเซมบลี เนื่องจากต้องการความเร็วในการทำงานสูงสุด การทดลองซอฟแวร์และฮาดแวร์จะกระทำไปพร้อมๆ กัน รวมทั้งการแก้ไขปรับปรุงฮาดแวร์และซอฟแวร์ตามความเหมาะสม

เมื่อได้สัญญาณเสียงที่พอใจแล้วก็เก็บลงไฟล์ขอมูล และนำไปเบรินลง EPROM และสร้างบอร์ดขึ้นมาใหม่ ที่ประกอบแต่ส่วนสร้างเสียงเท่านั้นไม่มีส่วนการเก็บเสียง โดยใช้ Z-80 เป็นตัวควบคุมบอร์ดนี้

โครงงานนี้เมื่ออยู่บน IBM PC/XT เสียงที่สร้างขึ้นนั้นมีความชัดเจนพอควร ไม่มีเสียงรบกวนเกิดขึ้น แต่เมื่อสร้างชุดควบคุมที่ใช้ Z-80 เสียงที่ออกมามีเสียงรบกวนเกิดขึ้น ทำให้การรับฟังขาดความชัดเจนไปบ้าง

บทที่ 5 วิจารณ์และสรุปผลการทดลอง

บอร์ดสร้างเสียงพูดที่สร้างเสร็จในโครงงานนี้ มีขีดความสามารถในการพูดที่จำกัดจำนวนคำพูด เนื่องจากมีจำนวนหน่วยความจำขนาดเล็ก ทางแก้ที่จะทำได้ก็คือเพิ่มจำนวนหน่วยความจำขึ้นอีกซึ่งก็จะเสียค่าใช้จ่ายที่เพิ่มขึ้นอีก และผลจากการสร้างเสียงที่ได้ก็ยังไม่สมบูรณ์ เนื่องจากหาอุปกรณ์บางอย่างไม่ได้ตามต้องการ เช่นอุปกรณ์ที่ใช้ในการสุ่มสัญญาณที่ทำขึ้นเองยังมีคุณภาพต่ำ อันเนื่องจากไอซีซีมอสที่เป็นสวิทช์มีขีดจำกัดทางความถี่สูงของการทำงาน ทำให้รูปสัญญาณที่สุ่มผิดเพี้ยน ทางแก้ที่ทำได้ก็คือใช้ไอซีที่เป็นตัวสุ่มสัญญาณโดยเฉพาะที่มีคุณภาพดีกว่า แต่หาในเมืองไทยไม่ได้

ADC0801, ADC0802, ADC0803, ADC0804, ADC0805

National Semiconductor

A to D, D to A

ADC0801, ADC0802, ADC0803, ADC0804, ADC0805 8-Bit μP Compatible A/D Converters

General Description

The ADC0801, ADC0802, ADC0803, ADC0804 and ADC0805 are CMOS 8-bit successive approximation A/D converters which use a differential potentiometric ladder—similar to the 256R products. These converters are designed to allow operation with the NSC800 and INS8080A derivative control bus, and TRI-STATE® output latches directly drive the data bus. These A/Ds appear like memory locations or I/O ports to the microprocessor and no interfacing logic is needed.

A new differential analog voltage input allows increasing the common-mode rejection and offsetting the analog zero input voltage value. In addition, the voltage reference input can be adjusted to allow encoding any smaller analog voltage span to the full 8 bits of resolution.

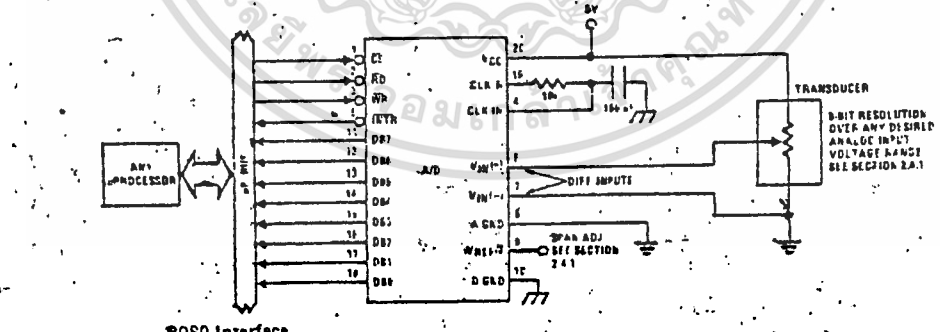
Features

- Compatible with 8080 μP derivatives—no interfacing logic needed – access time – 135 ns
- Easy interface to all microprocessors, or operates "stand alone"
- Differential analog voltage inputs
- Logic inputs and outputs meet both MOS and T²L voltage level specifications
- Works with 2.5V (LM336) voltage reference
- On-chip clock generator
- 0V to 5V analog input voltage range with single 5V supply
- No zero adjust required
- 0.3" standard width 20-pin DIP package
- Operates ratiometrically or with 5 VDC, 2.5 VDC, or analog span adjusted voltage reference

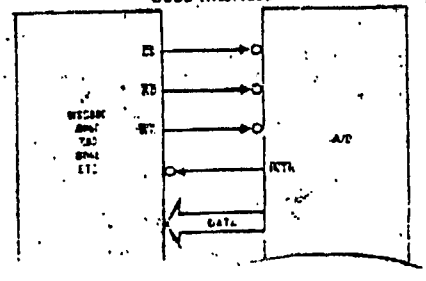
Key Specifications

- Resolution: 8 bits
- Total error: ±1/4 LSB, ±1/2 LSB and ±1 LSB
- Conversion time: 100 μs

Typical Applications

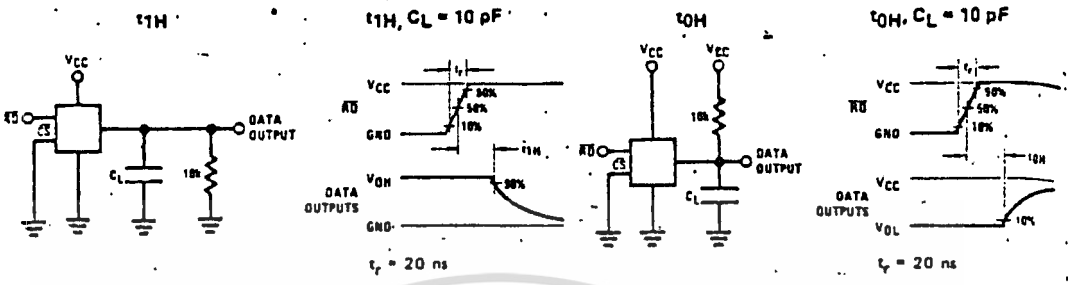


8080 Interface

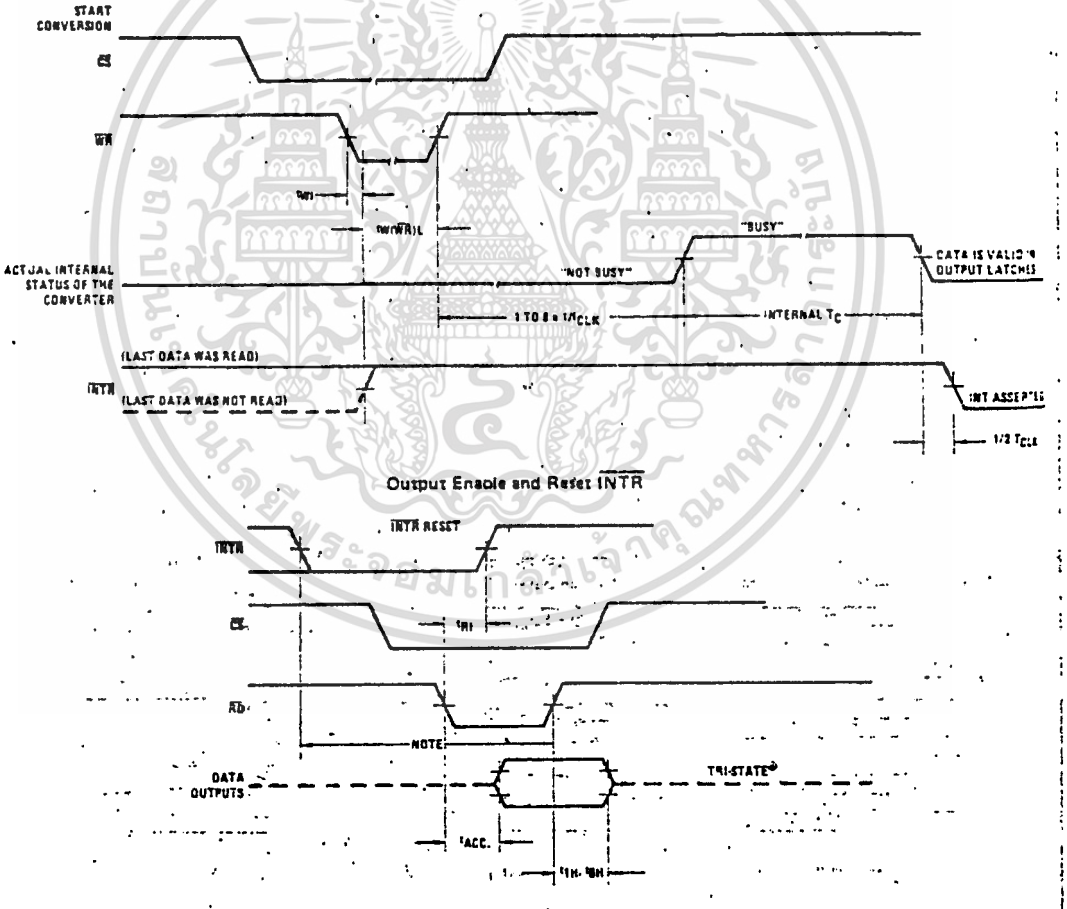


ERROR SPECIFICATION (INCLUDES FULL-SCALE, ZERO ERROR AND NON-LINEARITY)			
PART NUMBER	FULL-SCALE ADJUSTED	VREF/2 = 2.500 VDC (NO ADJUSTMENTS)	VREF/2 = NO CONNECTION (NO ADJUSTMENTS)
ADC0801			
ADC0802		±1/2 LSB	
ADC0803	±1/2 LSB		
ADC0804		±1 LSB	
ADC0805			±1 LSB

TRI-STATE® Test Circuits and Waveforms



Timing Diagrams (All timing is measured from the 50% voltage points)

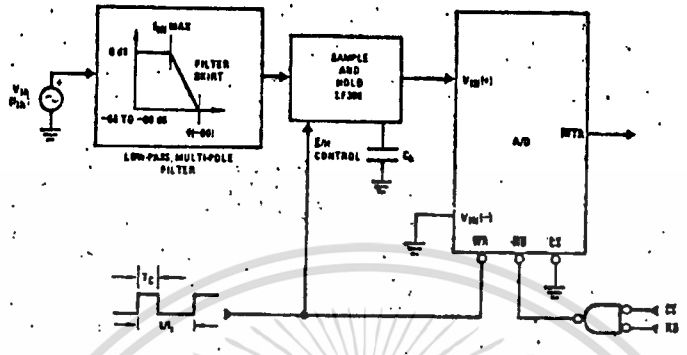


Output Enable and Reset INTR

Note: Read strobe must occur 8 clock periods ($8/f_{CLK}$) after assertion of interrupt to guarantee reset of $\overline{INTR}$.

Typical Applications (Continued)

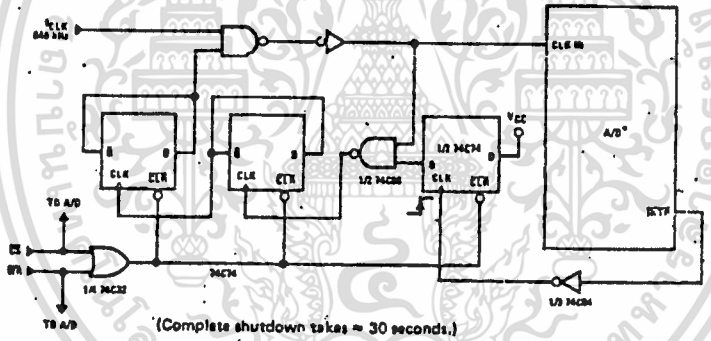
Sampling an AC Input Signal



Note 1: Oversample whenever possible [keep $f_S > 2f(-60)$] to eliminate input frequency folding (aliasing) and to allow for the skirt response of the filter.

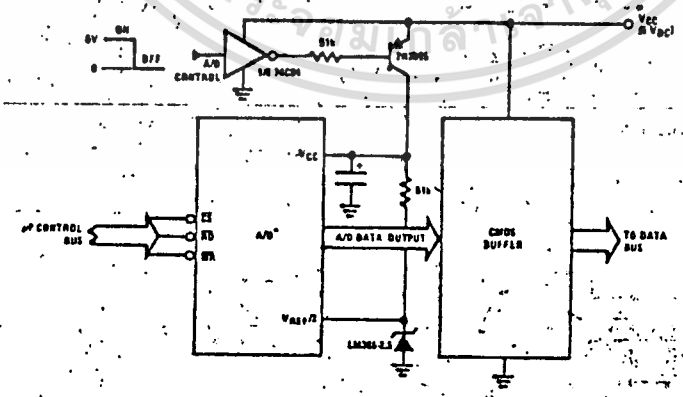
Note 2: Consider the amplitude errors which are introduced within the passband of the filter.

70% Power Savings by Clock Gating



(Complete shutdown takes ≈ 30 seconds.)

Power Savings by A/D and V_{REF} Shutdown



*Use ADC0801, 02, 03 or 05 for lowest power consumption.

Note: Logic inputs can be driven to V_{CC} with A/D supply at zero volts.

Buffer prevents data bus from overdriving outputs of A/D when in shutdown mode.

The standard control inputs of the 8080 ($\overline{IOR}$, $\overline{RD}$ and $\overline{IOW}$, $\overline{WR}$) can be directly connected to the $\overline{RD}$ and $\overline{WR}$ control inputs of the A/D and the bus timing requirements are met to allow both starting the converter and outputting the data onto the data bus. A bus driver should be used for larger microprocessor systems where the data bus leaves the PC board and/or must drive capacitive loads larger than 100 pF.

4.1.1 Sample 8080A CPU Interfacing Circuitry and Program

The following sample program and associated hardware shown in *Figure 10* may be used to input data from the converter to the INS8080A CPU chip set (comprised of the INS8080A microprocessor, the INS8228 system controller and the INS8224 clock generator). For simplicity, the A/D is controlled as an I/O device, specifically an 8-bit bi-directional port located at an arbitrarily chosen port address, E0. The TRI-STATE output capability of the A/D eliminates the need for a peripheral interface device, however address decoding is still required to generate the appropriate $\overline{CS}$ for the converter.

It is important to note that in systems where the A/D converter is 1-of-8 or less I/O mapped devices, no address decoding circuitry is necessary. Each of the 8 address bits (A0 to A7) can be directly used as $\overline{CS}$ inputs—one for each I/O device.

4.1.2 INS8048 Interface

The INS8048 interface technique with the ADC0801 series (see *Figure 11*) is simpler than the 8080A CPU interface. There are 24 I/O lines and three test input lines in the 8048. With these extra I/O lines available, one of the I/O lines (bit 0 of port 1) is used as the chip select signal to the A/D, thus eliminating the use of an external address decoder. Bus control signals $\overline{RD}$, $\overline{WR}$ and $\overline{INT}$ of the 8048 are tied directly to the A/D. The 16 converted data words are stored at on-chip RAM locations from 20 to 2F (Hex). The $\overline{RD}$ and $\overline{WR}$ signals are generated by reading from and writing into a dummy address, respectively. A sample interface program is shown below.

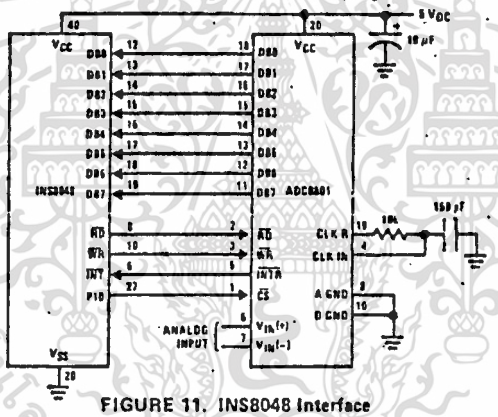


FIGURE 11. INS8048 Interface

SAMPLE PROGRAM FOR *FIGURE 11* INS8048 INTERFACE

```
04 10            JMP     10H          ; Program starts at addr 10
                 ORG     3H
04 50            JMP     50H          ; Interrupt jump vector
                 ORG     10H          ; Main program
99 FE            ANL     P1, #0FEH    ; Chip select
81               MOVX    A, @R1       ; Read in the 1st data
                                      ; to reset the intr
89 01   START:   ORL     P1, #1       ; Set port pin high
B8 20            MOV     R0, #20H     ; Data address
B9 FF            MOV     R1; #0FFH    ; Dummy address
BA 10            MOV     R2, #10H     ; Counter for 16 bytes
23 FF   AGAIN:   MOV     A, #0FFH     ; Set ACC for intr loop
99 FE            ANL     P1, #0FEH    ; Send CS (bit 0 of P1)
91               MOVX    @R1, A       ; Send WR out
05               EN      I            ; Enable interrupt
96 21   LOOP:    JNZ     LOOP         ; Wait for interrupt
EA 1B            DJNZ    R2, AGAIN    ; If 16 bytes are read
00               NOP                  ; go to user's program
00               NOP
                 ORG     50H
81      INDATA:  MOVX    A, @R1       ; Input data, CS still low
A0               MOV     @R0, A       ; Store in memory
18               INC     R0           ; Increment storage counter
89 01            ORL     P1, #1       ; Reset CS signal
27               CLR     A            ; Clear ACC to get out of
93               RETR                 ; the interrupt loop
```

Semiconductor

DAC0800(LMDAC08) 8-Bit Digital-to-Analog Converter

general description

The DAC08 is a monolithic 8-bit high-speed current-output digital-to-analog converter (DAC) featuring typical settling times of 100 ns. When used as a multiplying DAC, monotonic performance over a 40 to 1 reference current range is possible. The DAC08 also features high compliance complementary current outputs to allow differential output voltages of 20 Vp-p with simple resistor loads as shown in *Figure 1*. The reference-to-full-scale current matching of better than ±1 LSB eliminates the need for full scale trims in most applications while the nonlinearities of better than ±0.1% over temperature minimizes system error accumulations.

The noise immune inputs of the DAC08 will accept TTL levels with the logic threshold pin, V_{LC}, pin 1 grounded. Simple adjustments of the V_{LC} potential allow direct interface to all logic families. The performance and characteristics of the device are essentially unchanged over the full ±4.5V to ±18V power supply range; power dissipation is only 33 mW with ±5V supplies and is independent of the logic input states.

The DAC0800L, DAC0802L, DAC0800LC, DAC0801LC and DAC0802LC are a direct replacement for the DAC08, DAC08A, DAC08C, DAC08E and DAC08H, respectively.

features

- Fast settling output current — 100 ns
- Full scale error — ±1 LSB
- Nonlinearity over temperature — ±0.1%
- Full scale current drift — ±10 ppm/°C
- High output compliance — −10V to +18V
- Complementary current outputs
- Interface directly with TTL, CMOS, PMOS and others
- 2 quadrant wide range multiplying capability
- Wide power supply range — ±4.5V to ±18V
- Low power consumption — 33 mW at ±5V
- Low cost

typical applications

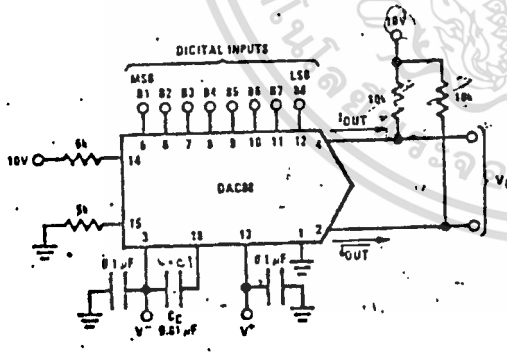
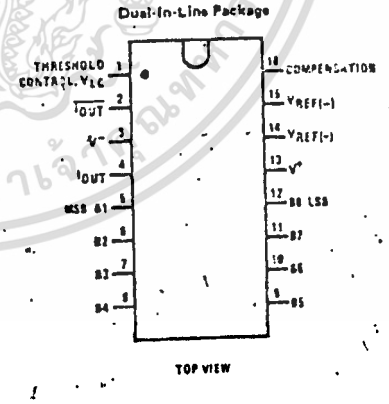


FIGURE 1. ±20 Vp-p Output Digital-to-Analog Converter

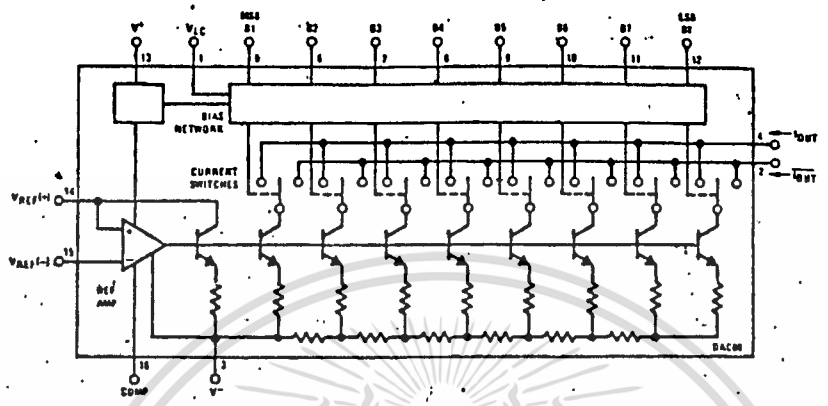
connection diagram



ordering information

NON LINEARITY	TEMPERATURE RANGE	ORDER NUMBERS* D PACKAGE (D16C)		J PACKAGE (J16A)		N PACKAGE (N16A)	
±0.1% FS	−55°C ≤ T_A ≤ +125°C	DAC0802LD	LMDAC08AD	DAC0800LAJ	LMDAC08J		
±0.1% FS	0°C ≤ T_A ≤ +70°C			DAC0802LCJ	LMDAC08HJ	DAC0802LCN	LMDAC08HN
±0.19% FS	−55°C ≤ T_A ≤ +125°C	DAC0800LD	LMDAC08D	DAC0800LJ	LMDAC08J		
±0.19% FS	0°C ≤ T_A ≤ +70°C			DAC0800LCJ	LMDAC08EJ	DAC0800LCN	LMDAC08EN
±[illegible]% FS	0°C ≤ T_A ≤ [illegible]						

*Note. Devices may be ordered by using either order number.



equivalent circuit

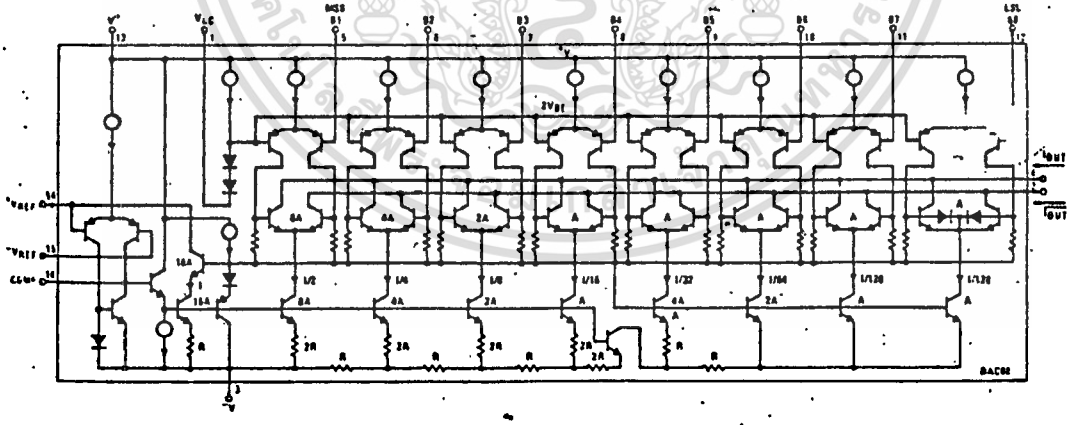


FIGURE 2

typical performance characteristics (Continued)

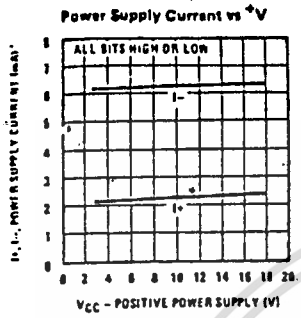


FIGURE 12

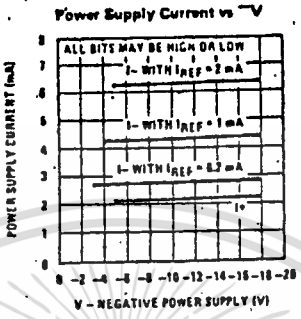


FIGURE 13

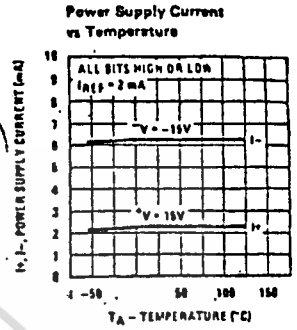
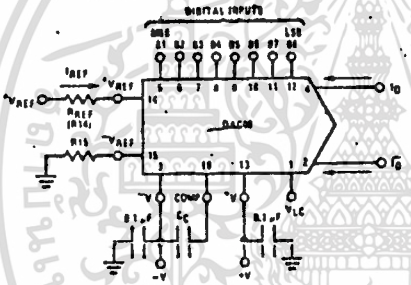


FIGURE 14

typical applications (Continued)



$$I_{FS} = \frac{+V_{REF}}{R_{REF}} \times \frac{255}{256}$$

$I_O + \overline{I_O} = I_{FS}$ for all logic states

For fixed reference, TTL operation, typical values are:
$V_{REF} = 10.000V$
$R_{REF} = 5.000k$
$R15 = R_{REF}$
$C_C = 0.01\ \mu F$
$V_{LC} = 0V$ (Ground)

FIGURE 15. Basic Positive Reference Operation

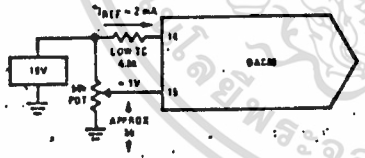
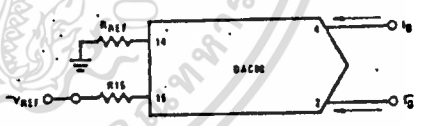


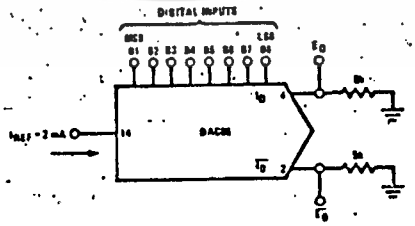
FIGURE 16. Recommended Full Scale Adjustment Circuit



$$I_{FS} = \frac{-V_{REF}}{R_{REF}} \times \frac{255}{256}$$

Note. R_{REF} sets I_{FS}; R15 is for bias current cancellation

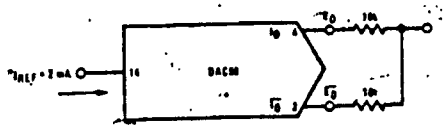
FIGURE 17. Basic Negative Reference Operation



	B1	B2	B3	B4	B5	B6	B7	B8	I_O mA	$\overline{I_O}$ mA	E_O	$\overline{E_O}$
Full Scale	1	1	1	1	1	1	1	1	1.992	0.000	−9.960	0.000
Full Scale−LSB	1	1	1	1	1	1	1	0	1.984	0.008	−9.920	−0.040
Half Scale+LSB	1	0	0	0	0	0	0	1	1.008	0.984	−5.040	−4.920
Half Scale	1	0	0	0	0	0	0	0	1.000	0.992	−5.000	−4.960
Half Scale−LSB	0	1	1	1	1	1	1	1	0.992	1.000	−4.960	−5.000
Zero Scale+LSB	0	0	0	0	0	0	0	1	0.008	[illegible]	[illegible]	[illegible]
Zero Scale	0	0	0	0	0	0	0	0				

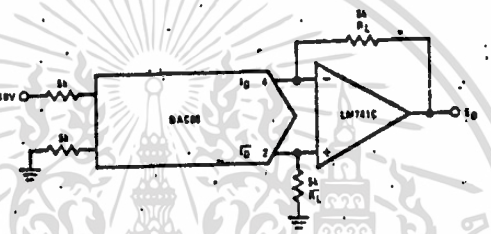
FIGURE 18. Basic Unipolar Negative Operation

...al applications (Continued)



	B1	B2	B3	B4	B5	B6	B7	B8	E_O	$\overline{E_O}$
Pos. Full Scale	1	1	1	1	1	1	1	1	−9.920	+10.000
Pos. Full Scale−LSB	1	1	1	1	1	1	1	0	−9.840	+9.920
Zero Scale+LSB	1	0	0	0	0	0	0	1	−0.080	+0.160
Zero Scale	1	0	0	0	0	0	0	0	0.000	+0.080
Zero Scale−LSB	0	1	1	1	1	1	1	1	+0.080	0.000
Neg. Full Scale+LSB	0	0	0	0	0	0	0	1	+9.920	−9.840
Neg. Full Scale	0	0	0	0	0	0	0	0	+10.000	−9.920

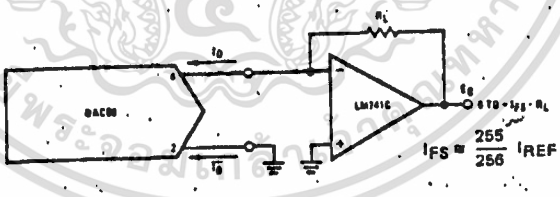
FIGURE 19. Basic Bipolar Output Operation



If $R_L = \overline{R_L}$ within ±0.05%, output is symmetrical about ground

	B1	B2	B3	B4	B5	B6	B7	B8	E_O
Pos. Full Scale	1	1	1	1	1	1	1	1	+9.920
Pos. Full Scale−LSB	1	1	1	1	1	1	1	0	+9.840
(+) Zero Scale	1	0	0	0	0	0	0	0	+0.040
(−) Zero Scale	0	1	1	1	1	1	1	1	−0.040
Neg. Full Scale+LSB	0	0	0	0	0	0	0	1	−9.840
Neg. Full Scale	0	0	0	0	0	0	0	0	−9.920

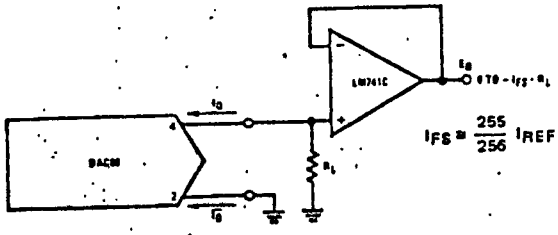
FIGURE 20. Symmetrical Offset Binary Operation



$$I_{FS} \cong \frac{255}{256} I_{REF}$$

For complementary output (operation as negative logic DAC), connect Inverting Input of op amp to $\overline{I_O}$ (pin 2), connect I_O (pin 4) to ground.

FIGURE 21. Positive Low Impedance Output Operation



$$I_{FS} \cong \frac{255}{256} I_{REF}$$

For complementary output (operation as a negative logic DAC) connect non-inverting input of op amp to ground (pin 4) to ground.

FIGURE 22. Negative Low Impedance Output Operation

กิตติกรรมประกาศ

ปริญญานิพนธ์ฉบับนี้สำเร็จขึ้นมาด้วยความกรุณาจาก อาจารย์กิตินันท์ อึ้งสกุล ที่ให้คำแนะนำที่มีค่าและชี้แนวทางที่เป็นประโยชน์ ในการแก้ปัญหาการทำปริญญานิพนธ์ ตลอดระยะเวลาที่ทำปริญานิพนธ์ ซึ่งผู้เขียนขอกราบพระคุณเป็นอย่างสูงไว้ ณ ที่นี้ด้วย

นอกจากนี้ขอแสดงความขอบคุณต่อเพื่อนนักศึกษาทุกๆ คนที่ให้คำปรึกษาในหลายๆ สิ่งที่เป็นประโยชน์ ขอขอบคุณ คุณนิพัทธ์ แซ่ตั้ง และคุณกัญญา จักรปั่น ที่ให้ความสะดวกในการจัดพิมพ์ปริญานิพนธ์ฉบับนี้ไว้ ณ ที่นี้ด้วยเช่นกัน

สังวรณ์ อยู่ยง

พงเดช ชลวิไล

หนังสืออ้างอิง

1. Howard M. Berlin. "Design of Op-Amp Circuit", Howard W. Sam. CO.,INC. ,pp 165-168, 1980

2. Rodnay Zaks. "Microprocessor Interfacig Techniques", pp261-283

3. Walter G. June."IC OP-AMP Cookbook", Howard W. Sam. CO.,INC.,pp 197-200,1974

4. J.L. Flanagan,"Voice of Men and Machine",J.Acoust. Sco.Am, 1972

6. Lewis C. Eggebrecht."Interfaceing to the IBM Personal Computer", Howard W. Sam. CO.,INC.,1983

6. ยืน ภู่วรวณ." เทคโนโลยี ไมโครคอมพิวเตอร์ 16 บิท",ซีเอ็ดยูเคชั่น,2530

7. ยืน ภู่วรวณ."ไมโครโปรเซสเซอร์ ไมโครคอมพิวเตอร์", ซีเอ็ดยูเคชั่น,2524