LAN Based Remote Access Multiple Users (RAMU) Voice Information Retrieval System

Sum Wai Chun

Information Engineering Department

Master of Science

June 1992

Table of Content

Acknowledgement

The author would like to express his sincere gratitude to his project supervisor Dr. Chang Michael for his kind discussion, guidance as well as encouragement throughout the project. He would also like to thank his younger sister Miss S. K. Sum for her great effort to type the manuscript and his classmates Mr. Lee Johnny and Mr. Ng Joe for their constructive advice on the production of graphical pictures in this paper.

Abstract

This paper describes a proposed Interactive Voice Response (IVR) system called RAMU. RAMU uses a LAN clone to support multi-access. Each Voice Response Unit (VRU) of RAMU is equipped with a Telephone Interface Unit (TIU) and an Voice Unit (VU) to serve a remote dial in user. Through DTMF telephone keypad, the user may select and then retrieve his intended voice information from the system. The operating System of RAMU is written in a very flexible manner that it allows application provider to generate his real life application via a menu driven program. In this paper, both the hardware implementation as well as the software design philosophy of RAMU will be discussed and explored.

LAN Based Remote Access Multiple Users (RAMU) Voice Information Retrieval System

I. INTRODUCTION

Electronic mail services are commonly known as mailbox services and can be provided on one's personal computer or by a public services, perhaps through a packet-switching network. The object of an E-mail service is to minimize the phenomenon so called the "telephone tag". Telephone tag is what happens when person A tries to call person B --- there is statistic result showing that over 70% chance of person B not being available especially if it is a business call[1]. A therefore leaves a message for B to return the call. When B does finally return the call, imagine what --- there is over 70% probability that A will not be on seat. This goes on and on until finally they speak to each other.

With an E-mail service, A and B would have access to terminals perhaps on their own desks, which would be connected into a mailbox computer. User A would log into B's mailbox and put the message in the mailbox. When B comes back to work, he would interrogate the mailbox, find A's question, and probably respond to it on the spot; subsequently, the answer will be put into A's mailbox. Later, A will come back and interrogate his mailbox and thus a communication between the two people is established on a non-real time basis.

---

[1] Nigel Haste,"Selecting Voice Mail and Call Processing Facilities", *Telecommunications, September 1991.*

The aforementioned scenario is splendid but it requires rather sophisticated end user equipment --- a terminal with mail box facility. If, on the other hand, a similar situation happens but what needed are just telephone sets, it would be much fantastic and convenient because telephone sets are generally available and that voice messages stored and forwarded are more affectionate and loving.

As a matter of fact, a voice information system with mail facility is a telecommunications-based device, but where the telephone enables communication across distance, voice mail enables communication across time since it is not necessary for the people to be physically there at the same time to achieve meaningful and effective communication. In addition, if such a voice information system allows its users to log in via a touch-tone or DTMF (Dual Tone Multiple Frequencies) telephone, the system itself may be turned to a interactive voice information retrieval system in such a way that using a telephone keypad as terminals, an user can extract, input or manipulate data stored on computer. The idea is not new. The Interactive Voice Response (IVR) machines emerged in US market since early 1980s, and they have now proven themselves to be useful technology with applications in a wide variety of areas.

In this paper, we are going to describe our proposed IVR system called LAN Based Remote Access Multiple Users (RAMU) Voice Information Retrieval System . Both hardware and software design of

RAMU will be explained.

II. SYSTEM OVERVIEW OF RAMU

RAMU uses PC LAN as a clone and a number of voice response units (VRU) to support multi-access. Fig. 1 depicts the overall architecture of RAMU.

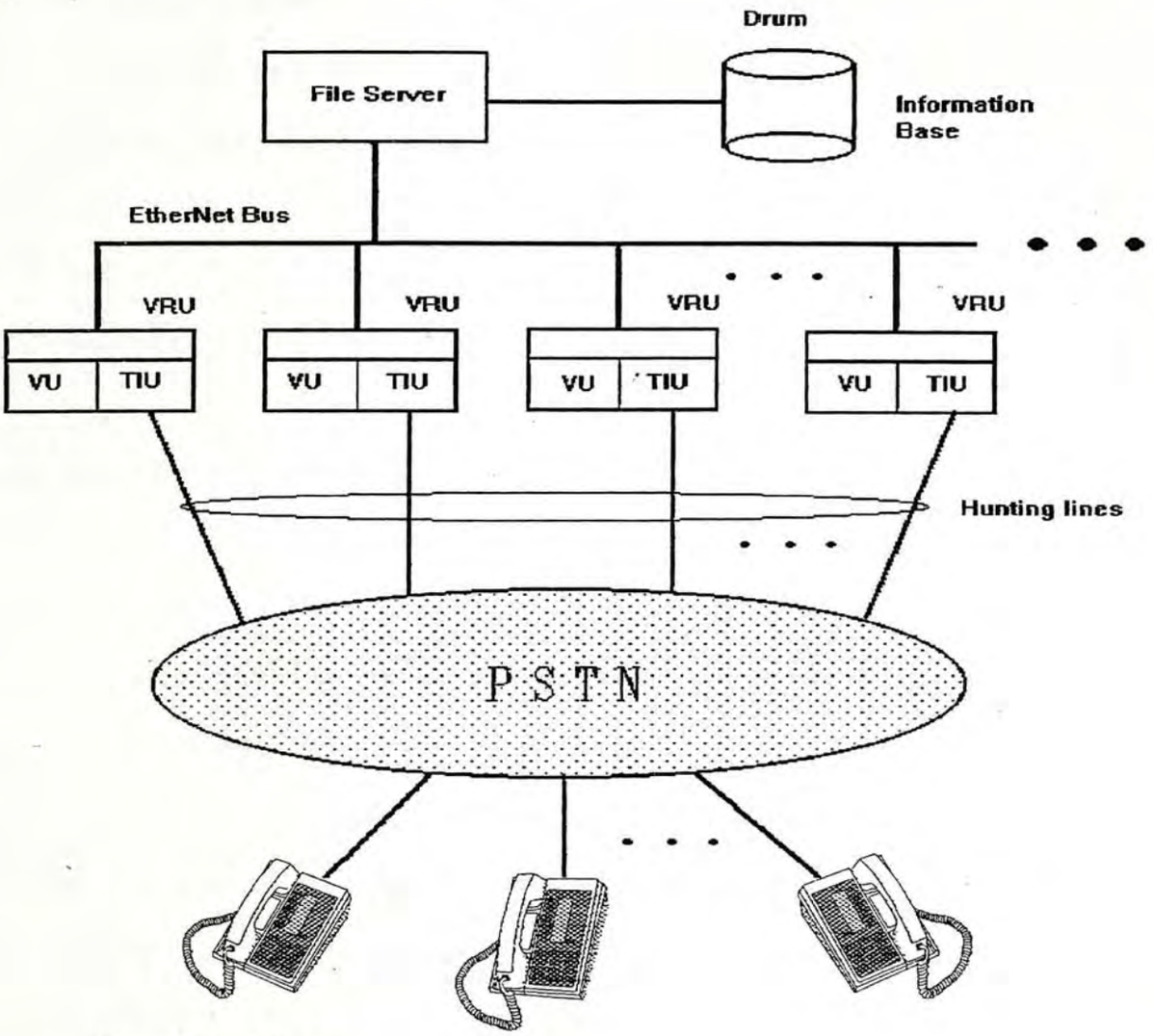


Fig.1 Architecture of RAMU

Each VRU is a PC workstation equipped with a voice unit as well as a telephone interface unit. The telephone interface unit is a tailor made peripheral card. Its main function are to :

1. detect ringing, and upon occurrence of such event, interrupt the CPU,
2. receive DTMF digits from remote user, and
3. control hookflash switch.

More detailed description of this unit will be explored in the next section. As for the voice unit, it is implemented with a VRP-70 voice card. It's main function is to perform low level voice I/O as required.

For the sake of showing how RAMU operates, let us devise a simple application. Suppose our RAMU is tailored to act as a dedicated Voice Information System for the Information Engineering Department of CUHK. It main functions are to announce important message to students and allow them to leave voice mail to the department. Initially, all VRUs are idle. When a call comes, one of the VRUs answers the call. Greeting voice message " Welcome to use our voice system. Please press 1 for course IEG 101, 2 for course IEG 202, 3 for course IEG 303, ... , * to leave voice mail to the IEG department and # to exit " will be played. Just when the first caller follows the menu to select, a second call may comes. If it really happens, other VRU will go active and prepare to serve the call.

So far we have an general overview of the RAMU system. The actual application of the system depends on how the Operating System is designed. We will leave this topic intact until we explain the actual software design later.

In the following section, we are going to explore the hardware of TIU and describe the main characteristic of the VRP-70 voice card.

III. HARDWARE IMPLEMENTATION OF VRU

As mentioned before, a VRU consists of a TIU (Telephone Interface Unit) and a VU (Voice Unit). Fig. 2 illustrates the block diagram of a TIU and its interfacing with VRP-70 and the computer peripheral. Full circuit layout is shown in appendix of this paper.

The 8255 PIA is initialized to operate in mode 1 with strobed Port B input. It does not only responsible for receiving DTMF digit from the DTMF decoder, but also play an important role to control the Hookflash Relay R_f , Voice Play/Record Relay R_{PR} via its various Port C I/O pins.

The Ring Detector uses a photo coupler to detect ringing. It includes a D-Flip flop network to reduce the frequency of ringing pattern and eventually reduce the number of interrupts when a ring comes.

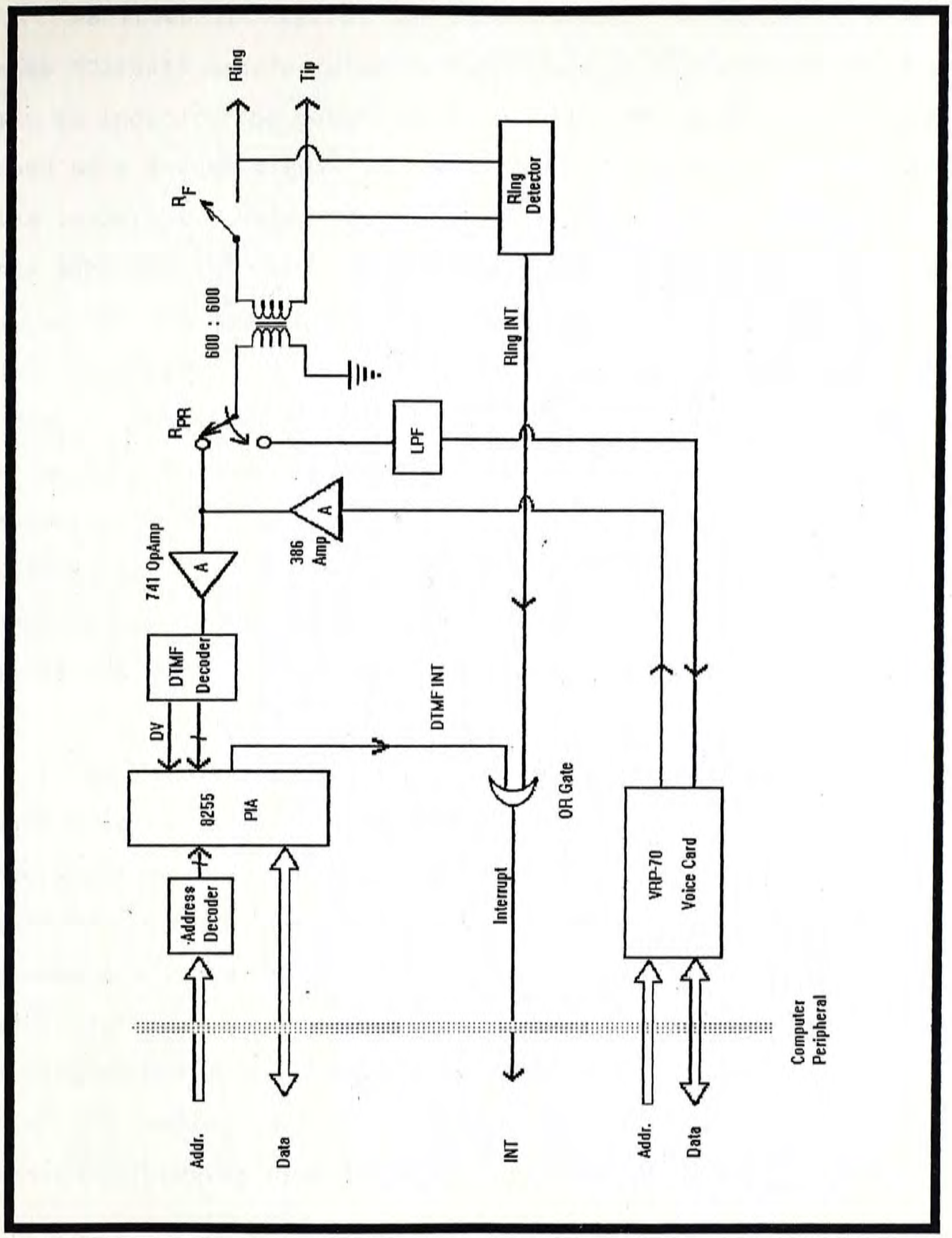


Fig. 2 Block Diagram of TIU

As shown in Fig. 2, the DTMF Decoder is actually a single clip MC145436 manufactured by Motorola. It has a Digit Valid (DV) pin to indicate the reception of a valid DTMF signal. This pin is used as a strobe signal to tell the PIA to generate an interrupt. The interrupt service routine, ISR, will then be activated to get the DTMF digit. Both the DTMF and Ring interrupts are ORed and served by the same ISR. This is so because they will not occur simultaneously. When VRU is waiting for an incoming call, R_f is open. Consequently, only the Ring Detector will generate interrupts in case ringing is received. When the call finally answered, R_f will be closed and a software flag called Machine Idle will be set to False. Further interrupts issued by the TIU will then be recognized as indication of valid DTMF signal. This design saves the number of hardware interrupts used.

The VU, on the other hand, is a VRP-70 voice I/O card. The card handles voice data by DMA and interrupt technology. The VRU operating software can communicate with a resident program to perform the functions of recording, playing or stopping the voice, change the voice sampling rate, or set the memory blocks as the data buffers. One of the distinctive feature of the card is that during recording or playing, the voice data is swapped between the 64K RAM buffer and the disk, thus the maximum continuous recording/playing times depends on the capacity of the hard disk.

Voice output from the voice card will be amplified by a power

amplifier prior to sending to the secondary coil of the 600Ω : 600Ω telecommunication transformer in order to obtain a better audio reception by remote end.

During recording mode, voice from remote user passes a Low Pass Filter. The LPF is indeed a RC network which is included to govern the amount of energy passed to the voice card.

IV. SOFTWARE DESIGN OF RAMU

In our RAMU system, each VRU runs locally a control program called the Application Execute Manager (AEM) to serve a remote dial-in user. Information voice are stored in centralized file server of the LAN clone as shareable files. Consider, for example, if AEM was written in a sequential and step by step manner, having a well defined beginning, middle, and end, it would provide an user with no flexibility to retrieve information in a to and fro way. This is very undesirable because if the user wants to get two pieces of information, he has to dial in twice and selects differently.

A remedy to the above drawback may be the use of an event driven programming technique. Fig. 3 gives a basic idea of how an event-driven program flow look like.

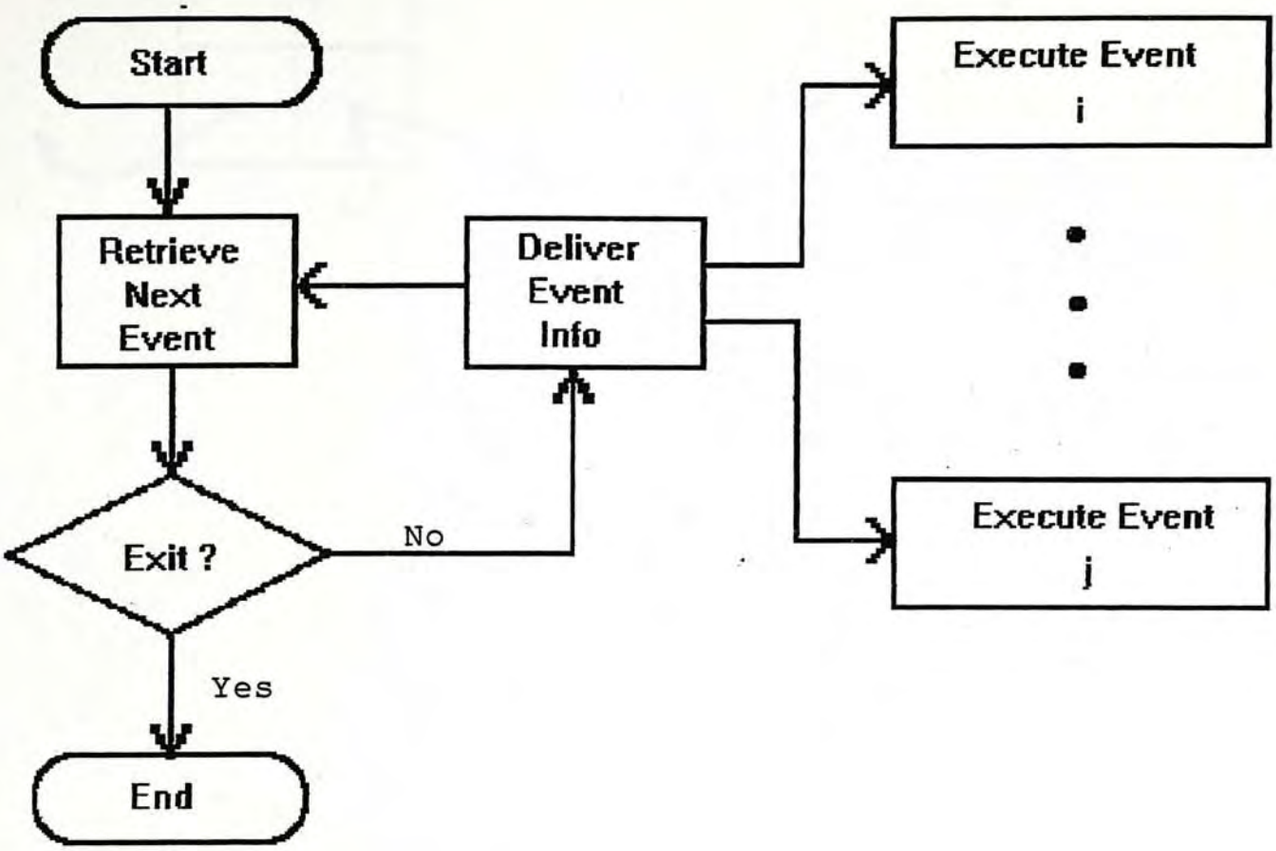


Fig. 3 An Event Driven Program

Employing Event-driven technique alone cannot tackle the problem. In order to give a flexible application to be executed, we require a data structure very similar to a doubly linked nodes list shown in Fig. 4. This kind of structure allows traverse of nodes in either directions and hence induce much flexibility to design a customized application.

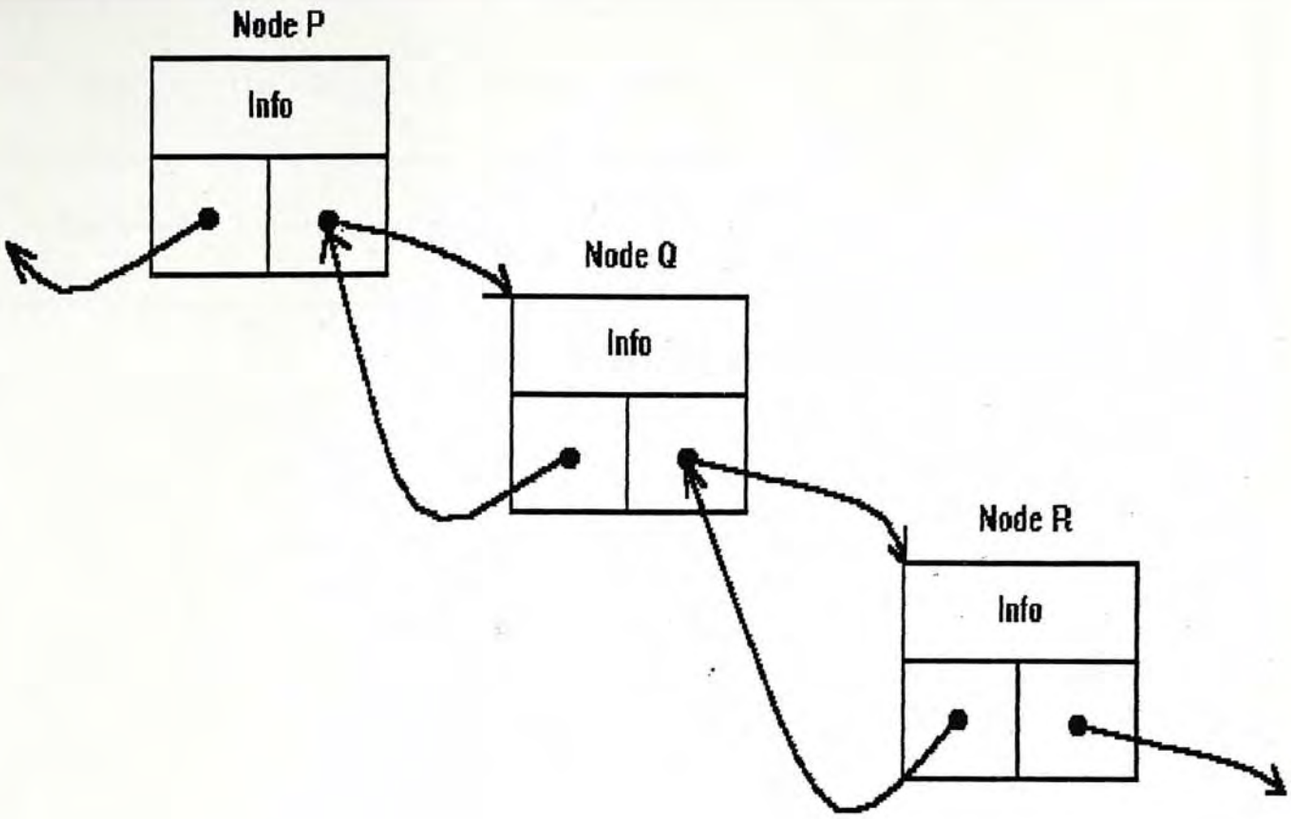


Fig.4 A Doubly Linked Node List

Indeed, in our design, we have created a node structure called an Event Node as depicted in Fig. 5. Each Event Node has its own and unique ID for linkage purpose. On the other hand, every event node will point to 13 event nodes, of which 12 are its child nodes and 1 is its parent node in case it is not a primary node. Some of the child Event Nodes may be identical. Fig. 6 illustrates how event nodes are linked together.

Event Node with ID n

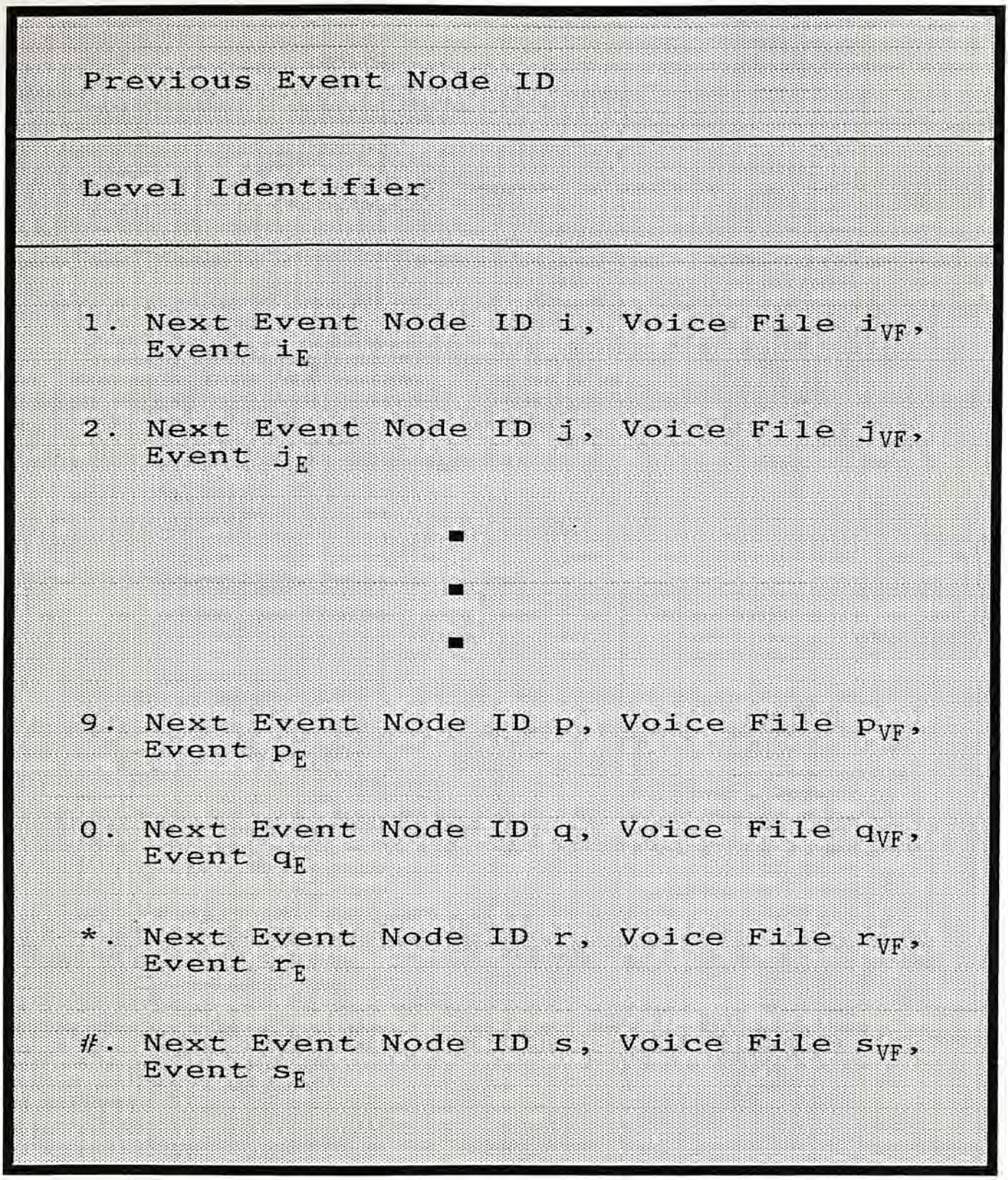


Fig. 5 Structure of an Event Node

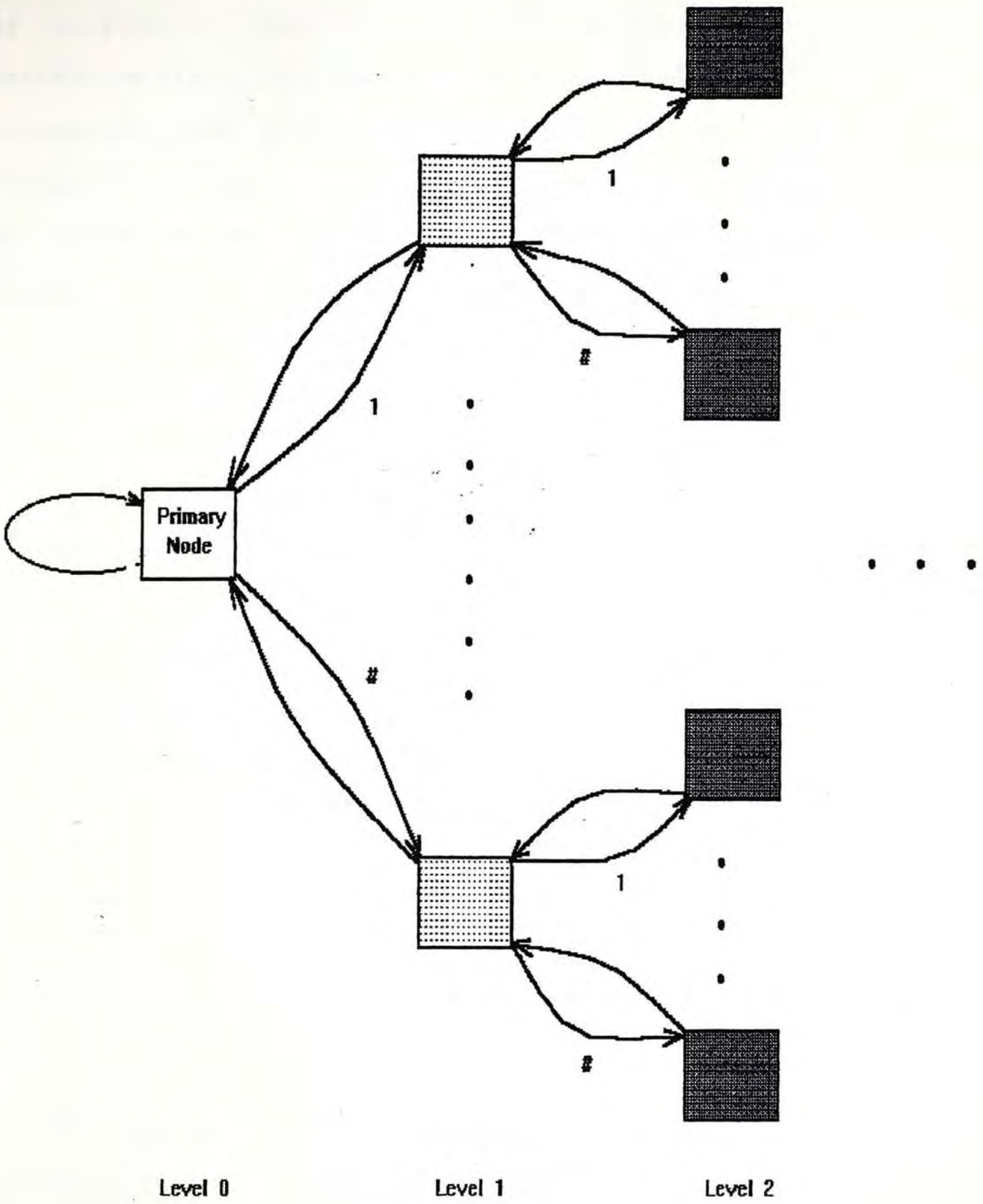


Fig.6 Linkage of Event Nodes

When the AEM program encounters Event Node n, it waits for the user to press a digit on his DTMF telephone keypad. Having received the digit, AEM then plays the corresponding Voice File and executes the event pointed by the digit. The content of the voice file may be instructions to user about the selection of next Event Node or any information desired. On the other hand, the event executed may instruct AEM to go backward, to end the current call, to stay at the present Event Node, to get password from user to perform a specific task or anything else, or even nothing. It all relies on the nature of the application and is very flexible. What the application provider needs to do is just editing the fields of all event nodes involved.

Without Event Node structure, it may be necessary to re-code the program for every application. The program itself will then contain numerous if .....then.....else or case statements to decide what to do in the next steps upon reception of DTMF digit. This is almost equivalent to start programming for every application from the very beginning. The approach is therefore not desirable both from application provider and programming point of view.

The backbone of AEM, however, is an endless loop. To understand how it lessens the burden of program design and facilitates the generation of a new application, we are going to write it in the form of comprehensive pseudo code as in the

following:-

```
repeat
  Wait For Incoming Call;
  A call comes, accept it and clears invalid DTMF digits;
  Load Primary Event Node;
  if not Application Provider wants to break the program or
     error occurs when loading primary Node;
    begin
      Play the Greeting voice;
      Execute the predefined event after call acceptance;
        repeat
          Wait for DTMF digit;
          Depending on the received digit,
            Play voice file i_VF;
            Execute next event i_E;
            Load next event node;
            if event node failure then
              begin
                Play system down message;
                Set call end flag;
              end;
        until call end flag is set;
    end;
  if primary node load failure, inform caller;
  Re-initialize all necessary variables;
until  program break request is received;
```

Another important issue of AEM to permit easy and flexible generation of complicated application is the use of event script interpreter. When AEM encounters an Execute Event i_E instruction, it calls the event script interpreter to parse i_E. In fact i_E is just a string of characters, says 'VMRECORD' (this stands for recording voice mail). If the event script is found, corresponding event will be executed. Consequently, assuming that a well designed set of service scripts is written, an application, simple or complicated, can be provided *without any programming*, and application provider only requires to do some typing with reference to a service scripts library available for the system.

V. SOFTWARE SUPPORT OF RAMU

In the previous section, we have examined the design philosophy of RAMU. A critical issue to make RAMU in operation is how to create Event Nodes and let them link together. In view of this, a program called Application Generator has been coded. It is a menu driven program which allows application provider not only to record all voice response required, but also to generate a series of Event Nodes in linked form. Therefore, an application provider simply invoke the Application generator program to tailor his application, and after that call AEM to make the application in effect.

VI. POTENTIAL APPLICATIONS OF RAMU

It is obvious that application of RAMU, a kind of IVR (Interactive Voice Response System), can be either transaction-based or simple inquiry, also well known as audiotex. Transaction-based IVR needs some sort of manipulation of data, such as transferring money by phone from one bank account to another. Inquiry IVR, however, support only a uni-direction flow of information, from the database to the user via VRU. This audiotex type of IVR provides the same voice menu selections and information to all callers and may be open to anyone with the computer's general database. In the following context, we are going to explore some of the potential applications of IVR.

1. Phone Banking:-

In Hong Kong, some banks have provided operators assisted phone banking services to their customers. These services included money transfer, foreign currencies buying or selling, so on and so forth. With sophisticated protocol developed, phone banking might be one of the most popular applications of IVR in future.

2. Bill Settling:-

Since IVR supports entry of numerical data via DTMF telephone keypad, it can be used by credit card holders to settle bills like town gas, telephone, electricity and so on. Verbal confirmation from VRU to caller may be

used as an acknowledgement.

3. Customer Service:-

This is an application in which IVR substitutes the so-called "complaint desk". Callers to the IVR System enter their custom identification, and go through a menu selection process in order to voice mail a complaint or problem.

4. Enquiry System:-

Large organizations like government can employ IVR to offloading a high percentage of telephone calls they receive requesting routine information. This type of passive application requires no input from the user other than menu selections and requires no manipulation of database.

VII. SUGGESTED FURTHER WORKS

Although RAMU provides a mean for users to input their selections through DTMF keypad, it is unable to serve those users who may not access to a touch-tone telephone. Hence, if RAMU is expanded to include the use of speech recognition as input device to the VRU, then human voice can replace the telephone keypad input with all other operational steps remaining unchanged.

Furthermore, it is not difficult for one to reveal that our RAMU system does not possess call processing capability --- a technology which automatically answers the telephone and then routes callers to the required destination. It is possible that such a capability can be expanded to RAMU by constructing an Intelligent Processing Unit (IPU). The IPU is connected to the EtherNet bus and it talks to other VRUs using the peer to peer communication protocol. In fact, IPU is not necessary a dedicated call processing unit. Conversely, it is a polymorphic unit which performs different tasks to suit for different applications. Needless to say, the polymorphism nature of IPU is brought by installing different set of hardware and software.

VIII. Demonstrated Applications Developed on RAMU

In order to demonstrate the use of our RAMU, two demo applications have been generated with the aid of the Application Generator program. Both of them can be executed with AEM. The first demo application simulates a voice bulletin system. When a VRU answers a call, it prompts the caller to hit a key and then reports to him what digit he has just pressed. It goes without saying that if all responses to caller are replaced by meaningful informative voice, it is necessary a real life system that is applicable in many areas.

The second demo application is one which integrates voice mail

facility. It functions as if it were a voice information retrieval system for the IEG department of CUHK. It provides the department and various lecturers with room to record their latest message remotely. These messages are accessible by incoming callers. In the meantime, callers can leave their verbal messages to respective lecturer's or the department's voice mail pigeon on his own accord and desire if essential.

IX. Conclusion

Our proposed RAMU IVR system actually employs a template concept which presents the application providers with software and hardware platform to meet the basic needs of a voice information retrieval system, but can be customized to fulfill specific needs of the final application. The success lies in the invention of Event Node structure as well as the induction of open nature of the event scripts execution. The idea of event script was indeed stolen from the service script interpreter concept of the tomorrow's Advanced Intelligent Network. It is well believed that IVR systems like RAMU will become one of the popular telecommunication devices in the near future. In addition, the inclusion of speech recognition technology and IPU will further enhance their proliferation.

Bibliography

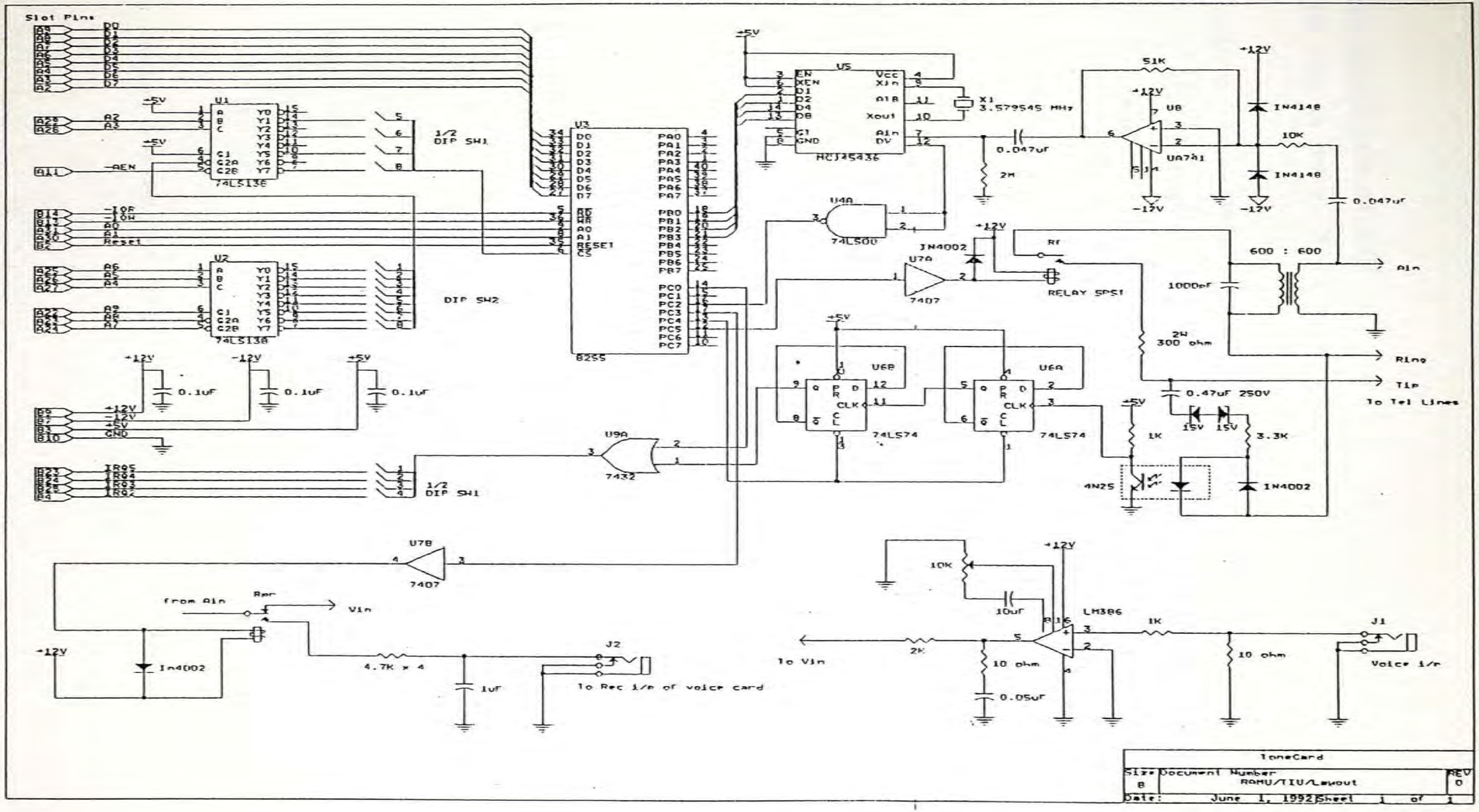
1. Herbert Schildt, "C Complete Reference", McGraw Hill, 1987.

2. Michael Durr, "Networking IBM PCs", QUE, Second Edition, 1987.

3. Ronald L. Krutz, "Interfacing Techniques In Digital Design With Emphasis On Microprocessors", Wiley, 1988.

4. "PC/AT Technical Reference", International Business Corporation, First Edition, 1984.

Appendix 1


Slot Pins
U1
74LS138
U2
74LS138
1/2 DIP SW1
DIP SW2
1/2 DIP SW1
U3
8255
U5
HCJ45436
X1
3.579545 MHz
0.047uF
2M
U4A
74LS00
U7A
7407
U6B
U6A
74LS74
74LS74
U9A
7432
U7B
7407
51K
U8
UA741
IN4148
10K
IN4148
0.047uF
RELAY SPST
600 : 600
1000pF
2W
300 ohm
0.47uF 250V
15V 15V
3.3K
1K
4N25
IN4002
Ain
Ring
Tip
To Tel Lines
0.1uF
0.1uF
0.1uF
from Ain
Rer
Vin
In4002
4.7K x 4
1uF
J2
To Rec i/p of voice card
10K
10uF
LM386
1K
2K
To Vin
10 ohm
0.05uF
10 ohm
J1
Voice i/p
ToneCard
Size B
Document Number RAMU/TIU/Layout
REV 0
Date: June 1, 1992 Sheet 1 of 1

Circuit Operation of TIU

1. As TIU is idle, R_f is open.

2. When a ring comes, AC current from exchange will drive the photo coupler 4N25 on and off alternately. This in turn sends a series of pulse trains into the cascade 74LS74 network. Consequently, processor of VRU is interrupted.

3. After a pre-defined number of ringing (the value is set in Unit ToneCard.Pas), R_f is closed by toggling the pin 5 of port C of 8255 and the call is answered.

4. DTMF signals received will be coupled to the second coil of the 600 Ω : 600 Ω telecommunication transformer and be amplified by μA 741 OpAmp. The signal, having detected by MC 145436, will be strobed into 8255. 8255 then informs the processor of VRU to read it via execution of a hardware interrupt through pin 0 of its port C.

5. R_{PR} is normally in 'Play' position. Hence, voice message from VRP-70, after amplification, can be heard by remote user. If recording is required, AEM will instruct 8255 to toggle pin 3 of its port C so as to set R_{PR} to 'Record' position.

6. Having served the call, R_f will be set to open again.

Dip Switch Settings of TIU

S/W 1 :

	Interrupt
1	IRQ 5
2	IRQ 4
3	IRQ 3
4	IRQ 2

	Address Selected
5	$ XX0
6	$ XX4
7	$ XX8
8	$ XXC

S/W 2 :

	Address Selected
1	$ 20X
2	$ 21X
3	$ 22X
4	$ 23X
5	$ 24X
6	$ 25X
7	$ 26X
8	$ 27X

Example :

The following settings of S/W 1 and 2 will choose IRQ 3 and base address at $270.

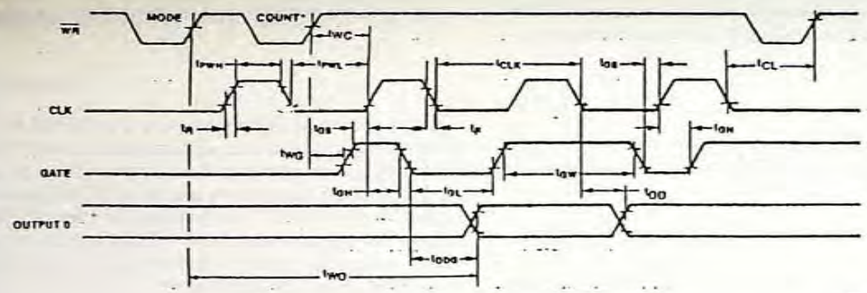
S/W 1 :

	1	2	3	4	5	6	7	8
ON			x		x			
OFF	x	x		x		x	x	x

S/W 2 :

	1	2	3	4	5	6	7	8
ON								x
Off	x	x	x	x	x	x	x	

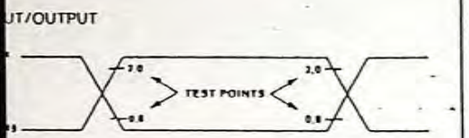
CK AND GATE



231244-17

* Last byte of count being written

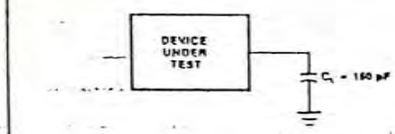
TESTING INPUT, OUTPUT WAVEFORM



231244-18

Testing: Inputs are driven at 2.4V for a logic "1" and 0.45V logic "0." Timing measurements are made at 2.0V for a logic and 0.8V for a logic "0."

A.C. TESTING LOAD CIRCUIT



231244-19

C_L = 150 pF

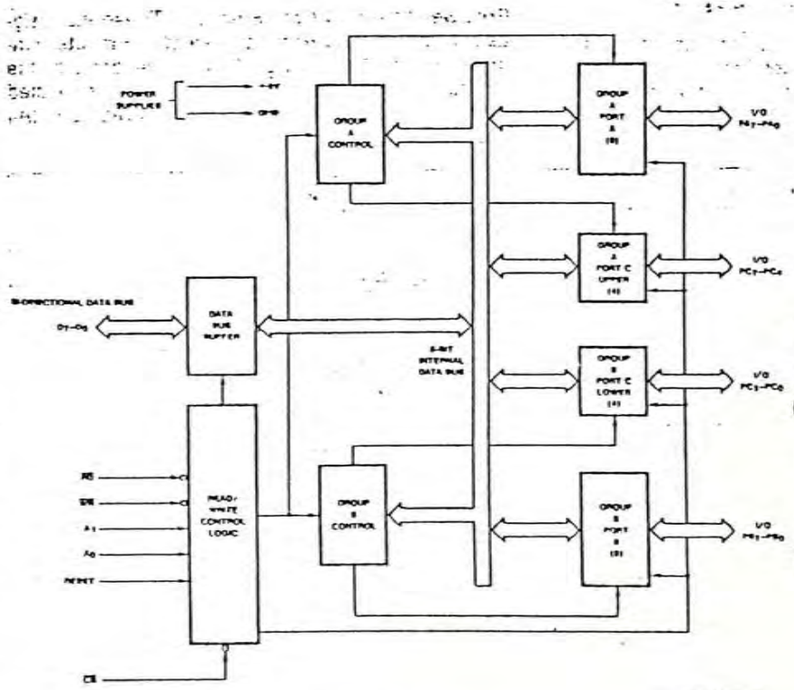
C_L includes jig capacitance

255A/8255A-5
PROGRAMMABLE PERIPHERAL INTERFACE

- MCS-85™ Compatible 8255A-5
- 24 Programmable I/O Pins
- Completely TTL Compatible
- Fully Compatible with Intel Microprocessor Families
- Improved Timing Characteristics
- Direct Bit Set/Reset Capability Easing Control Application Interface
- Reduces System Package Count
- Improved DC Driving Capability
- Available in EXPRESS
  - Standard Temperature Range
  - Extended Temperature Range
- 40 Pin DIP Package or 44 Lead PLCC

(See Intel Packaging: Order Number: 231369)

The Intel 8255A is a general purpose programmable I/O device designed for use with Intel microprocessors. It has 24 I/O pins which may be individually programmed in 2 groups of 12 and used in 3 major modes of operation. In the first mode (MODE 0), each group of 12 I/O pins may be programmed in sets of 4 to be input or output. In MODE 1, the second mode, each group may be programmed to have 8 lines of input or output. Of the remaining 4 pins, 3 are used for handshaking and interrupt control signals. The third mode of operation (MODE 2) is a bidirectional bus mode which uses 8 lines for a bidirectional bus, and 5 lines, borrowing one from the other group, for handshaking.



231308-1

Figure 1. 8255A Block Diagram

231308-2

Figure 2. Pin Configuration

October 1988
Order Number: 231308-001

8255A FUNCTIONAL DESCRIPTION

General

The 8255A is a programmable peripheral interface (PPI) device designed for use in Intel microcomputer systems. Its function is that of a general purpose I/O component to interface peripheral equipment to the microcomputer system bus. The functional configuration of the 8255A is programmed by the system software so that normally no external logic is necessary to interface peripheral devices or structures.

Data Bus Buffer

This 3-state bidirectional 8-bit buffer is used to interface the 8255A to the system data bus. Data is transmitted or received by the buffer upon execution of input or output instructions by the CPU. Control words and status information are also transferred through the data bus buffer.

Read/Write and Control Logic

The function of this block is to manage all of the internal and external transfers of both Data and Control or Status words. It accepts inputs from the CPU Address and Control busses and in turn, issues commands to both of the Control Groups.

($\overline{CS}$)

Chip Select. A "low" on this input pin enables the communication between the 8255A and the CPU.

($\overline{RD}$)

Read. A "low" on this input pin enables the 8255A to send the data or status information to the CPU on the data bus. In essence, it allows the CPU to "read from" the 8255A.

($\overline{WR}$)

Write. A "low" on this input pin enables the CPU to write data or control words into the 8255A.

(A_0 and A_1)

Port Select 0 and Port Select 1. These input signals, in conjunction with the RD and WR inputs, control the selection of one of the three ports or the control word registers. They are normally connected to the least significant bits of the address bus (A_0 and A_1).

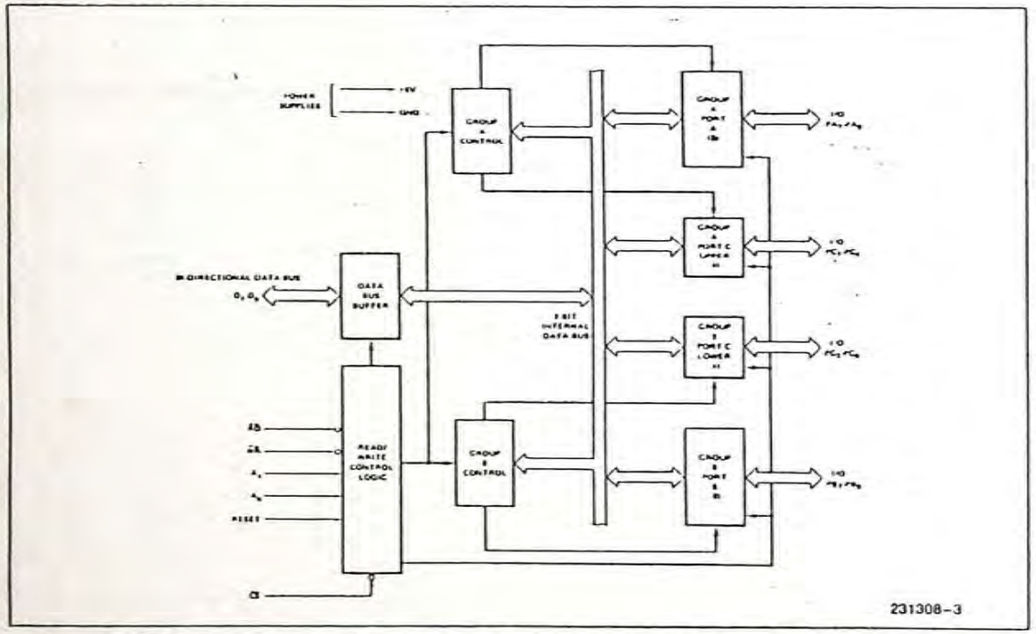


Figure 3. 8255A Block Diagram Showing Data Bus Buffer and Read/Write Control Logic Functions

8255A BASIC OPERATION

A_1	A_0	$\overline{RD}$	$\overline{WR}$	$\overline{CS}$	Input Operation (READ)
0	0	0	1	0	Port A → Data Bus
0	1	0	1	0	Port B → Data Bus
1	0	0	1	0	Port C → Data Bus
					Output Operation (WRITE)
0	0	1	0	0	Data Bus → Port A
0	1	1	0	0	Data Bus → Port B
1	0	1	0	0	Data Bus → Port C
1	1	1	0	0	Data Bus → Control
					Disable Function
X	X	X	X	1	Data Bus → 3-State
1	1	0	1	0	Illegal Condition
X	X	1	1	0	Data Bus → 3-State

(RESET)

Reset. A "high" on this input clears the control register and all ports (A, B, C) are set to the input mode.

Group A and Group B Controls

The functional configuration of each port is programmed by the systems software. In essence, the CPU "outputs" a control word to the 8255A. The control word contains information such as "mode", "bit set", "bit reset", etc., that initializes the functional configuration of the 8255A.

Each of the Control blocks (Group A and Group B) accepts "commands" from the Read/Write Control Logic, receives "control words" from the internal data bus and issues the proper commands to its associated ports.

Control Group A—Port A and Port C upper (C7–C4)

Control Group B—Port B and Port C lower (C3–C0)

The Control Word Register can Only be written into. No Read operation of the Control Word Register is allowed.

Ports A, B, and C

The 8255A contains three 8-bit ports (A, B, and C). All can be configured in a wide variety of functional characteristics by the system software but each has its own special features or "personality" to further enhance the power and flexibility of the 8255A.

Port A. One 8-bit data output latch/buffer and one 8-bit data input latch.

Port B. One 8-bit data input/output latch/buffer and one 8-bit data input buffer.

Port C. One 8-bit data output latch/buffer and one 8-bit data input buffer (no latch for input). This port can be divided into two 4-bit ports under the mode control. Each 4-bit port contains a 4-bit latch and it can be used for the control signal outputs and status signal inputs in conjunction with ports A and B.

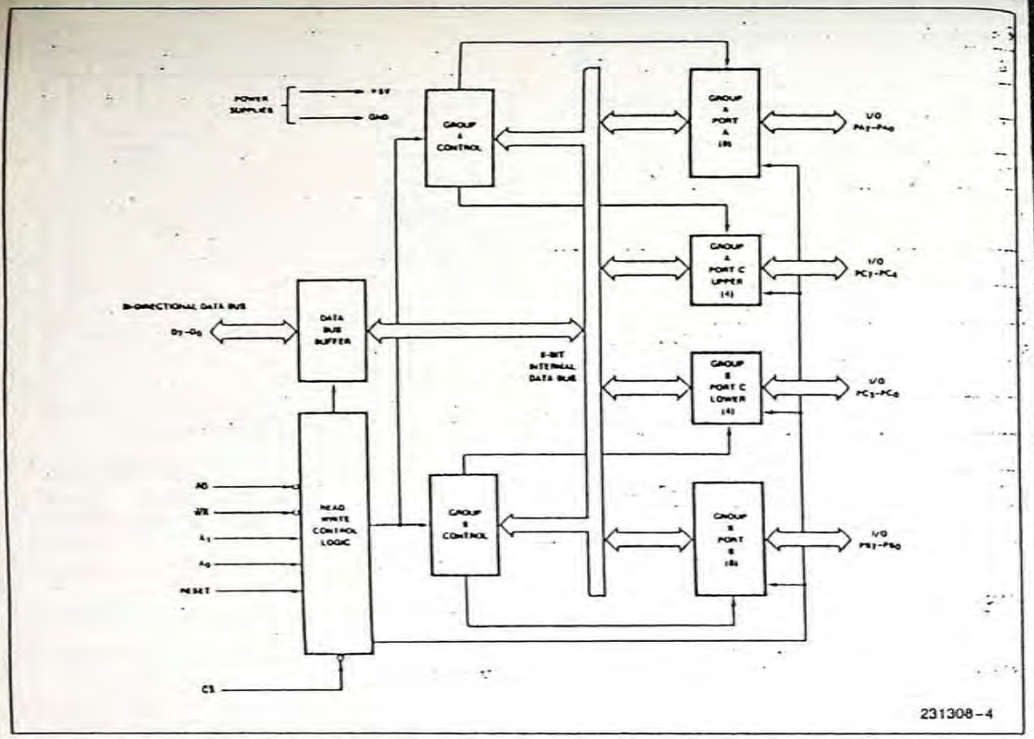


Figure 4. 8225A Block Diagram Showing Group A and Group B Control Functions

Pin Configuration

Pin Names

D_7–D_0	Data Bus (Bi-Directional)
RESET	Reset Input
$\overline{CS}$	Chip Select
$\overline{RD}$	Read Input
$\overline{WR}$	Write Input
A0, A1	Port Address
PA7–PA0	Port A (BIT)
PB7–PB0	Port B (BIT)
PC7–PC0	Port C (BIT)
V_{CC}	+5 Volts
GND	0 Volts

8255A OPERATIONAL DESCRIPTION

Mode Selection

There are three basic modes of operation that can be selected by the system software:

Mode 0—Basic Input/Output

Mode 1—Strobed Input/Output

Mode 2—Bi-Directional Bus

When the reset input goes "high" all ports will be set to the input mode (i.e., all 24 lines will be in the high impedance state). After the reset is removed the 8255A can remain in the input mode with no additional initialization required. During the execution of the system program any of the other modes may be selected using a single output instruction. This allows a single 8255A to service a variety of peripheral devices with a simple software maintenance routine.

The modes for Port A and Port B can be separately defined, while Port C is divided into two portions as required by the Port A and Port B definitions. All of the output registers, including the status flip-flops, will be reset whenever the mode is changed. Modes may be combined so that their functional definition can be "tailored" to almost any I/O structure. For instance; Group B can be programmed in Mode 0 to monitor simple switch closings or display computational results, Group A could be programmed in Mode 1 to monitor a keyboard or tape reader on an interrupt-driven basis.

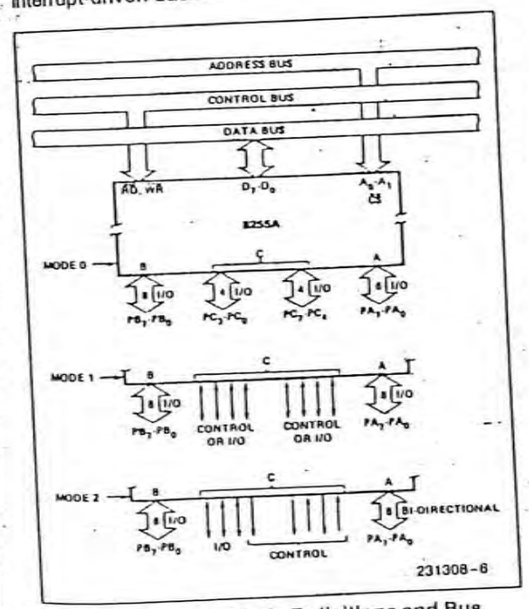


Figure 5. Basic Mode Definitions and Bus Interface

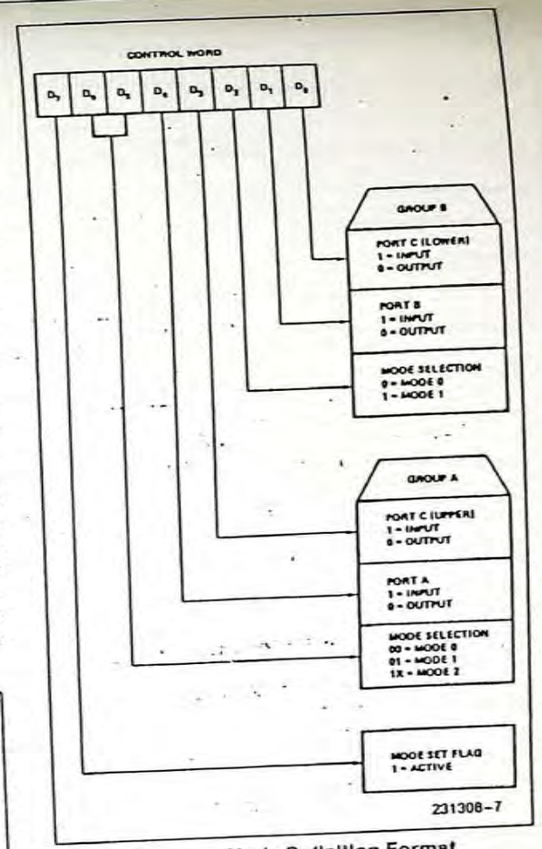


Figure 6. Mode Definition Format

The mode definitions and possible mode combinations may seem confusing at first but after a cursory review of the complete device operation a simple, logical I/O approach will surface. The design of the 8255A has taken into account things such as efficient PC board layout, control signal definition vs PC layout and complete functional flexibility to support almost any peripheral device with no external logic. Such design represents the maximum use of the available pins.

Single Bit Set/Reset Feature

Any of the eight bits of Port C can be Set or Reset using a single OUTput instruction. This feature reduces software requirements in Control-based applications.

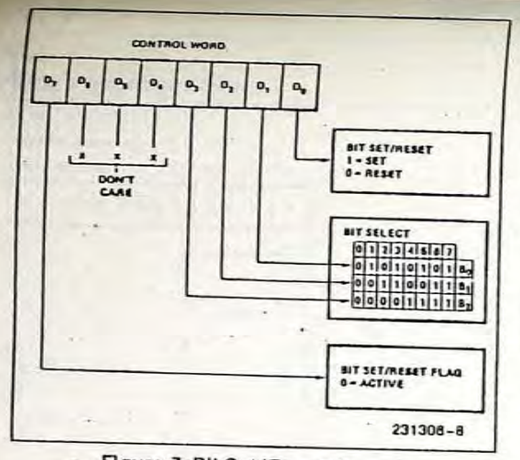


Figure 7. Bit Set/Reset Format

When Port C is being used as status/control for Port A or B, these bits can be set or reset by using the Bit Set/Reset operation just as if they were data output ports.

Interrupt Control Functions

When the 8255A is programmed to operate in mode 1 or mode 2, control signals are provided that can be used as interrupt request inputs to the CPU. The interrupt request signals, generated from port C, can be inhibited or enabled by setting or resetting the associated INTE flip-flop, using the bit set/reset function of port C.

This function allows the Programmer to disallow or allow a specific I/O device to interrupt the CPU without affecting any other device in the interrupt structure.

INTE flip-flop definition:

(BIT-SET)—INTE is set—Interrupt enable

(BIT-RESET)—INTE is RESET—Interrupt disable

NOTE:
All Mask flip-flops are automatically reset during mode selection and device Reset.

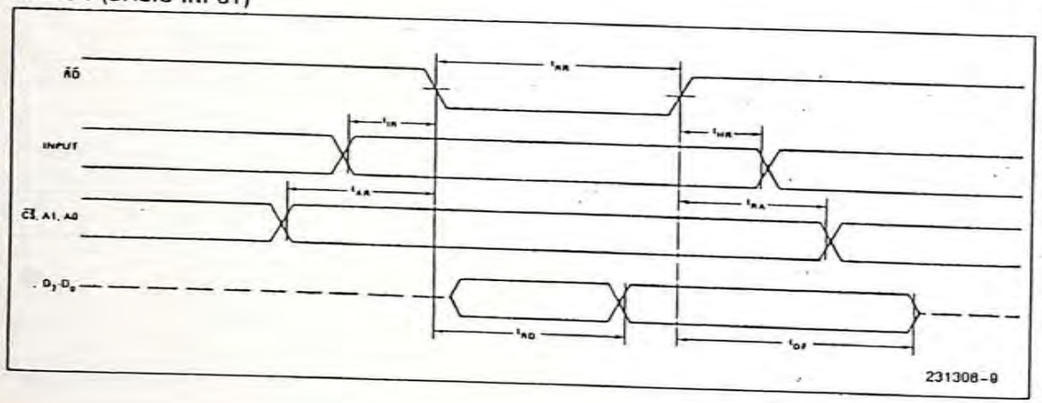
Operating Modes

MODE 0 (Basic Input/Output). This functional configuration provides simple input and output operations for each of the three ports. No "handshaking" is required, data is simply written to or read from a specified port.

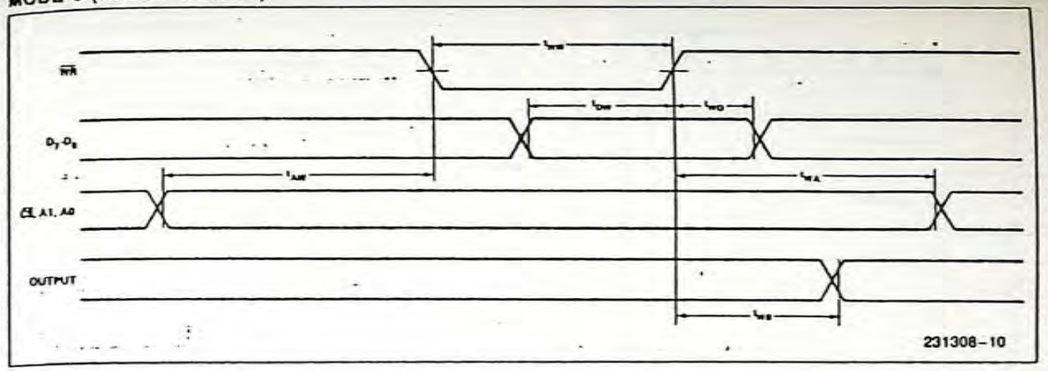
Mode 0 Basic Functional Definitions:

- Two 8-bit ports and two 4-bit ports.
- Any port can be input or output.
- Outputs are latched.
- Inputs are not latched.
- 16 different Input/Output configurations are possible in this Mode.

MODE 0 (BASIC INPUT)



MODE 0 (BASIC OUTPUT)



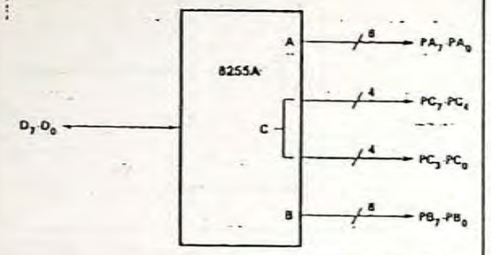
MODE 0 PORT DEFINITION

A		B		Group A			Group B	
D_4	D_3	D_1	D_0	Port A	Port C (Upper)	#	Port B	Port C (Lower)
0	0	0	0	OUTPUT	OUTPUT	0	OUTPUT	OUTPUT
0	0	0	1	OUTPUT	OUTPUT	1	OUTPUT	INPUT
0	0	1	0	OUTPUT	OUTPUT	2	INPUT	OUTPUT
0	0	1	1	OUTPUT	OUTPUT	3	INPUT	INPUT
0	1	0	0	OUTPUT	INPUT	4	OUTPUT	OUTPUT
0	1	0	1	OUTPUT	INPUT	5	OUTPUT	INPUT
0	1	1	0	OUTPUT	INPUT	6	INPUT	OUTPUT
0	1	1	1	OUTPUT	INPUT	7	INPUT	INPUT
1	0	0	0	INPUT	OUTPUT	8	OUTPUT	OUTPUT
1	0	0	1	INPUT	OUTPUT	9	OUTPUT	INPUT
1	0	1	0	INPUT	OUTPUT	10	INPUT	OUTPUT
1	0	1	1	INPUT	OUTPUT	11	INPUT	INPUT
1	1	0	0	INPUT	INPUT	12	OUTPUT	OUTPUT
1	1	0	1	INPUT	INPUT	13	OUTPUT	INPUT
1	1	1	0	INPUT	INPUT	14	INPUT	OUTPUT
1	1	1	1	INPUT	INPUT	15	INPUT	INPUT

MODE CONFIGURATIONS

CONTROL WORD #0

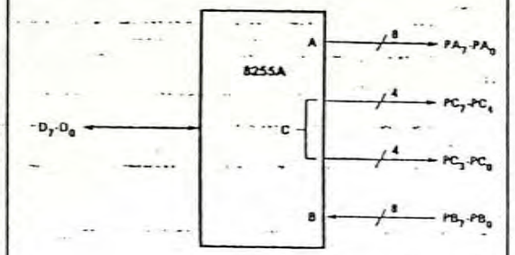
D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
1	0	0	0	0	0	0	0



231308-11

CONTROL WORD #2

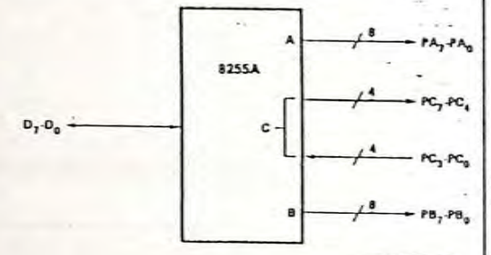
D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
1	0	0	0	0	0	1	0



231308-12

CONTROL WORD #1

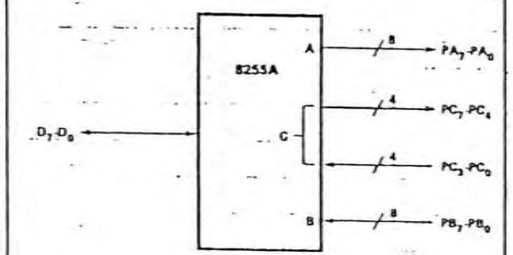
D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
1	0	0	0	0	0	0	1



231308-13

CONTROL WORD #3

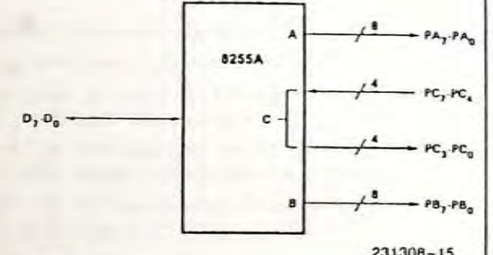
D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
1	0	0	0	0	0	1	1



231308-14

CONTROL WORD #4

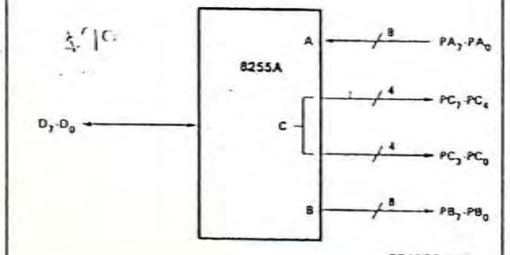
D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
1	0	0	0	1	0	0	0



231308-15

CONTROL WORD #8

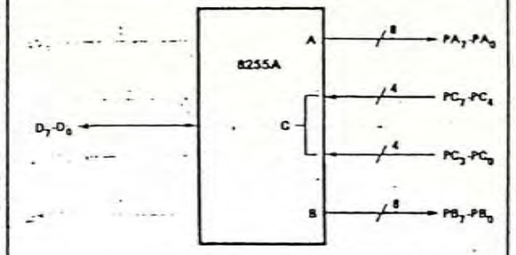
D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
1	0	0	1	0	0	0	0



231308-16

CONTROL WORD #5

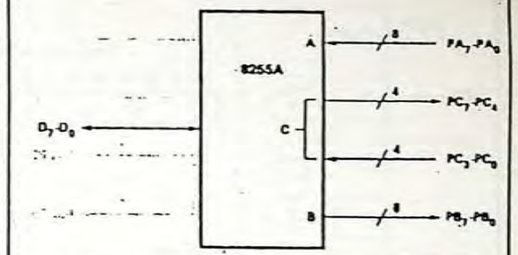
D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
1	0	0	0	1	0	0	1



231308-17

CONTROL WORD #9

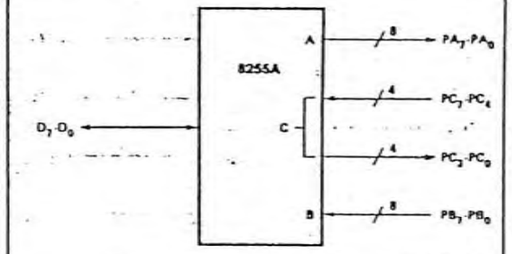
D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
1	0	0	1	0	0	0	1



231308-18

CONTROL WORD #6

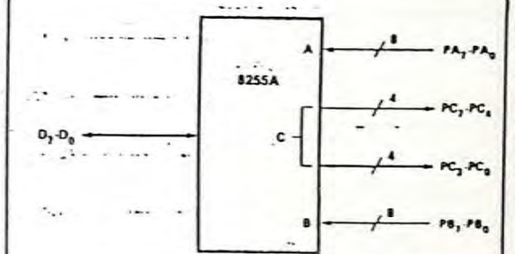
D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
1	0	0	0	1	0	1	0



231308-19

CONTROL WORD #10

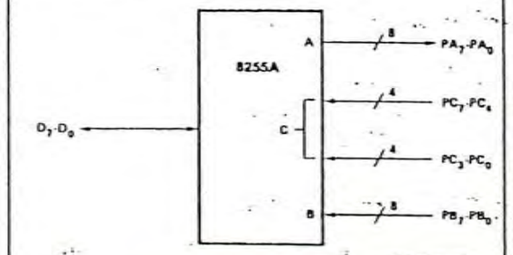
D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
1	0	0	1	0	0	1	0



231308-20

CONTROL WORD #7

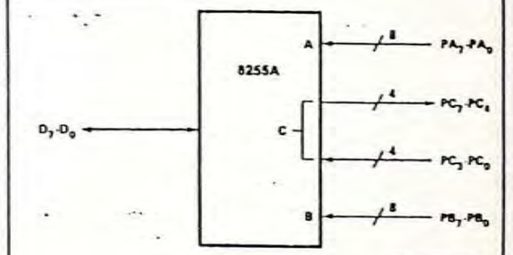
D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
1	0	0	0	1	0	1	1



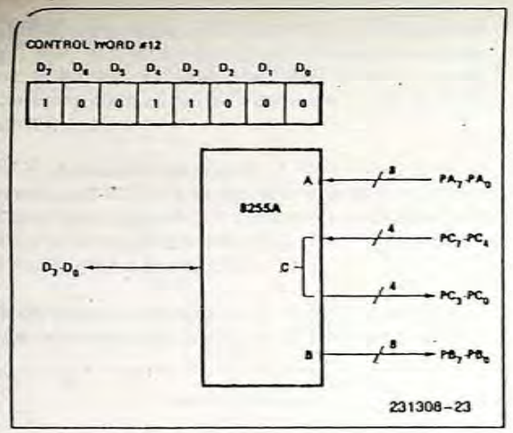
231308-21

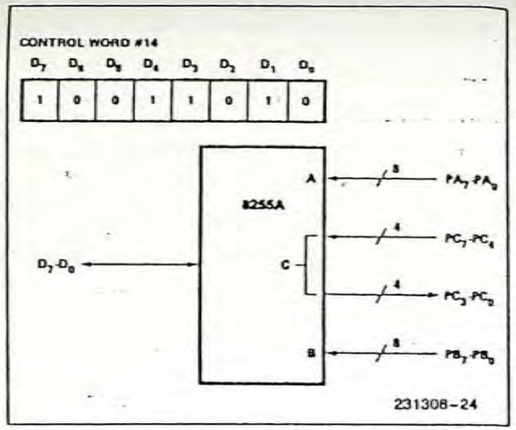
CONTROL WORD #11

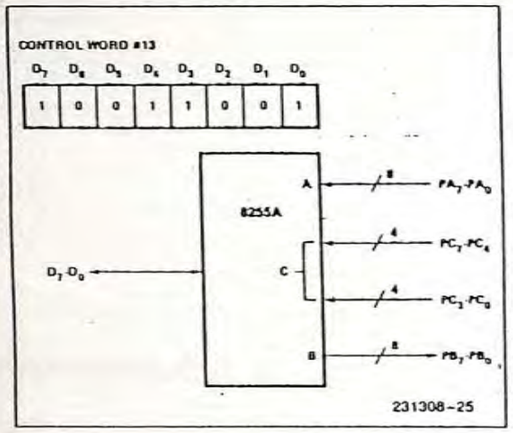
D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
1	0	0	1	0	0	1	1

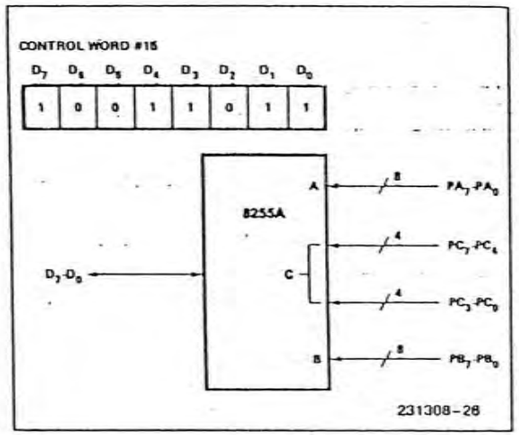


231308-22









Operating Modes

MODE 1 (Strobed Input/Output). This functional configuration provides a means for transferring I/O data to or from a specified port in conjunction with strobes or "handshaking" signals. In mode 1, port A and port B use the lines on port C to generate or accept these "handshaking" signals.

Mode 1 Basic Functional Definitions:

- Two Groups (Group A and Group B)
- Each group contains one 8-bit data port and one 4-bit control/data port.
- The 8-bit data port can be either input or output. Both inputs and outputs are latched.
- The 4-bit port is used for control and status of the 8-bit data port.

Input Control Signal Definition

$\overline{STB}$ (Strobe Input). A "low" on this input loads data into the input latch.

IBF (Input Buffer Full F/F)

A "high" on this output indicates that the data has been loaded into the input latch; in essence, an acknowledgement. IBF is set by STB input being low and is reset by the rising edge of the RD input.

INTR (Interrupt Request)

A "high" on this output can be used to interrupt the CPU when an input device is requesting service. INTR is set by the $\overline{STB}$ is a "one", IBF is a "one" and INTE is a "one". It is reset by the falling edge of $\overline{RD}$. This procedure allows an input device to request service from the CPU by simply strobing its data into the port.

INTE A

Controlled by bit set/reset of PC_4.

INTE B

Controlled by bit set/reset of PC_2.

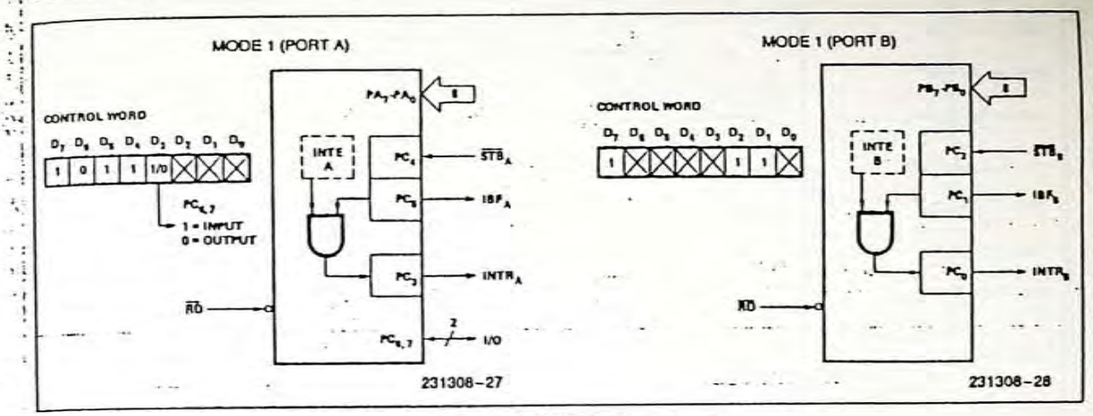


Figure 8. MODE 1 Input

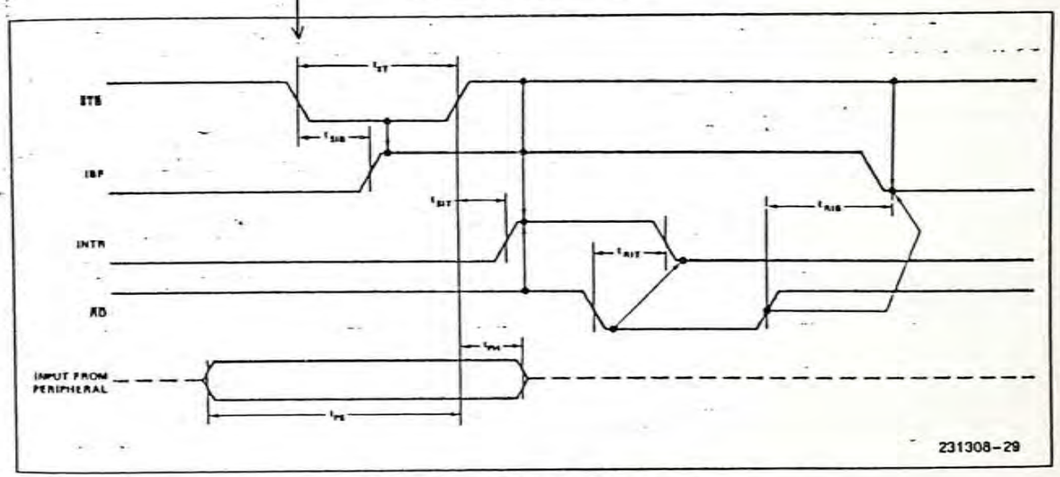


Figure 9. MODE 1 (Strobed Input)

Output Control Signal Definition

$\overline{OBF}$ (Output Buffer Full F/F). The $\overline{OBF}$ output will go "low" to indicate that the CPU has written data out to the specified port. The $\overline{OBF}$ F/F will be set by the rising edge of the $\overline{WR}$ input and reset by $\overline{ACK}$ input being low.

$\overline{ACK}$ (Acknowledge Input). A "low" on this input informs the 8255A that the data from port A or port B has been accepted. In essence, a response from the peripheral device indicating that it has received the data output by the CPU.

INTR (Interrupt Request). A "high" on this output can be used to interrupt the CPU when an output device has accepted data transmitted by the CPU. INTR is set when $\overline{ACK}$ is a "one", $\overline{OBF}$ is a "one", and INTE is a "one". It is reset by the falling edge of $\overline{WR}$.

INTE A

Controlled by bit set/reset of PC_6.

INTE B

Controlled by bit set/reset of PC_2.

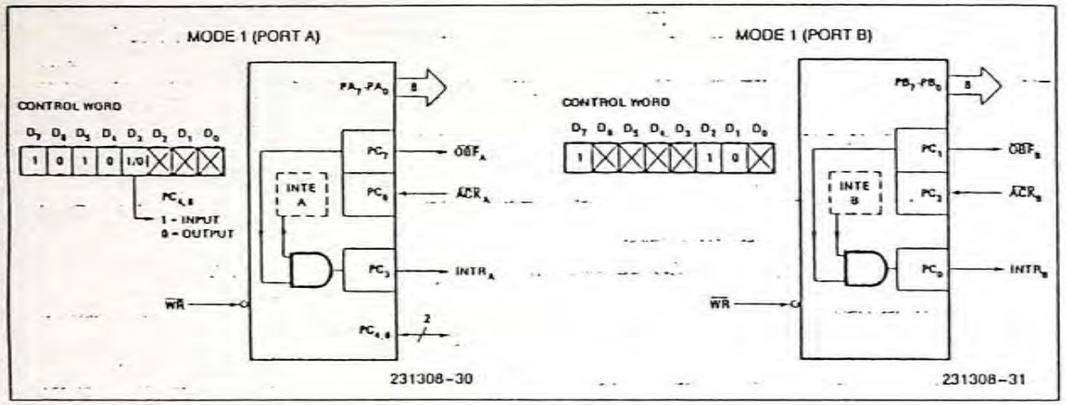


Figure 10. MODE 1 Output

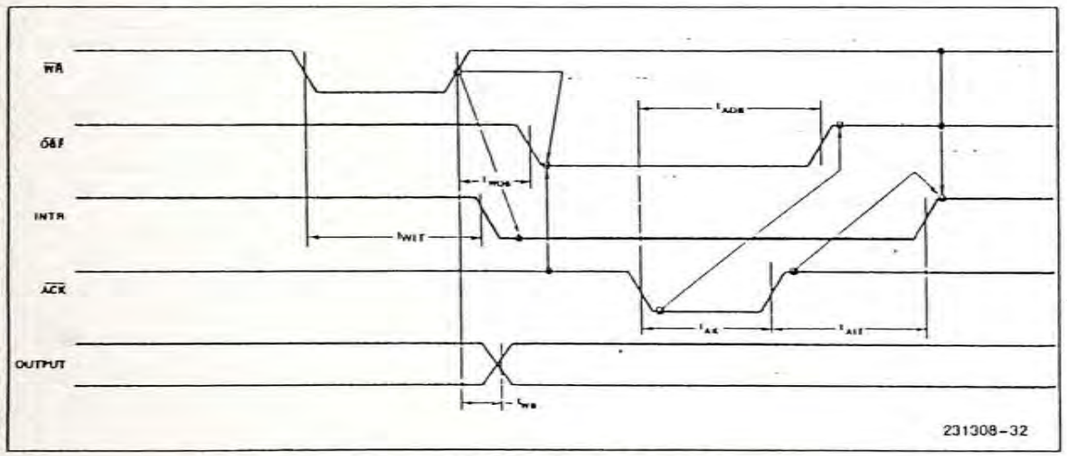


Figure 11. MODE 1 (Strobed Output)

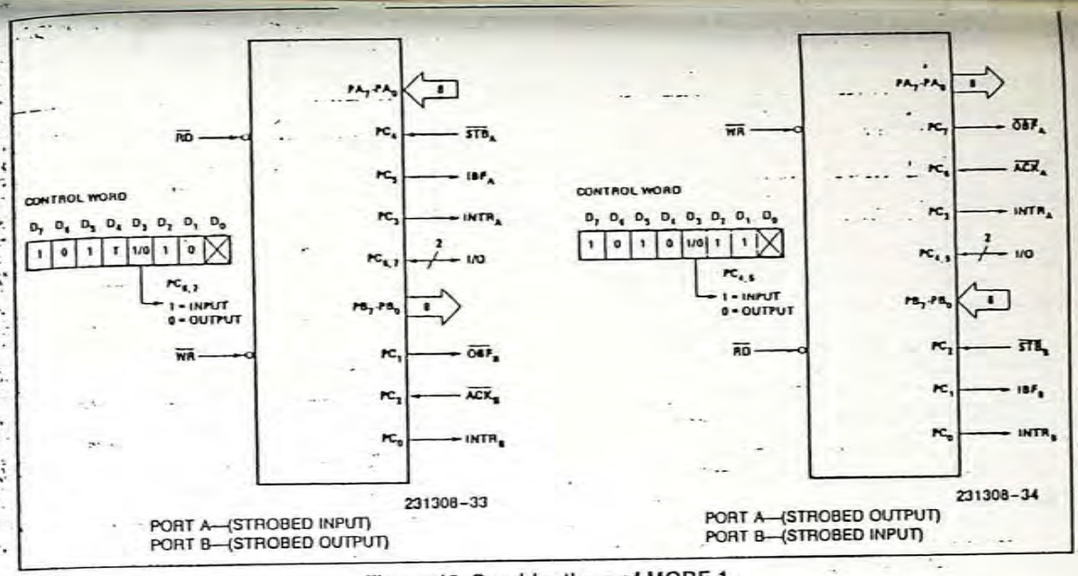


Figure 12. Combinations of MODE 1

Combinations of MODE 1

Port A and Port B can be individually defined as input or output in MODE 1 to support a wide variety of strobed I/O applications.

Operating Modes

MODE 2 (Strobed Bidirectional Bus I/O). This functional configuration provides a means for communicating with a peripheral device or structure on a single 8-bit bus for both transmitting and receiving data (bidirectional bus I/O). "Handshaking" signals are provided to maintain proper bus flow discipline in a similar manner to MODE 1. Interrupt generation and enable/disable functions are also available.

MODE 2 Basic Functional Definitions:

- Used in Group A only.
- One 8-bit, bi-directional bus Port (Port A) and a 5-bit control Port (Port C).
- Both inputs and outputs are latched.
- The 5-bit control port (Port C) is used for control and status for the 8-bit, bi-directional bus port (Port A).

Bidirectional Bus I/O Control Signal Definition

INTR (Interrupt Request). A high on this output can be used to interrupt the CPU for both input or output operations.

Output Operations

$\overline{OBF}$ (Output Buffer Full). The $\overline{OBF}$ output will go "low" to indicate that the CPU has written data out to port A.

$\overline{ACK}$ (Acknowledge). A "low" on this input enables the tri-state output buffer of port A to send out the data. Otherwise, the output buffer will be in the high impedance state.

INTE 1 (The INTE Flip-Flop Associated with OBF). Controlled by bit set/reset of PC_6.

Input Operations

$\overline{STB}$ (Strobe Input). A "low" on this input loads data into the input latch.

...ut Buffer Full F/F). A "high" on this output ...ates that data has been loaded into the input ...ch.

INTE 2 (The INTE Flip-Flop Associated with IBF). Controlled by bit set/reset of PC_4.

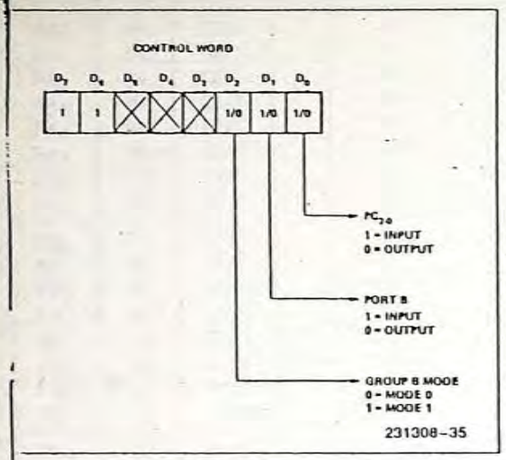


Figure 13. MODE Control Word

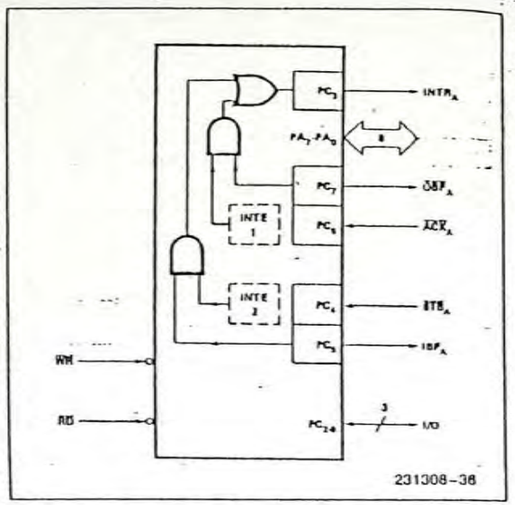
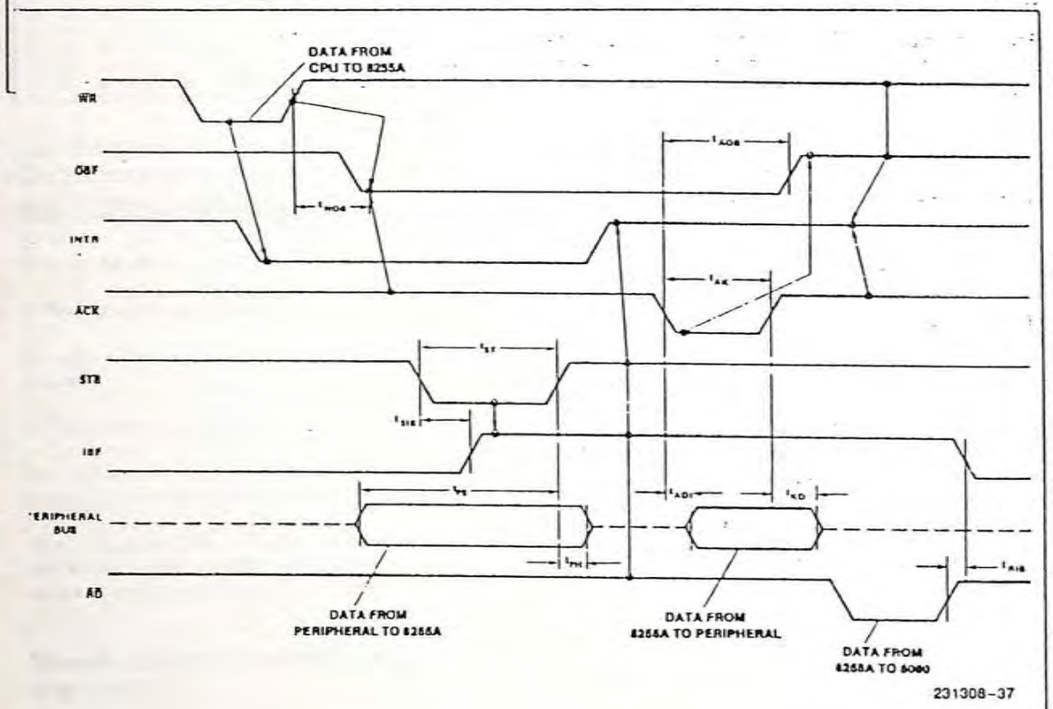


Figure 14. MODE 2



NOTE:
Any sequence where $\overline{WR}$ occurs before $\overline{ACK}$ and $\overline{STB}$ occurs before $\overline{RD}$ is permissible.
$INTR = IBF \cdot \overline{MASK} \cdot \overline{STB} \cdot \overline{RD} + \overline{OBF} \cdot \overline{MASK} \cdot \overline{ACK} \cdot \overline{WR}$)

Figure 15. MODE 2 (Bidirectional)

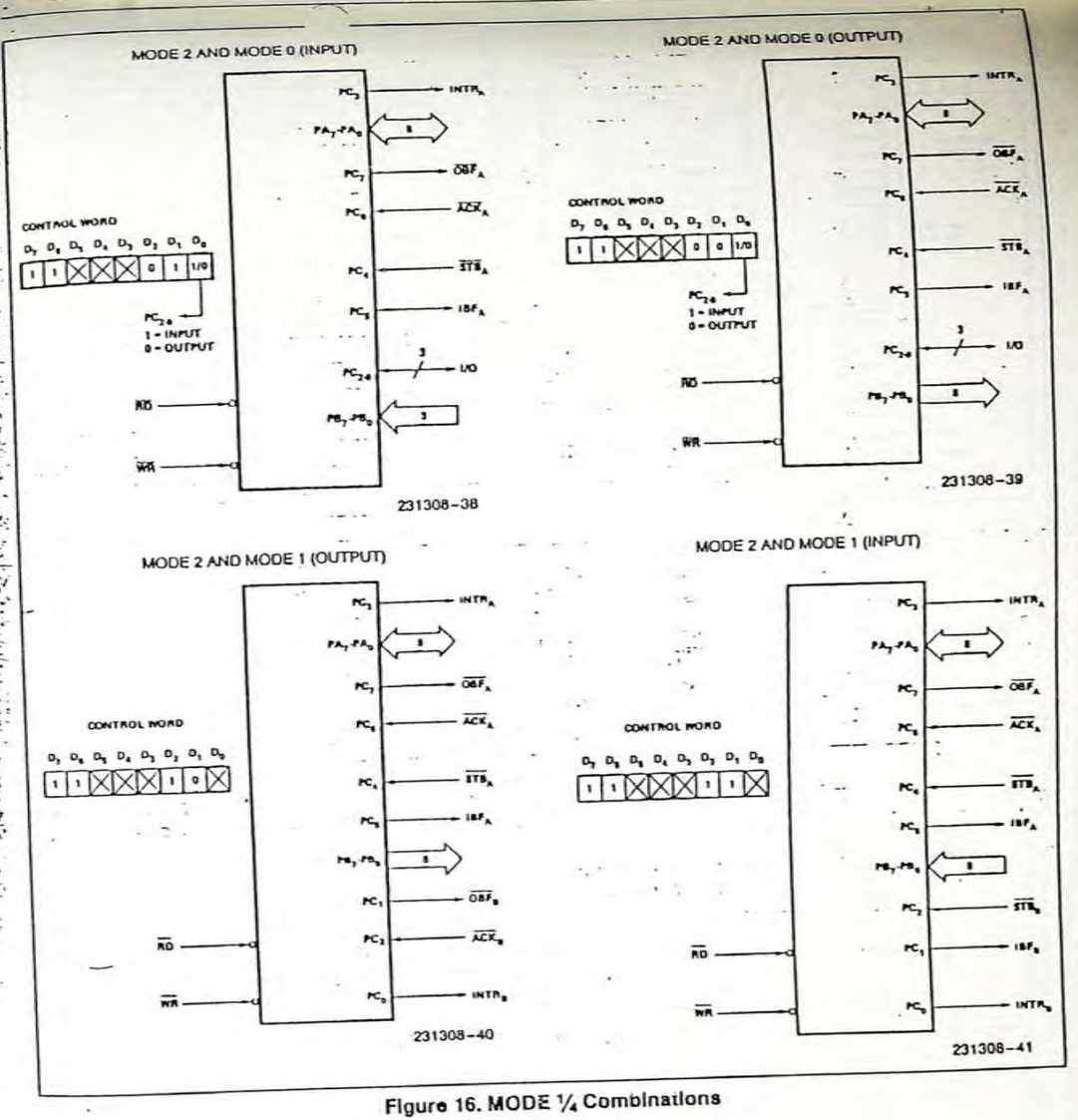


Figure 16. MODE 1/4 Combinations

	MODE 0		MODE 1		MODE 2	
	IN	OUT	IN	OUT	GROUP A ONLY	
PA_0	IN	OUT	IN	OUT	⟷	
PA_1	IN	OUT	IN	OUT	⟷	
PA_2	IN	OUT	IN	OUT	⟷	
PA_3	IN	OUT	IN	OUT	⟷	
PA_4	IN	OUT	IN	OUT	⟷	
PA_5	IN	OUT	IN	OUT	⟷	
PA_6	IN	OUT	IN	OUT	⟷	
PA_7	IN	OUT	IN	OUT	⟷	
PB_0	IN	OUT	IN	OUT	—	MODE 0 OR MODE 1 ONLY
PB_1	IN	OUT	IN	OUT	—	
PB_2	IN	OUT	IN	OUT	—	
PB_3	IN	OUT	IN	OUT	—	
PB_4	IN	OUT	IN	OUT	—	
PB_5	IN	OUT	IN	OUT	—	
PB_6	IN	OUT	IN	OUT	—	
PB_7	IN	OUT	IN	OUT	—	
PC_0	IN	OUT	$INTR_B$	$INTR_B$	I/O	
PC_1	IN	OUT	IBF_B	$\overline{OBF}_B$	I/O	
PC_2	IN	OUT	$\overline{STB}_B$	$\overline{ACK}_B$	I/O	
PC_3	IN	OUT	$INTR_A$	$INTR_A$	$INTR_A$	
PC_4	IN	OUT	$\overline{STB}_A$	I/O	$\overline{STB}_A$	
PC_5	IN	OUT	IBF_A	I/O	IBF_A	
PC_6	IN	OUT	I/O	$\overline{ACK}_A$	$\overline{ACK}_A$	
PC_7	IN	OUT	I/O	$\overline{OBF}_A$	$\overline{OBF}_A$	

Special Mode Combination Considerations

There are several combinations of modes when not all of the bits in Port C are used for control or status. The remaining bits can be used as follows:

If Programmed as Inputs—

All input lines can be accessed during a normal Port C read.

If Programmed as Outputs—

Bits in C upper (PC_7–PC_4) must be individually accessed using the bit set/reset function.

Bits in C lower (PC_3–PC_0) can be accessed using the bit set/reset function or accessed as a threesome by writing into Port C.

Source Current Capability on Port B and Port C

Any set of eight output buffers, selected randomly from Ports B and C can source 1 mA at 1.5 volts. This feature allows the 8255 to directly drive Darlington type drivers and high-voltage displays that require such source current.

Reading Port C Status

In Mode 0, Port C transfers data to or from the peripheral device. When the 8255 is programmed to function in Modes 1 or 2, Port C generates or accepts "hand-shaking" signals with the peripheral device. Reading the contents of Port C allows the programmer to test or verify the "status" of each peripheral device and change the program flow accordingly.

There is no special instruction to read the status information from Port C. A normal read operation of Port C is executed to perform this function.

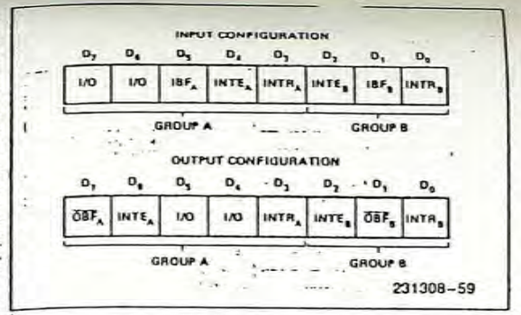


Figure 17. MODE 1 Status Word Format

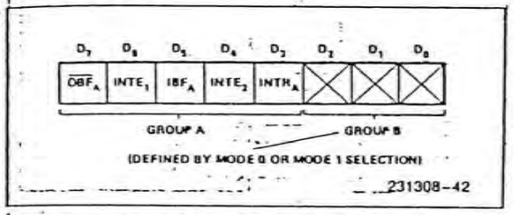


Figure 18. MODE 2 Status Word Format

APPLICATIONS OF THE 8255A

The 8255A is a very powerful tool for interfacing peripheral equipment to the microcomputer system. It represents the optimum use of available pins and is flexible enough to interface almost any I/O device without the need for additional external logic.

Each peripheral device in a microcomputer system usually has a "service routine" associated with it. The routine manages the software interface between the device and the CPU. The functional definition of the 8255A is programmed by the I/O service routine and becomes an extension of the system software. By examining the I/O devices interface characteristics for both data transfer and timing, and matching this information to the examples and tables in the detailed operational description, a control word can easily be developed to initialize the 8255A to exactly "fit" the application. Figures 19 through 25 represent a few examples of typical applications of the 8255A.

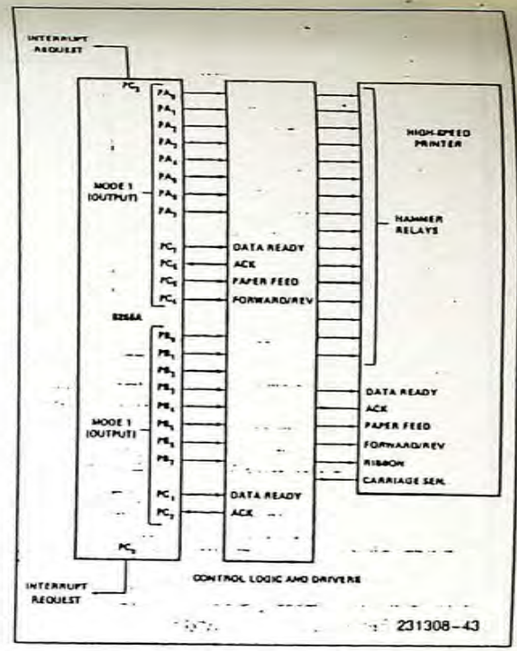


Figure 19. Printer Interface

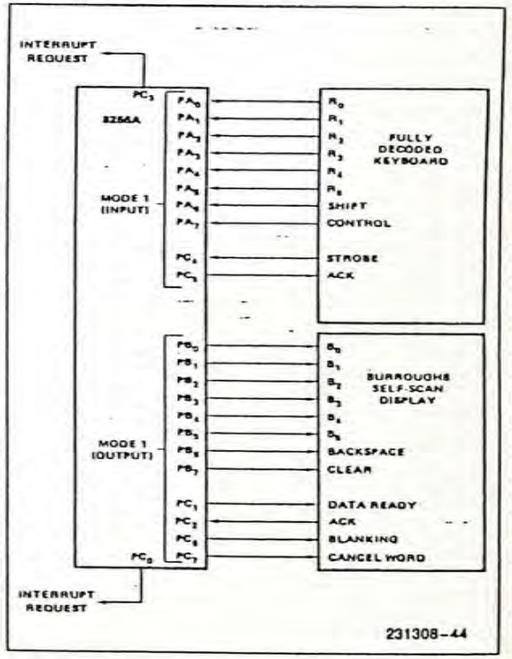


Figure 20. Keyboard and Display Interface

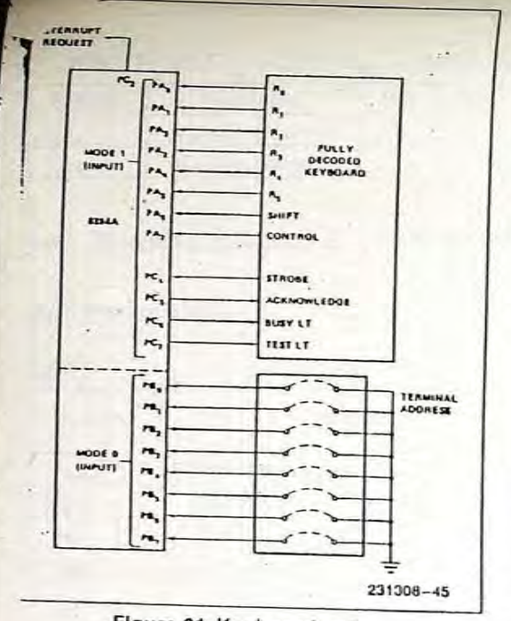


Figure 21. Keyboard and Terminal Address Interface

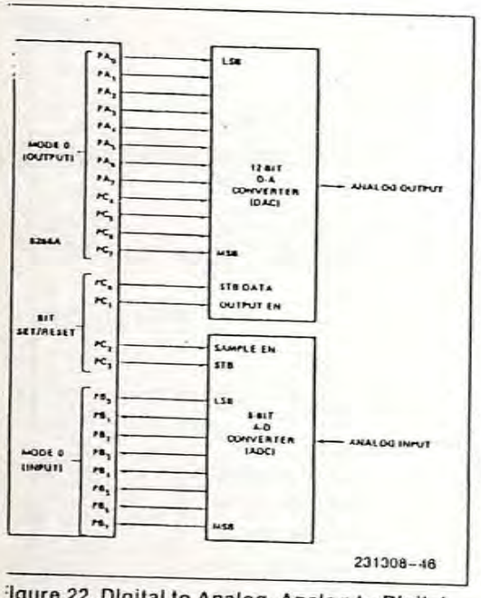


Figure 22. Digital to Analog, Analog to Digital

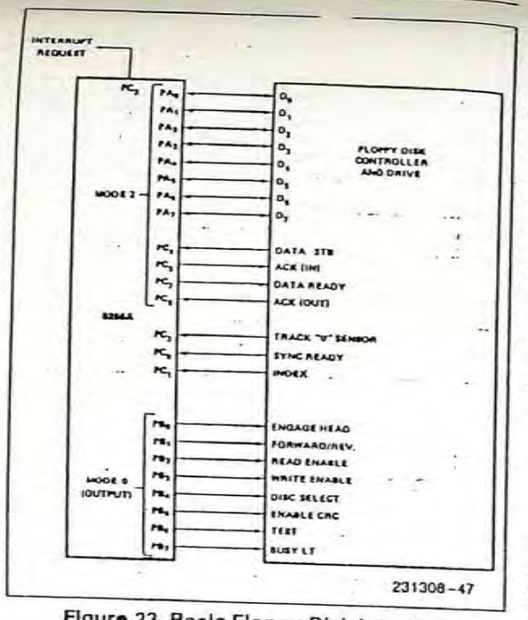


Figure 23. Basic Floppy Disk Interface

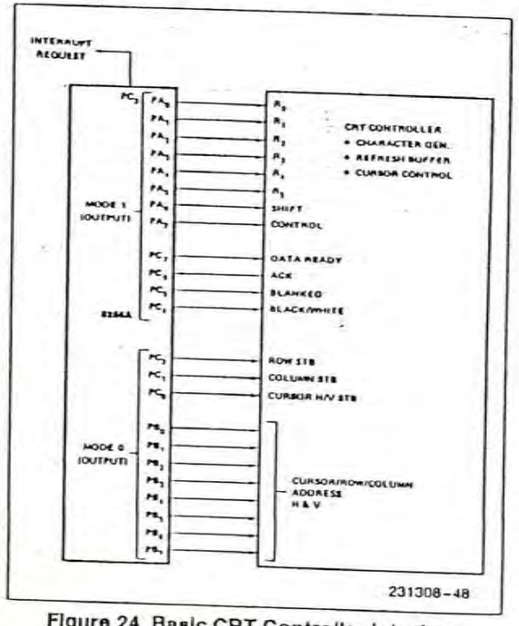


Figure 24. Basic CRT Controller Interface

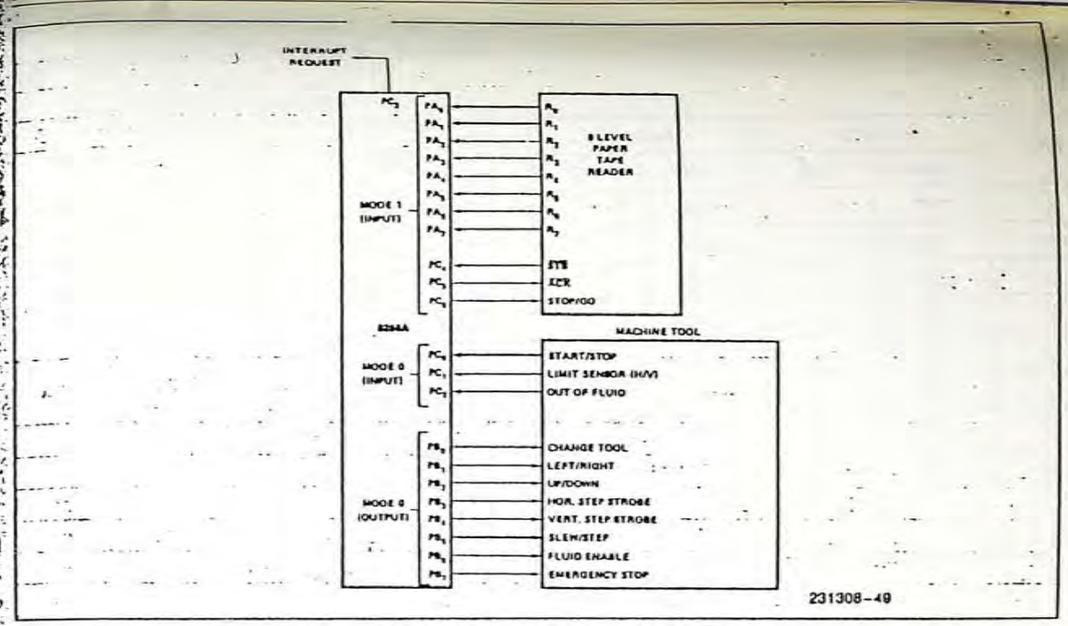


Figure 25. Machine Tool Controller Interface

ABSOLUTE MAXIMUM RATINGS*

Ambient Temperature Under Bias .....0°C to 70°C
Storage Temperature .....−65°C to +150°C
Voltage on Any Pin
with Respect to Ground.....−0.5V to +7V
Power Dissipation .....1 Watt

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

D.C. CHARACTERISTICS T_A = 0°C to 70°C, V_{CC} = +5V ±10%, GND = 0V*

Symbol	Parameter	Min	Max	Unit	Test Conditions
V_{IL}	Input Low Voltage	−0.5	0.8	V	
V_{IH}	Input High Voltage	2.0	V_{CC}	V	
V_{OL} (DB)	Output Low Voltage (Data Bus)		0.45*	V	I_{OL} = 2.5 mA
V_{OL} (PER)	Output Low Voltage (Peripheral Port)		0.45*	V	I_{OL} = 1.7 mA
V_{OH} (DB)	Output High Voltage (Data Bus)	2.4		V	I_{OH} = −400 μA
V_{OH} (PER)	Output High Voltage (Peripheral Port)	2.4		V	I_{OH} = −200 μA
I_{DAR}(1)	Darlington Drive Current	−1.0	−4.0	mA	R_{EXT} = 750Ω; V_{EXT} = 1.5V
I_{CC}	Power Supply Current		120	mA	
I_{IL}	Input Load Current		±10	μA	V_{IN} = V_{CC} to 0V
I_{OFL}	Output Float Leakage		±10	μA	V_{OUT} = V_{CC} to 0.45V

NOTE:
1. Available on any 8 pins from Port B and C.

CAPACITANCE T_A = 25°C, V_{CC} = GND = 0V

Symbol	Parameter	Min	Typ	Max	Unit	Test Conditions
C_{IN}	Input Capacitance			10	pF	f_c = 1 MHz
$C_{I/O}$	I/O Capacitance			20	pF	Unmeasured pins returned to GND

A.C. CHARACTERISTICS T_A = 0°C to 70°C, V_{CC} = +5V ±10%, GND = 0V*

Bus Parameters

READ

Symbol	Parameter	8255A		8255A-5		Unit
		Min	Max	Min	Max	
t_{AR}	Address Stable before READ	0		0		ns
t_{RA}	Address Stable after READ	0		0		ns
t_{RR}	READ Pulse Width	300		300		ns
t_{RD}	Data Valid from READ(1)		250		200	ns
t_{DF}	Data Float after READ	10	150	10	100	ns
t_{RV}	Time between READs and/or WRITEs	850		850		ns

WRITE

Symbol	Parameter	8255A		8255A-5		Unit
		Min	Max	Min	Max	
t_{AW}	Address Stable before WRITE	0		0		ns
t_{WA}	Address Stable after WRITE	20		20		ns
t_{WW}	WRITE Pulse Width	400		300		ns
t_{DW}	Data Valid to WRITE (T.E.)	100		100		ns
t_{WD}	Data Valid after WRITE	30		30		ns

OTHER TIMINGS

Symbol	Parameter	8255A		8255A-5		Unit
		Min	Max	Min	Max	
t_{WB}	WR = 1 to Output(1)		350		350	ns
t_{IR}	Peripheral Data before RD	0		0		ns
t_{HR}	Peripheral Data after RD	0		0		ns
t_{AK}	ACK Pulse Width	300		300		ns
t_{ST}	STB Pulse Width	500		500		ns
t_{PS}	Per. Data before T.E. of STB	0		0		ns
t_{PH}	Per. Data after T.E. of STB	180		180		ns
t_{AD}	ACK = 0 to Output(1)		300		300	ns
t_{KD}	ACK = 1 to Output Float	20	250	20	250	ns

A.C. CHARACTERISTICS (Continued)

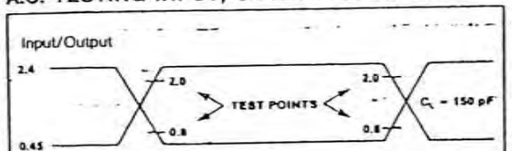
OTHER TIMINGS (Continued)

Symbol	Parameter	8255A		8255A-5		Unit
		Min	Max	Min	Max	
t_{WOB}	WR = 1 to OBF = 0(1)		650		650	ns
t_{AOB}	ACK = 0 to OBF = 1(1)		350		350	ns
t_{SIB}	STB = 0 to IBF = 1(1)		300		300	ns
t_{RIB}	RD = 1 to IBF = 0(1)		300		300	ns
t_{RIT}	RD = 0 to INTR = 0(1)		400		400	ns
t_{SIT}	STB = 1 to INTR = 1(1)		300		300	ns
t_{AIT}	ACK = 1 to INTR = 1(1)		350		350	ns
t_{WIT}	WR = 0 to INTR = 0(1, 3)		450		450	ns

NOTES:
1. Test Conditions: C_L = 150 pF.
2. Period of Reset pulse must be at least 50 μs during or after power on. Subsequent Reset pulse can be 500 ns min.
3. INTR ↑ may occur as early as $\overline{WR}$ ↓.

*For Extended Temperature EXPRESS, use M8255A electrical parameters.

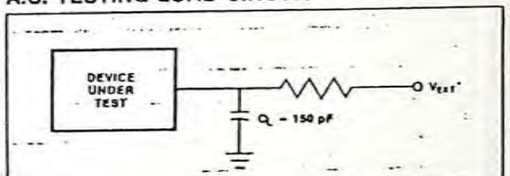
A.C. TESTING INPUT, OUTPUT WAVEFORM



231308-50

A.C. Testing: Inputs are driven at 2.4V for a Logic "1" and 0.45V for a Logic "0". Timing measurements are made at 2.0V for a Logic "1" and 0.8V for a Logic "0".

A.C. TESTING LOAD CIRCUIT

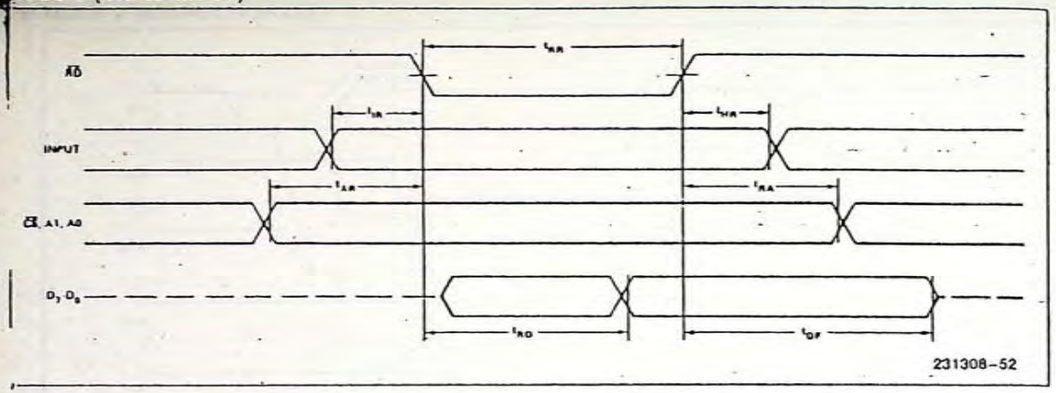


231308-51

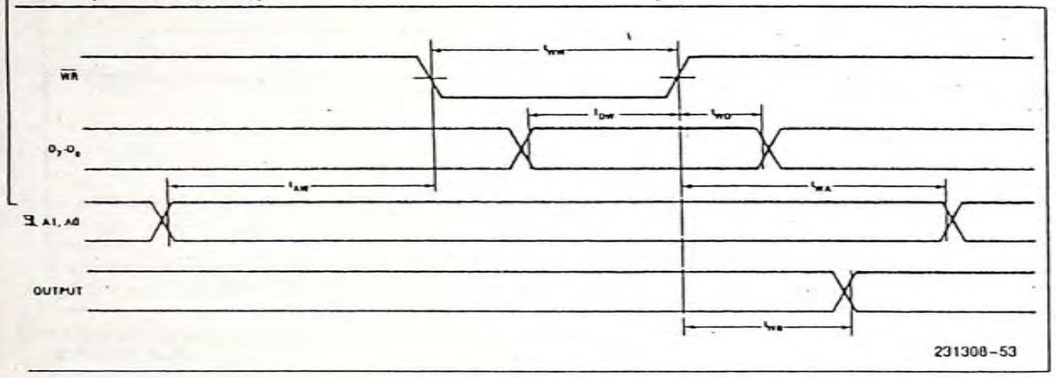
*V_{EXT} is set at various voltages during testing to guarantee the specification. C_L includes jig capacitance.

ORMS

ODE 0 (BASIC INPUT)

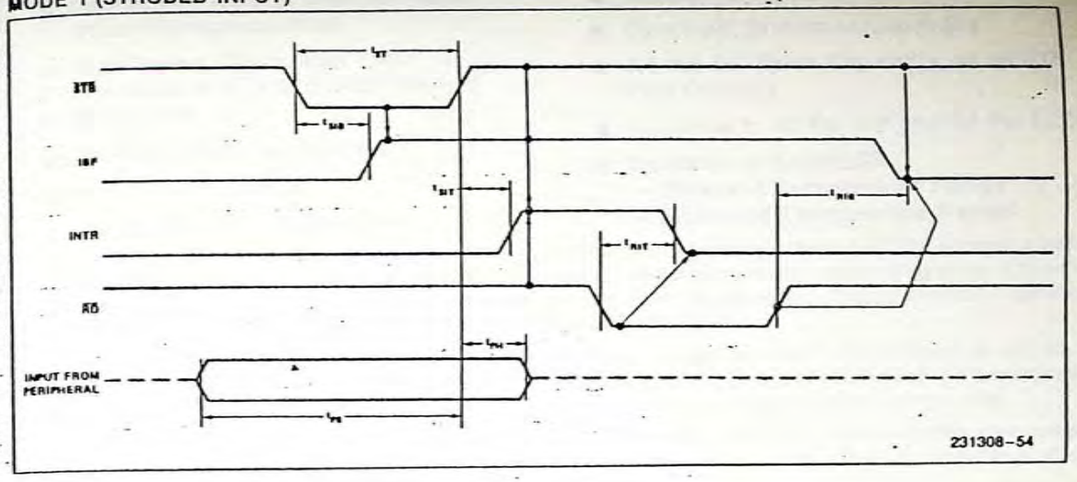


ODE 0 (BASIC OUTPUT)

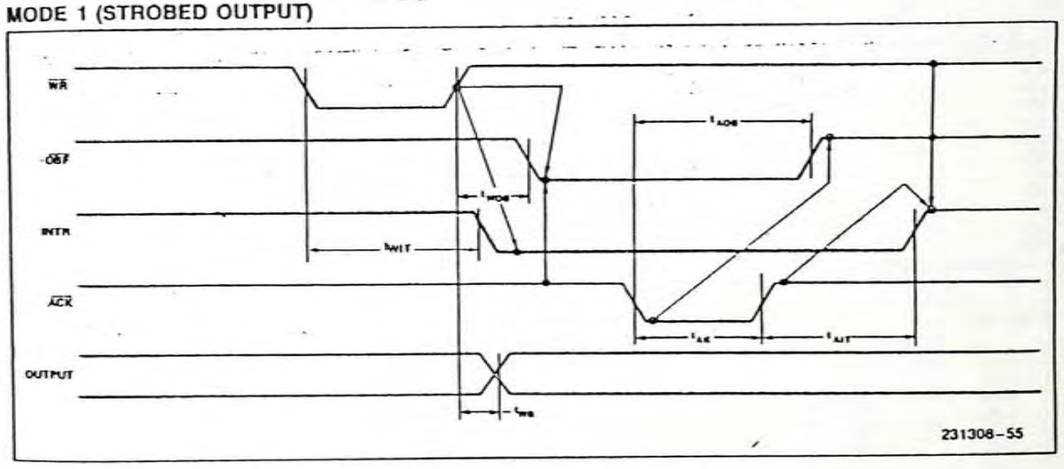


WAVEFORMS (Continued)

MODE 1 (STROBED INPUT)

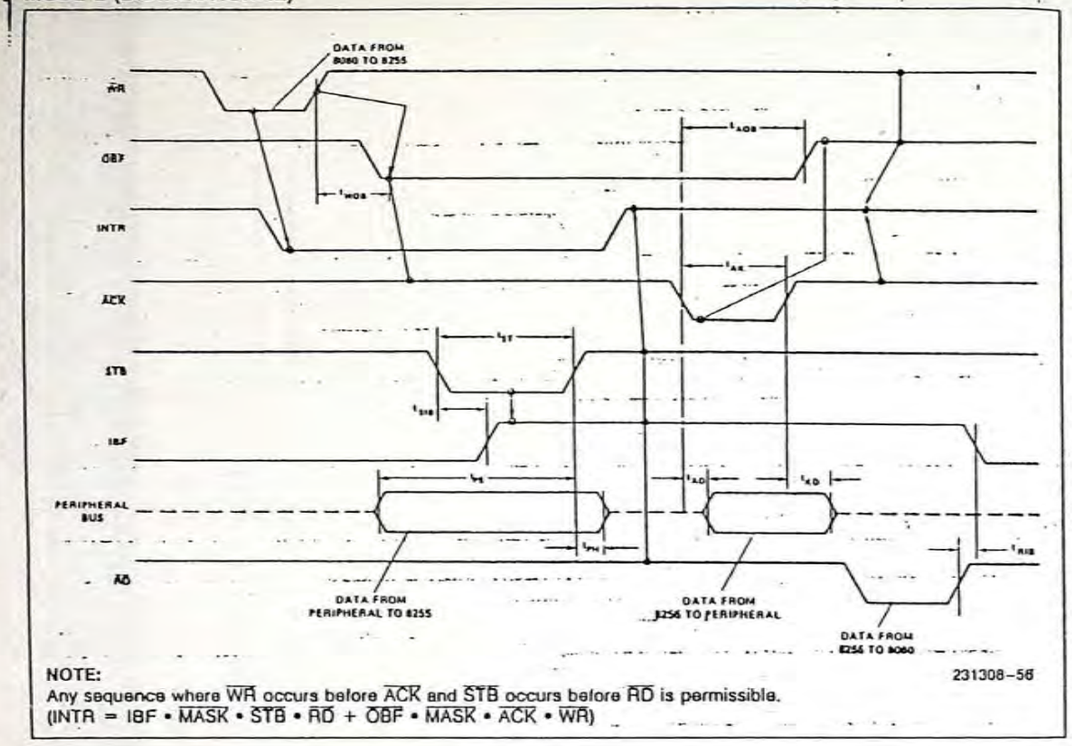


MODE 1 (STROBED OUTPUT)



...EFORMS (Continued)

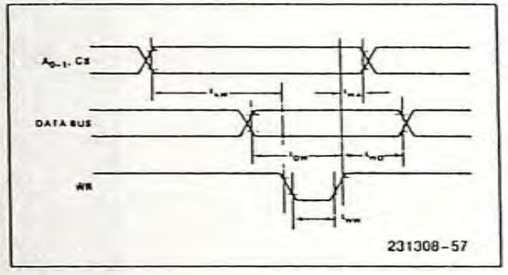
MODE 2 (BIDIRECTIONAL)



231308-56

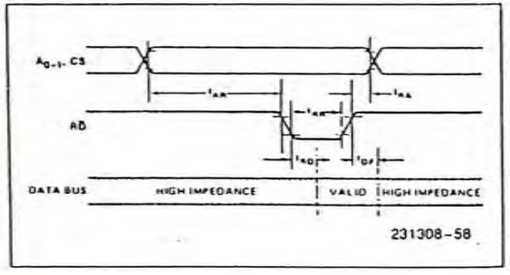
NOTE:
Any sequence where $\overline{WR}$ occurs before $\overline{ACK}$ and $\overline{STB}$ occurs before $\overline{RD}$ is permissible.
($INTR = IBF \cdot \overline{MASK} \cdot \overline{STB} \cdot \overline{RD} + \overline{OBF} \cdot \overline{MASK} \cdot \overline{ACK} \cdot \overline{WR}$)

WRITE TIMING



231308-57

READ TIMING



231308-58

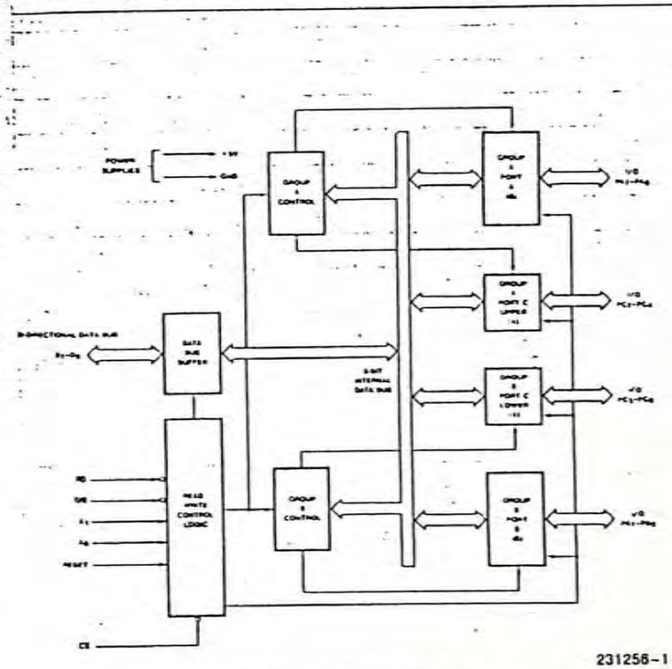
82C55A
CHMOS PROGRAMMABLE PERIPHERAL INTERFACE

- Compatible with all Intel and Most Other Microprocessors
- High Speed, "Zero Wait State" Operation with 8 MHz 8086/88 and 80186/188
- 24 Programmable I/O Pins
- Low Power CHMOS
- Completely TTL Compatible
- Control Word Read-Back Capability
- Direct Bit Set/Reset Capability
- 2.5 mA DC Drive Capability on all I/O Port Outputs
- Available in 40-Pin DIP and 44-Pin PLCC
- Available in EXPRESS
  - Standard Temperature Range
  - Extended Temperature Range

The Intel 82C55A is a high-performance, CHMOS version of the industry standard 8255A general purpose programmable I/O device which is designed for use with all Intel and most other microprocessors. It provides 24 I/O pins which may be individually programmed in 2 groups of 12 and used in 3 major modes of operation. The 82C55A is pin compatible with the NMOS 8255A and 8255A-5.

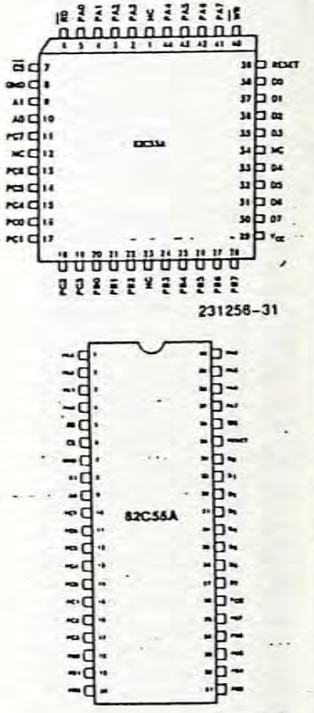
In MODE 0, each group of 12 I/O pins may be programmed in sets of 4 and 8 to be inputs or outputs. In MODE 1, each group may be programmed to have 8 lines of input or output. 3 of the remaining 4 pins are used for handshaking and interrupt control signals. MODE 2 is a strobed bi-directional bus configuration.

The 82C55A is fabricated on Intel's advanced CHMOS III technology which provides low power consumption with performance equal to or greater than the equivalent NMOS product. The 82C55A is available in 40-pin DIP and 44-pin plastic leaded chip carrier (PLCC) packages.



231256-1

Figure 1. 82C55A Block Diagram



231258-2

Figure 2. 82C55A Pinout
Diagrams are for pin reference only. Package sizes are not to scale.

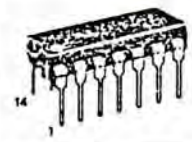
October 1988
Order Number: 231258-003

MOTOROLA
SEMICONDUCTOR
TECHNICAL DATA

MC145436

Advance Information

Dual Tone Multiple Frequency Receiver

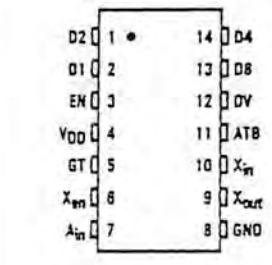


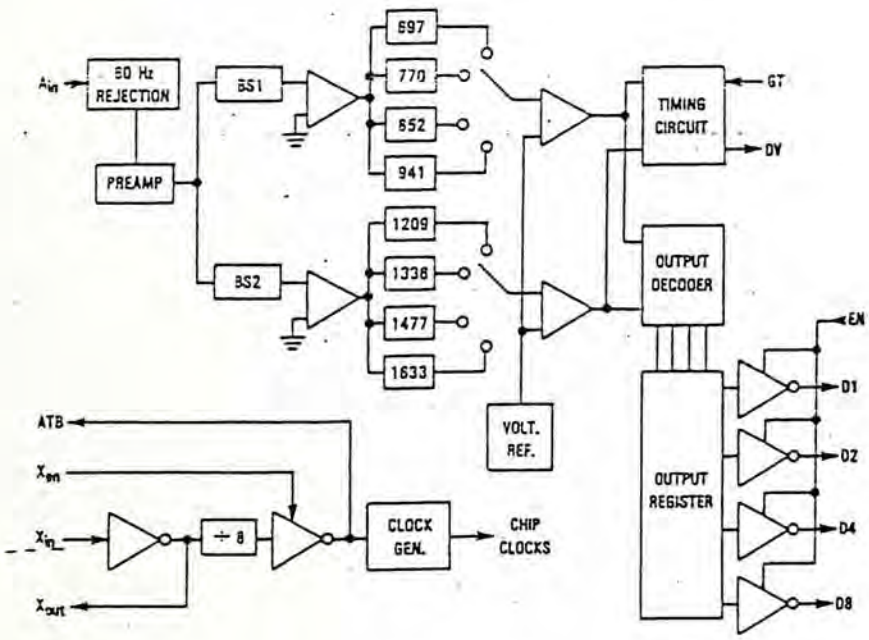
P SUFFIX
PLASTIC
CASE 646

The MC145436 is a silicon-gate CMOS LSI device containing the filter and decoder for detection of a pair of tones conforming to the DTMF standard with outputs in hexadecimal. Switched capacitor filter technology is used together with digital circuitry for the timing control and output circuits. The MC145436 provides excellent power-line noise and dial tone rejection, and is suitable for applications in central office equipment, PABX, key-phone systems, remote control equipment, and consumer telephony products.

- Single +5 V Power Supply
- Detects All 16 Standard Digits
- Uses Inexpensive 3.579545 MHz Colorburst Crystal
- Provides Guard Time Controls to Improve Speech Immunity
- Output in 4-Bit Hexadecimal Code
- Built-In 60 Hz and Dial Tone Rejection
- Pin Compatible with SSI-204

PIN ASSIGNMENT





This document contains information on a new product. Specifications and information herein are subject to change without notice.

ABSOLUTE MAXIMUM RATINGS

(Voltages Referenced to GND Unless Otherwise Noted)

Rating	Symbol	Value	Unit
DC Supply Voltage	V_{DD}	−0.5 to +6.0	V
Input Voltage, Any Pin Except A_{in}	V_{in}	−0.5 to V_{DD}+0.5	V
Input Voltage, A_{in}-	V_{in}	V_{DD}−10 to V_{DD}+0.5	V
DC Current Drain per Pin	I	±10	mA
Operating Temperature Range	T_A	−40 to +85	°C
Storage Temperature Range	T_{stg}	−65 to +150	°C

This device contains circuitry to protect the inputs against damage due to high static voltages or electric fields; however, it is advised that normal precautions be taken to avoid applications of any voltage higher than the maximum rated voltages to this high impedance circuit.

For proper operation it is recommended that V_{in} and V_{out} be constrained to the range $V_{SS} \le (V_{in}$ or $V_{out}) \le V_{DD}$. Reliability of operation is enhanced if unused inputs are tied to an appropriate logic voltage level (e.g., either V_{SS} or V_{DD}).

ELECTRICAL CHARACTERISTICS

(All Polarities Referenced to V_{DD}=5.0 V ±10%, T_A=−40 to +85°C Unless Otherwise Noted)

Parameter		Symbol	Min	Typ	Max	Unit
DC Supply Voltage		V_{DD}	4.5	5	5.5	V
Supply Current (f_{CLK}=3.58 MHz)		I_{DD}	—	7	15	mA
Input Current	GT	I_{in}	—	—	200	μA
	EN, X_{in}, $\overline{X_{en}}$		—	—	±1	
Input Voltage Low	EN, GT, X_{en}	V_{IL}	—	—	1.5	V
Input Voltage High	EN, GT, X_{en}	V_{IH}	3.5	—	—	V
High Level Output Current (V_{OH}=V_{DD}−0.5 V; Source)	Data, DV	I_{OH}	800	—	—	μA
Low Level Output Current (V_{OL}=0.4 V; Sink)	Data, DV	I_{OL}	1.0	—	—	mA
Input Impedance	A_{in}	R_{in}	90	100	—	kΩ
Fanout	ATB	FO	—	—	10	
Input Capacitance	X_{en}, EN	C_{in}	—	6	—	pF

ANALOG CHARACTERISTICS (V_{DD}=5.0 V ±10%, T_A=−40 to +85°C)

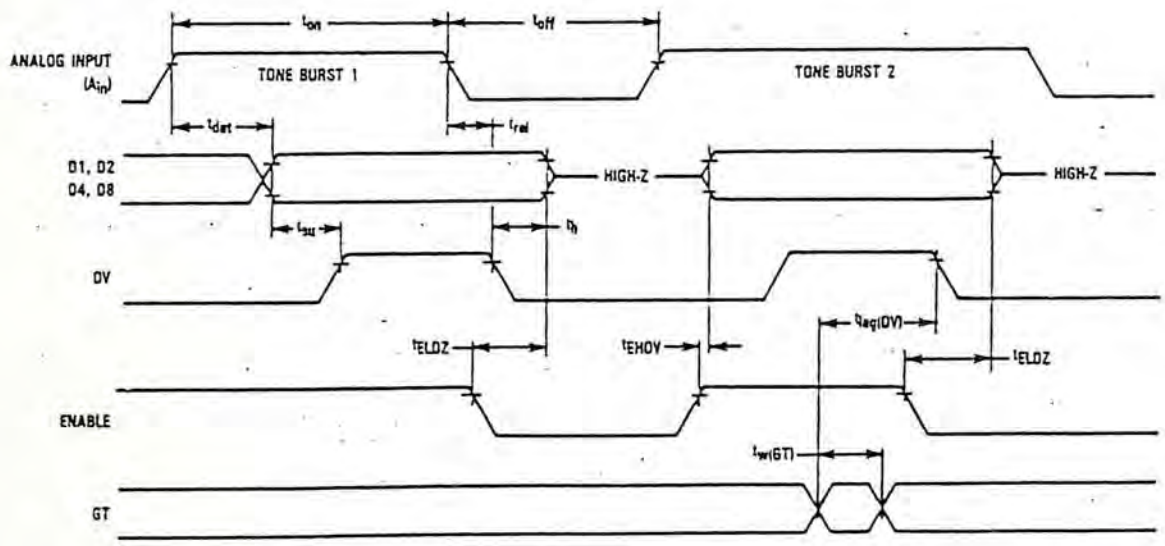
Parameter	Min	Typ	Max	Unit
Signal Level for Detection (A_{in})	−32	—	−2	dBm
Twist = High Tone/Low Tone	−10	—	10	dB
Frequency Detect Bandwidth (Notes 1 and 2)	±(1.5+2 Hz)	±2.5	±3.5	% f_C
60 Hz Tolerance	—	—	0.8	Vrms
Dial Tone Tolerance (Note 3) (Dial Tone 330+440)	—	—	0	dB
Noise Tolerance (Notes 3 and 4)	—	—	−12	dB
Power Supply Noise (Wide Band)	—	—	10	mV p-p
Talk Off (Mitel Tape #CM7290)	—	2	—	Hits

NOTES:
1. f_C is center frequency of bandpass filters.
2. Maximum frequency detect bandwidth of the 1477 Hz filter is +3.5% to −4%.
3. Referenced to lower amplitude tone.
4. Bandwidth limited (0 to 3.4 kHz) Gaussian noise.

AC CHARACTERISTICS (V_{DD} = 5.0 V ± 10%, T_A = −40 to +85°C)

Characteristic		Symbol	Min	Typ	Max	Unit
Tone On Time	For Detection	$Tone_{on}$	40	—	—	ms
	For Rejection		—	—	20	
Pause Time	For Detection	$Tone_{off}$	40	—	—	ms
	For Rejection		—	—	20	
Detect Time	GT = 0	t_{det}	22	—	40	ms
	GT = 1		32	—	50	
Release Time	GT = 0	t_{rel}	28	—	40	ms
	GT = 1		18	—	30	
Data Setup Time		t_{su}	7	—	—	µs
Data Hold Time		t_h	4.2	4.6	5	ms
Pulse Width	GT	$t_{w(GT)}$	18	—	—	µs
DV Reset Lag Time		$t_{lag(DV)}$	—	—	5	ms
Enable High to Output Data Valid		t_{EHDV}	—	200	—	ns
Enable Low to Output High-Z		t_{ELOZ}	—	150	—	ns

TIMING



D1, D2, D4, D8—

These digital ou
responding to the
outputs become va
are cleared when
are high impedanc

EN—ENABLE

Outputs D1, D2
1, and high impec

GT—GUARD T

The Guard Tin
time and release
on and tone off.
short to be consi
is improved, sin
ulated by speec
to be accepted.
probability that
will be detected
tone detect time
for an extremely
and immunity t
tone signal time
followed by a p
to-high transiti
the MC145436

X_{en}—OSCILL

A logic 1 on
using alternat
tied to GND.

Tab

Digit
1
2
3
4
5
6
7
8
9
0
*
#
A
B
C
D

PIN DESCRIPTION

D1, D2, D4, D8—DATA OUTPUT

These digital outputs provide the hexadecimal codes corresponding to the detected digit (see Table 1). The digital outputs become valid after a tone pair has been detected, and are cleared when a valid pause is timed. These output pins are high impedance when Enable is at a logic 0.

EN—ENABLE

Outputs D1, D2, D4, D8 are enabled when EN is at a logic 1, and hign impedance (disabled) when EN is at a logic 0.

GT—GUARD TIME

The Guard Time control input provides two sets of detected time and release time, both within the allowed ranges of tone on and tone off. A longer tone detect time rejects signals too short to be considered valid. With GT = 1, talk off performance is improved, since it reduces the probability that tones simulated by speech will maintain signal conditions long enough to be accepted. In addition, a shorter release time reduces the probability that a pause simulated by an interruption in speech will be detected as a valid pause. On the other hand, a shorter tone detect time with a long release time would be appropriate for an extremely noisy environment where fast acquisition time and immunity to drop-outs would be required. In general, the tone signal time generated by a telephone is 100 ms, nominal, followed by a pause of about 100 ms. A high-to-low, or low-to-high transition on the GT pin resets the internal logic, and the MC145436 is immediately ready to accept a new tone input.

X_{en}—OSCILLATOR ENABLE

A logic 1 on X_{en} enables the on-chip crystal oscillator. When using alternate time base from the ATB pin, X_{en} should be tied to GND.

A_{in}—ANALOG INPUT

This pin accepts the analog input, and is internally biased so that the input signal may be ac coupled. The input may be dc coupled so long as it does not exceed the positive supply. (See Figure 1.)

X_{in}/X_{out}—OSCILLATOR IN AND OSCILLATOR OUT

These pins connect to an internal crystal oscillator. In operation, a parallel resonant crystal is connected from X_{in} to X_{out}, as well as a 1 MΩ resistor in parallel with the crystal. When using the alternate clock source from ATB, X_{in} should be tied to V_{DD}.

ATB—ALTERNATE TIME BASE

This pin serves as a frequency reference when more than one MC145436 is used, so that only one crystal is required for multiple MC145436s. In this case, all ATB pins should be tied together as shown in Figure 2. When only one MC145436 is used, this pin should be left unconnected. The output frequency of ATB is 447.4 kHz.

DV—DATA VALID

DV signals a detection by going high after a valid tone pair is sensed and decoded at output pins D1, D2, D4, D8. DV remains high until a loss of the current DTMF signal occurs, or until a transition in GT occurs.

V_{DD}—POSITIVE POWER SUPPLY

The digital supply pin, which is connected to the positive side of the power supply.

GND—GROUND

Ground return pin is typically connected to the system ground.

Table 1. Hexadecimal Codes

Digit	Output Code			
	D8	D4	D2	D1
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
9	1	0	0	1
0	1	0	1	0
*	1	0	1	1
#	1	1	0	0
A	1	1	0	1
B	1	1	1	0
C	1	1	1	1
D	0	0	0	0

OPERATIONAL INFORMATION

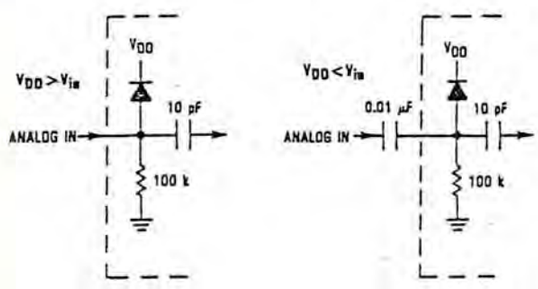


Figure 1. Analog Input

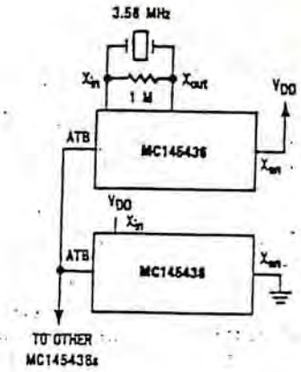


Figure 2. Multiple MC145436s

		COL 1	COL 2	COL 3	COL 4	
	697	1	2	3	A	ROW 1
STD	770	4	5	6	B	ROW 2
DTMF	852	7	8	9	C	ROW 3
(Hz)	941	*	0	#	D	ROW 4
		1209	1336	1477	1633	

STD DTMF (Hz)

Figure 3. 4×4 Keyboard Matrix

Appendix 2

User Guide of the Application Generator Program (AppGen.EXE)

AppGen is a menu driven program which serves to help users to tailor their application. After typing 'AppGen' and then followed by an <Enter> at the DOS prompt, a horizontal bar menu as shown in Fig. A2.1 appears on the screen. One can select what he intend to perform by using the arrow keys.

When the FileRetrieve option is chosen, a submenu as depicted in Fig. A2.2 will pop up. If sub-item New Application is selected, an Application Generator screen will then show on video display unit (Fig. A2.3). This page allows users to edit all the voice messages and events required in a specific Event Node. Users can even traverse among Event Nodes by placing a small cursor beside the digit column and then press <Enter> to decide whether to go forward or backward. Another convenient facility provided here is that users may press Alt-R, Alt-P or Alt-S to record, play or save voice message respectively.

Choosing the Get Application File item functions more or less the same as the previous one except it requires users to select an application file from a pop up window as shown in Fig. A2.4.

Fig. A2.5 and Fig. A2.6 illustrate the voice record function of AppGen. Fig. A2.7 shows the scenario when the PlayBack option is selected. The program will prompt users to choose whether to play voice in memory or in disk. Indeed, Voice in memory means

voice message that is just recorded or loaded from disk while voice in disk refers to messages that are stored in hardisk or diskette.

The Browse option has not yet been implemented. It is primarily intended to provide users with a means to test Application stored using keyboard to emulate touch tone telephone keypad.

The Environment option allows users to configure all the necessary items required for running an Application. Meanwhile, its also contains an sub-option called Display Configuration Info which presents the current settings on screen (Please refer to Fig. A2.8 and A2.9).

Fig. A2.10 and Fig. A2.11 depict the functions of Save and Quit options and are self explanatory.

Retrieve Application File

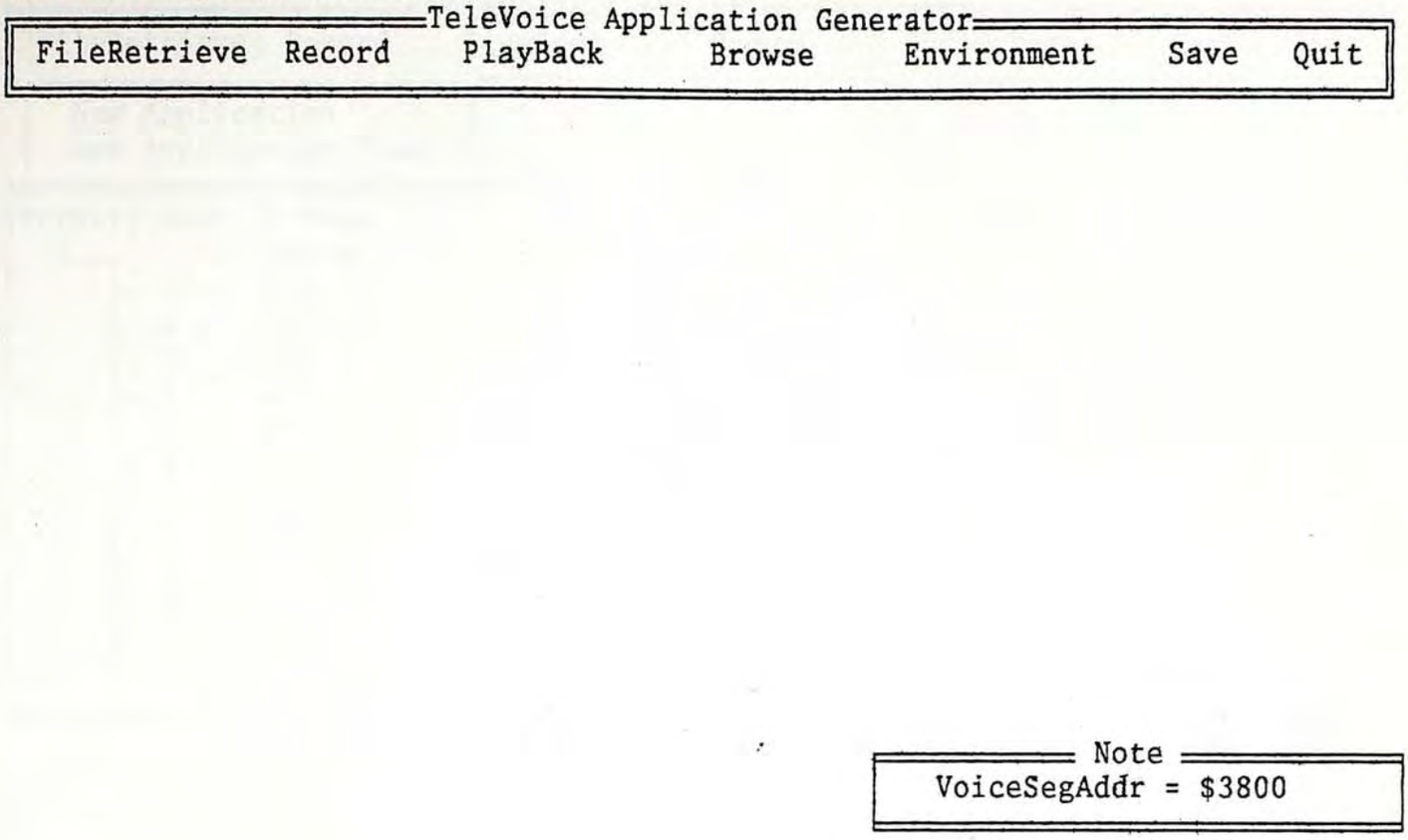


Fig. A2.1

Create A New Application

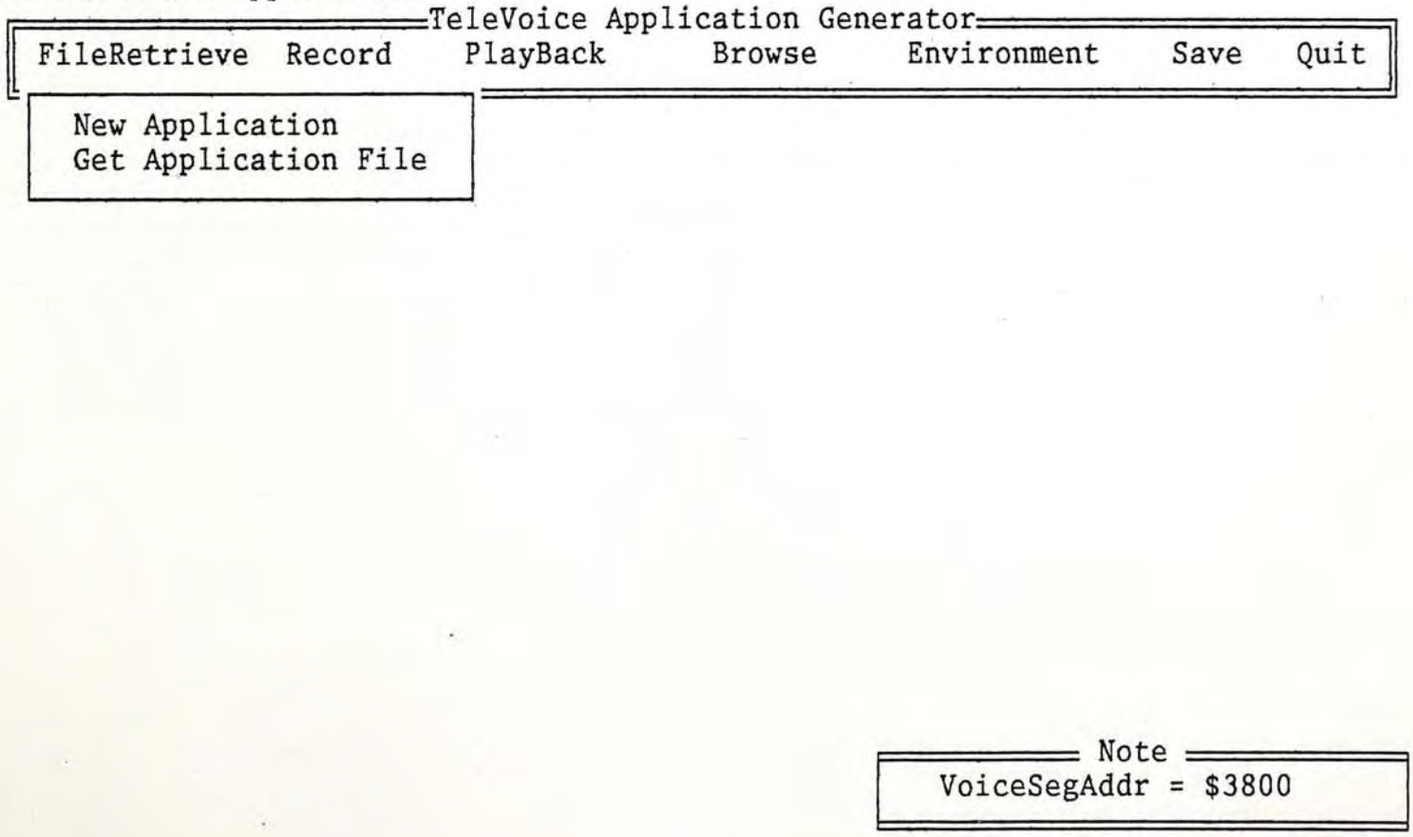


Fig. A2.2

Create A New Application

```
TeleVoice Application Generator
FileRetrieve  Record  PlayBack  Browse  Environment  Save  Quit

  New Application
  Get Application File

Application Generator
Primary Node :P-Node          Action : Welcome      Voice : Greeting.msg
  └─┐          Voice              Action
    ├─ 1 -     Nil                Nil
    ├─ 2 -     Nil                Nil
    ├─ 3 -     Nil                Nil
    ├─ 4 -     Nil                Nil
    ├─ 5 -     Nil                Nil
    ├─ 6 -     Nil                Nil
    ├─ 7 -     Nil                Nil
    ├─ 8 -     Nil                Nil
    ├─ 9 -     Nil                Nil
    ├─ 0 -     Nil                Nil
    ├─ * -     Nil                Nil
    └─ # -     Nil                Nil                    Level :: 0

        Note
  Use Arrow Keys To Select
```

Fig. A2.3

Retrieve Stored Application

```
TeleVoice Application Generator
FileRetrieve  Record  PlayBack  Browse  Environment  Save  Quit

  New Application
  Get Application File

        A:\*.*
MAILDATA        <dir>   4/23/92   6:27p
_DEMO1          <dir>   4/23/92   6:09p
_DEMO2          <dir>   4/23/92   6:10p
sr9.com          1112   3/20/92   3:49p

        Note
   VoiceSeg = $3800
```

Fig. A2.4

Voice Recording

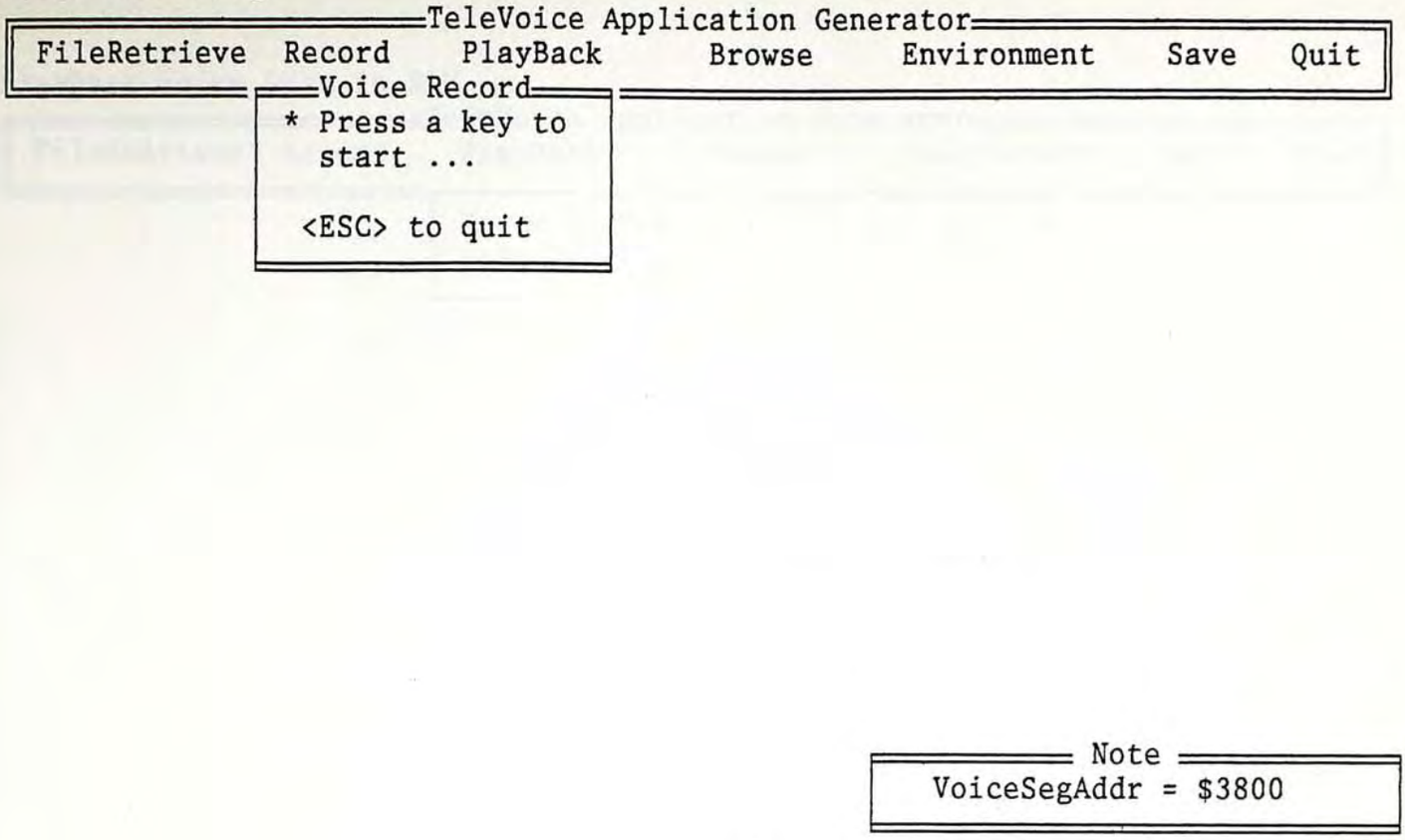


Fig. A2.5

Voice Recording

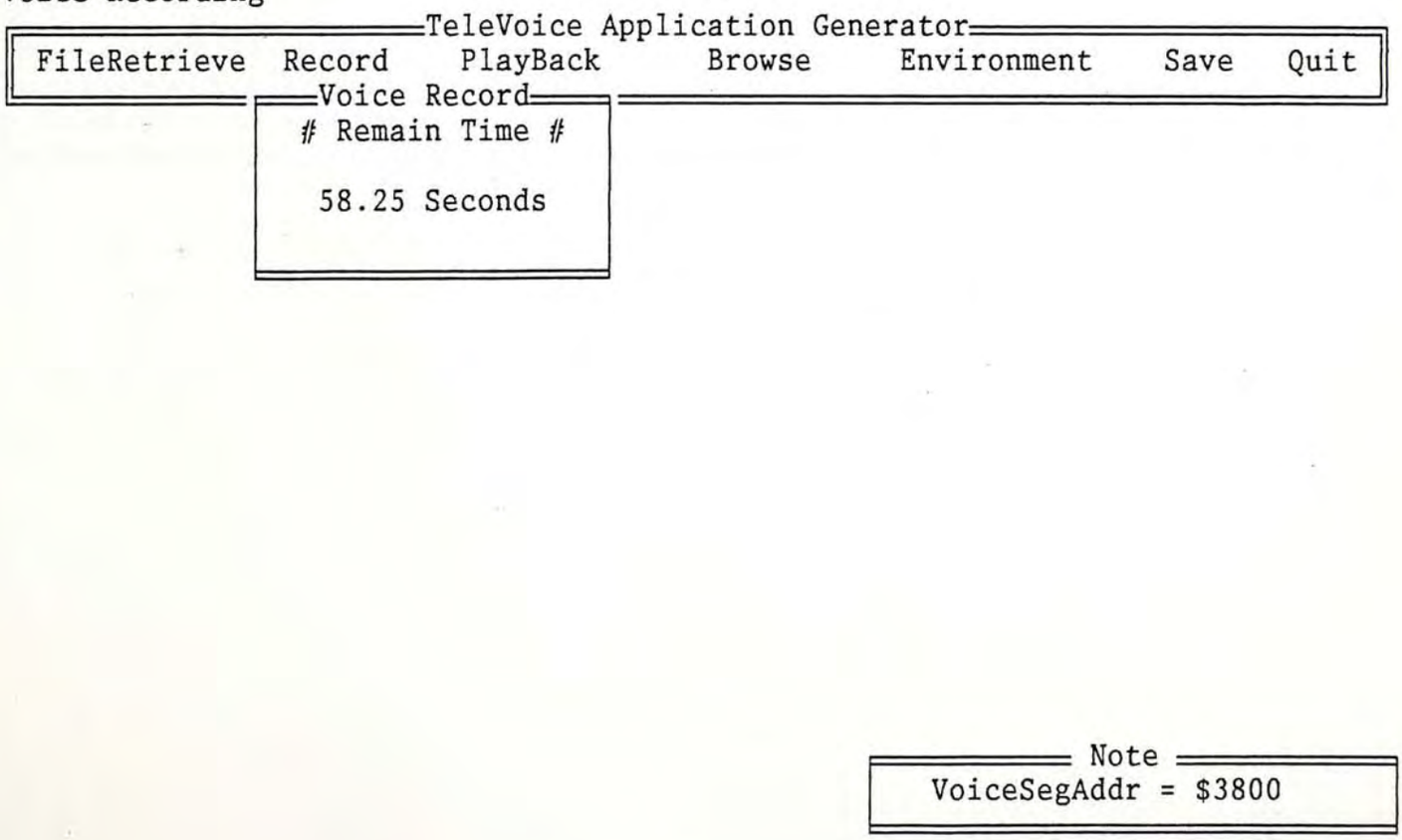


Fig. A2.6

PlayBack Voice Data in RAM

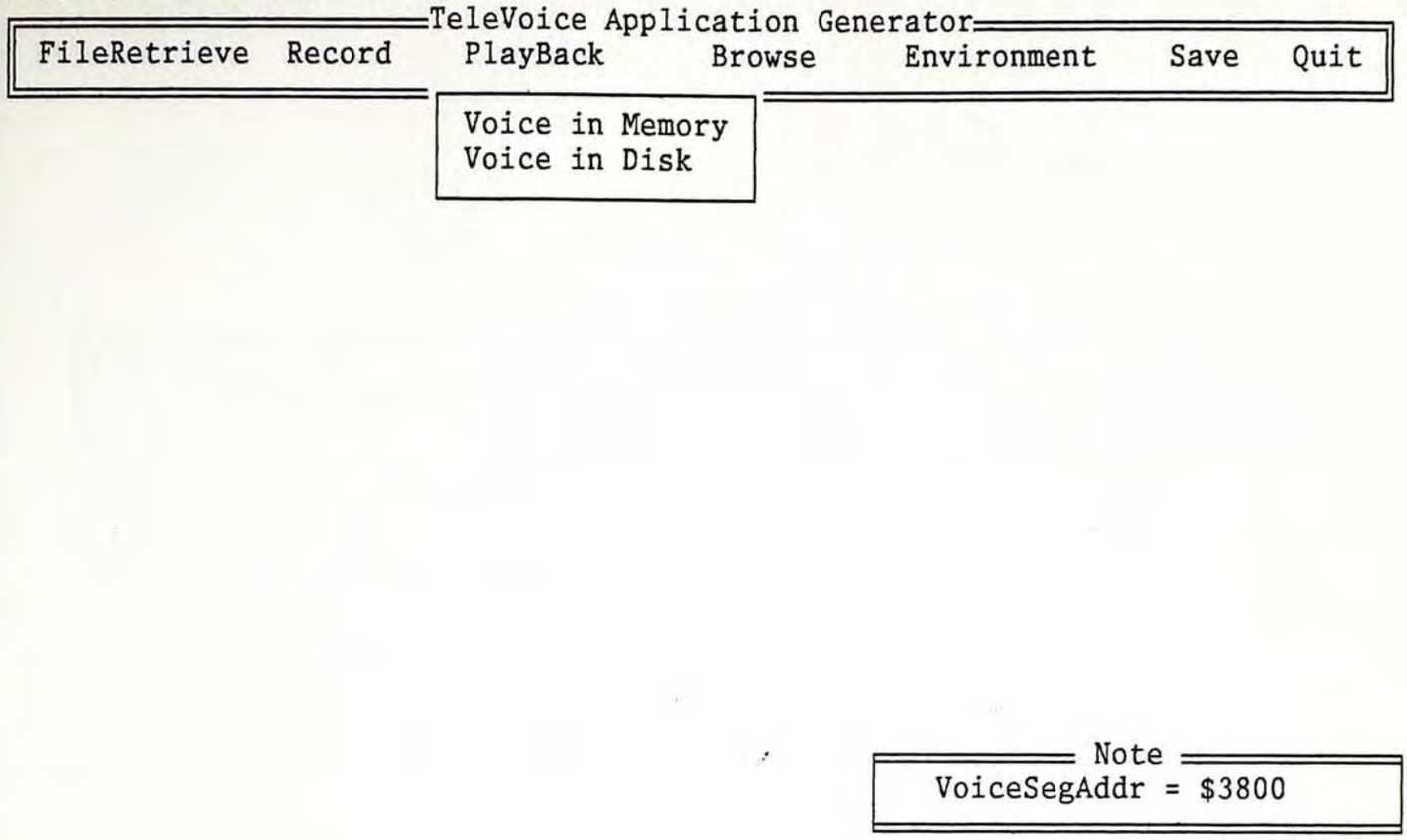


Fig. A2.7

Show Configuration Information

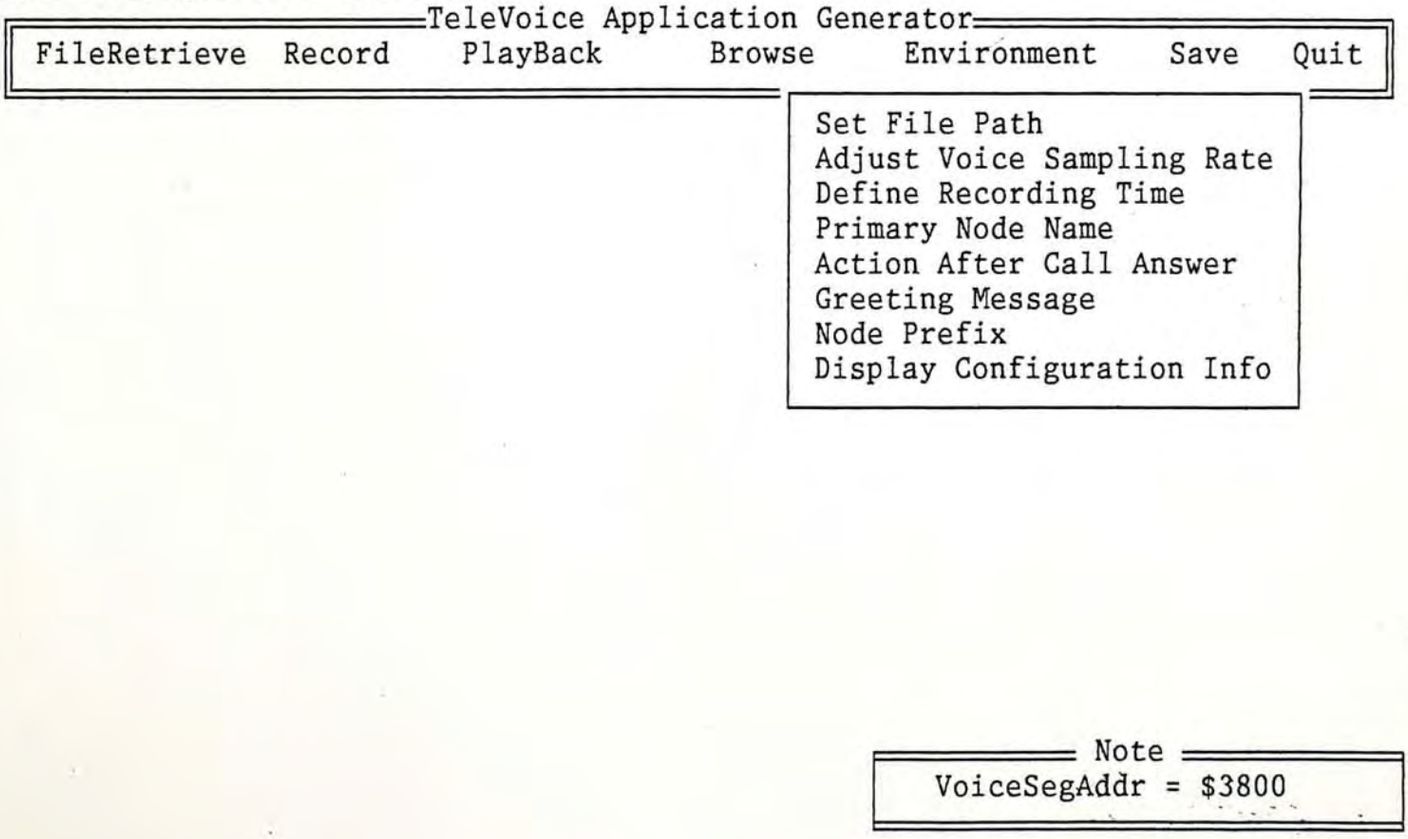


Fig. A2.8

Show Configuration Information

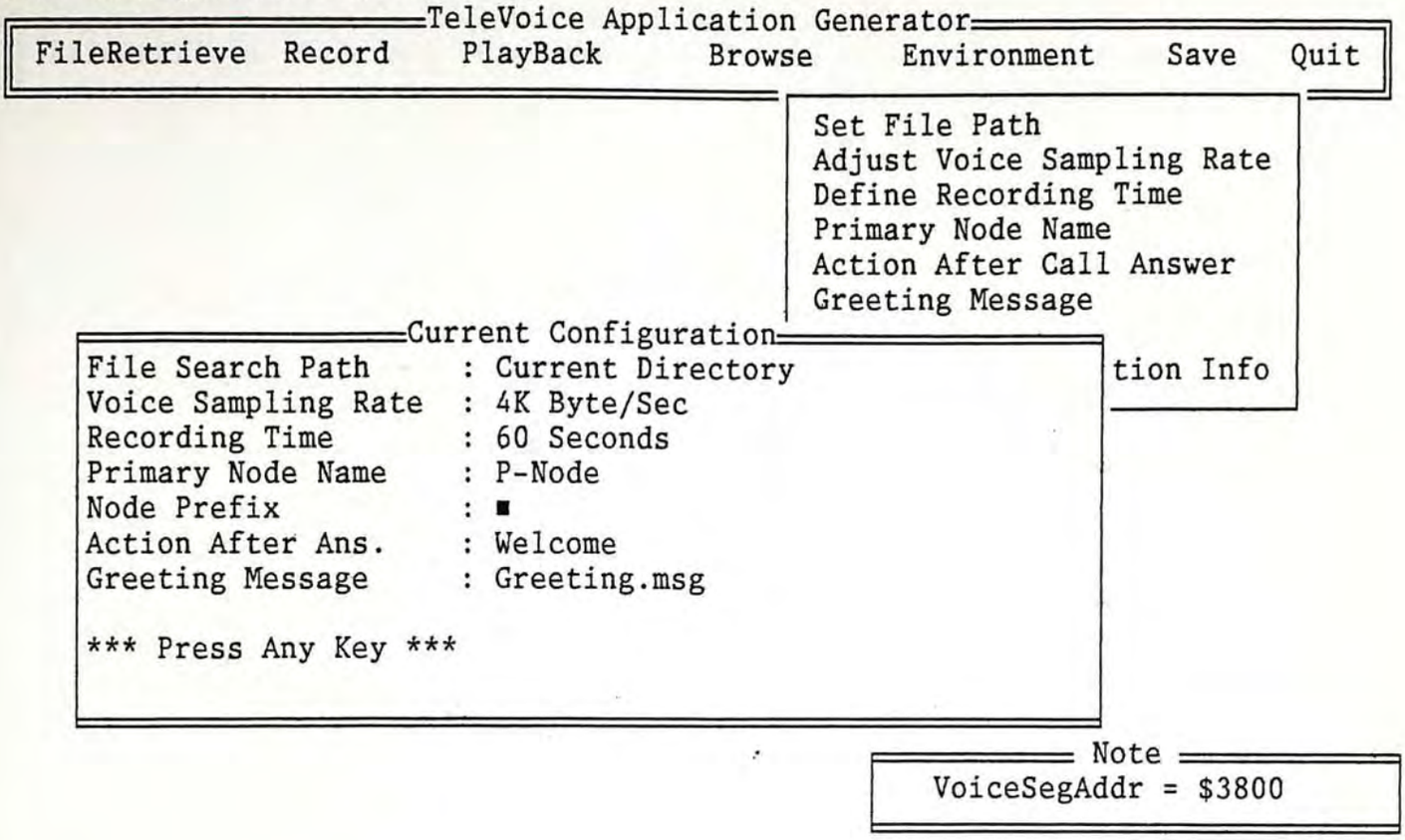


Fig. A2.9

Store Current Voice Data in RAM to Disk

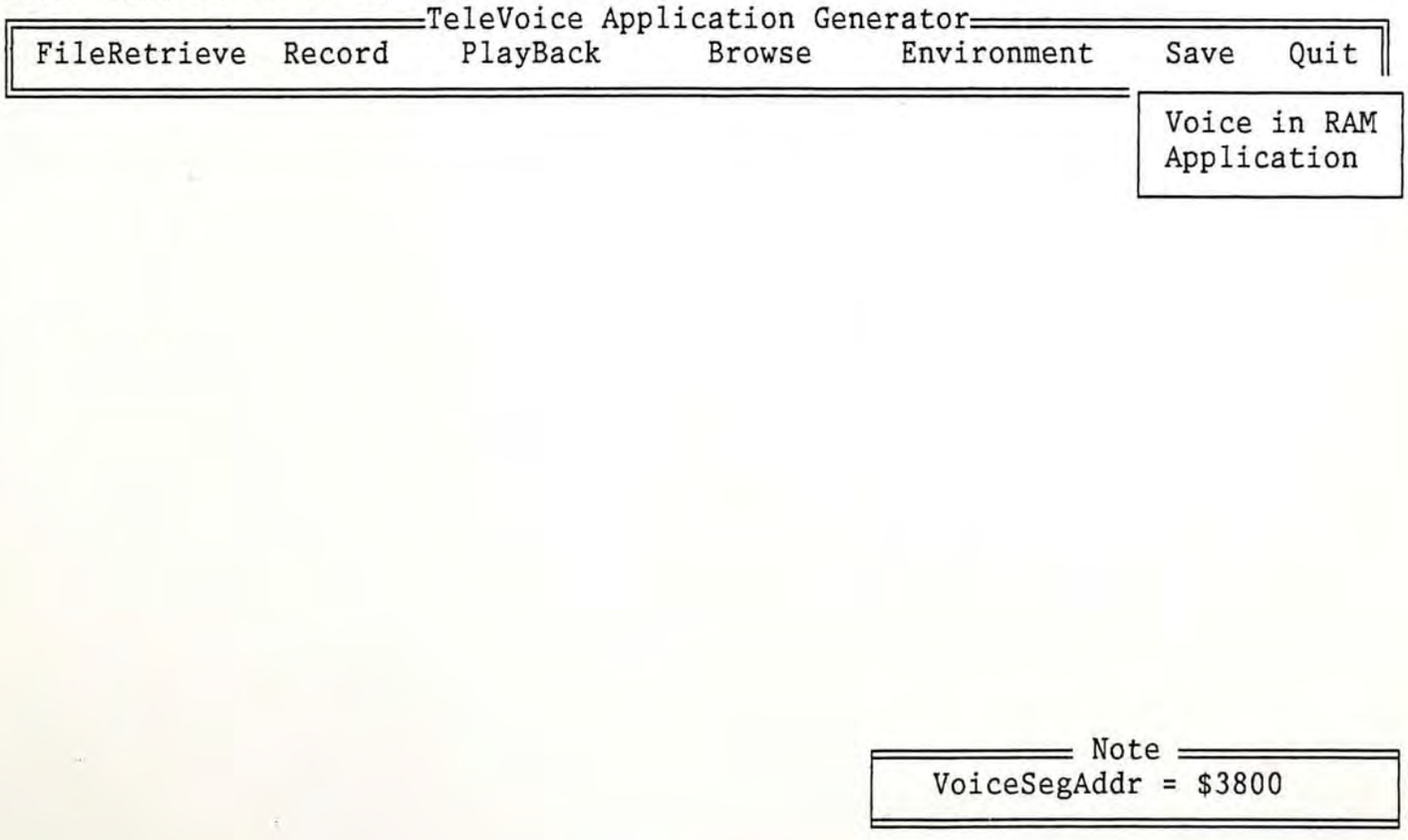
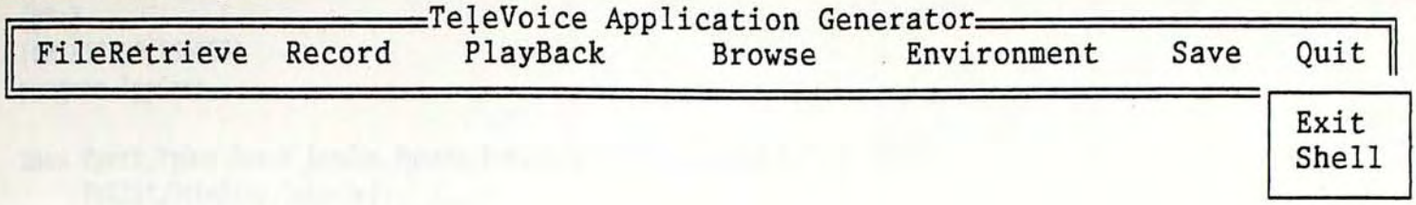


Fig. A2.10

Exit The Program



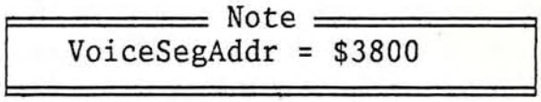


Fig. A2.11

```
{$F+}
{$M 8192,0,150000}
program AppGen;

uses Tpcrt,TpDos,Dos,M_AppGen,Tpmenu,TpWindow,TpString,Tpcmd,TpPick,TpDir,
     TpEdit,DataStru,ColorDef;

type
  anystring = string[80];


const
  _32K      = 32*1024;
  ESC       = #27;
  CR        = #13;
  EOF       = #26;                (* $1A *)

  MaxNode   = 100000;

  Signature : string = 'WCSUM';

  (* Default Config Values *)
  FilePath       : String    = ''           ;          { current directory }
  SamplingRate   : Char      = 'D'          ;          { 4K bytes per sec }
  RecTime        : string    = '60'         ;          { 60 Seconds }
  PrimaryNodeName : string   = 'P-Node'     ;
  NodePrefix     : string    = '■'          ;
  ActionAfterAns : string    = 'Welcome'    ;
  GreetMessage   : string    = 'Greeting.msg';

  VoiceSeg       : string[4] = '****';
  VoiceOfs       : string[4] = '****';
  HexDigit       : string[4] = '****';

  playflag       : Boolean = False;
  PlayInterrupted : Boolean = False;

  ANewApplication : Boolean = True;

  SystemNodeNumber : longint = 0;

  NothingToSave : Boolean = True;

var
   segment             : word;
   ptr1,ptr2,ptr3      : pointer;
   DecMemSeg,DecMemOfs : word;
   choice              : MenuKey;
   ch                  : char;              { Used by Main Menu }
   Mono                : Boolean;           { Video Mode }
   SaveVoiceTime       : Real;
   VoiceLoadedFromDisk : Boolean;
```

```
   LoadedVoiceFile      : string;

   NoteBoard                    : WindowPtr;
   RecWindow                    : WindowPtr;
   PlayWindow                   : WindowPtr;
   DummyWindow                  : WindowPtr;
   SetFilePathWin               : WindowPtr;
   DefineRecTimeWindow          : WindowPtr;
   DispConfigInfoWindow         : WindowPtr;
   SaveVoiceMessageWindow       : WindowPtr;
   ApplicationWindow            : WindowPtr;
   EditWindow                   : WindowPtr;
   SaveApplicationWindow        : WindowPtr;
   GetApplicationWindow         : WindowPtr;
   DefinePrimaryNodeNameWindow  : WindowPtr;
   DefineNodePrefixWindow       : WindowPtr;
   DefineActionAfterAnsWindow   : WindowPtr;
   DefineGreetMessageWindow     : WindowPtr;

   ScreenBufPtr         : pointer;

   WindowAttr           : Byte;
   FrameAttr            : Byte;
   HeaderAttr           : Byte;
   NWA,NFA,NHA          : Byte;              { Corresponding Attr for NoteBoard}

   node             : NodeStructure;
   PresentNodeName  : string;
   PresentLevel     : integer;
   IsPrimaryNode    : Boolean;
   SaveOkay         : Boolean;

{forward declaration}
procedure DispNote(message : string);    forward;
procedure WriteNote(message : string);   forward;

(*************************************************************************)
(*                     Supporting Routines                               *)
(*************************************************************************)

procedure DisableInterrupts;
begin
  inline($FA);
end;

procedure EnableInterrupts;
begin
  inline($FB);
end;

function power(value,index : integer) : word;  { index should +ve }
var
```

```
  i      : integer;
  result : word;
begin
  result := 1;
  for i := 1 to index do
    begin
      result := result*value;
    end;
  power := result;
end;

procedure DecToHex(decimal : word);
var
  addr    : string[4];
  rem     : word;
  i,buff  : integer;
begin
  rem := decimal;
  addr := '****';
  for i := 1 to 4 do
    begin
      buff := trunc(rem/power(16,4-i));
      if (buff > 9) then addr[i] := char(integer('A')+buff-10)
      else addr[i] := char(integer('0')+buff);
      rem := rem - buff*power(16,4-i);
    end;
  HexDigit := copy(addr,1,length(addr));
end;
(* Call to DecToHex causes result to be stored in hexdigit
   which is of type string[4] *)

function HexToDec(hex : string) : word;
var
  result     : word;
  i,buff     : integer;
begin
  result := 0;
  for i := 1 to 4 do
    begin
      if hex[i] in ['A'..'F'] then
        buff := integer(hex[i]) - integer('A') + 10
      else
        buff := integer(hex[i]) - integer('0');

      result := result + buff*power(16,4-i);
    end;
  HexToDec := result;
end;

procedure waitkey;
var
  ch    : char;
```

```
begin
  repeat until keypressed;
  while keypressed do ch := readkey;
end;


function StopWatch(hr,min,sec,sec100 : word) : real;
var
  hour,minute  : word;
  second       : word;
  second100    : word;
  t1,t2        : real;
begin
  GetTime(hour,minute,second,second100);
  if (hour = 0) and (hr <> 0) then hour := 24;
  t1 := hour*3600 + minute*60 + second + second100*0.01;
  t2 := hr*3600 + min*60 + sec + sec100*0.01;
  StopWatch := t1 - t2;
end;

procedure PseudoMusic;
var
  freq       : integer;
  i          : integer;
begin
  for i := 1 to 6 do begin
    randomize;
    freq := random(1000);
    sound(freq);
    delay(50);
    nosound
  end;
end;


procedure VideoDetect;
var
  regs         : registers;
  disp_mode    : byte;
begin
  { Detect Display Mode }
  regs.ah := 15;
  intr($10,regs);
  disp_mode := regs.al;

  { Select Display Mode }
  if (disp_mode = 2) or (disp_mode = 3) then
    begin
      Mono := False;
      TextMode(C80);
    end
  else if (disp_mode = 7) then Mono := True;
```

```
end;

procedure Shell;
var
  ErrorCode    : string[3];
begin
  if not SaveWindow(1,1,80,25,True,ScreenBufPtr) then
    begin
      WriteNote('Insufficient memory');
      sound(3000); delay(500); nosound;
      exit;
    end;

  SwapVectors;
  Exec(GetEnv('COMSPEC'),'');
  SwapVectors;

  if DosError <> 0 then begin
    str(DosError,ErrorCode);
    WriteNote('Cannot Shell <' + ErrorCode + '>');
    waitkey;
  end;

  RestoreWindow(1,1,80,25,True,ScreenBufPtr);
end;

function AppendPath(path : string) : string;
begin
  if path = '' then
    begin
      AppendPath := '';
      exit;
    end;
  if (path[length(path)] <> '\') then
    AppendPath := path + '\'
  else
    AppendPath := path;
end;

procedure SelectFile(var Mask,fname : string);
const
  DirColorC : PickColorArray = (BlueOnLtGray,BlackOnLtGray,RedOnLtGray,
                                BlackOnGreen,YellowOnCyan,BlackOnGreen);

  DirColorM : PickColorArray = (BlueOnLtGray,BlackOnLtGray,RedOnLtGray,
                                BlackOnLtGray,YellowOnCyan,BlackOnLtGray);
var
  ResultCode  : word;
  DirColor    : PickColorArray;

begin
  if Mono then
```

```
    DirColor := DirColorM
  else
    DirColor := DirColorC;

  TpPick.PickSrch := StringPickSrch;
  TpDir.ShowSizeDateTime := True;

  ResultCode := GetFileName(Mask,
                            AnyFile,
                            3,8,              { TopLeft corner of Dir Window }
                            24,3,             { One Column, ending at row 24 }
                            DirColor,
                            fname);

  if (fname[length(fname)] = '.') then
    fname := copy(fname,1,length(fname)-1);

  if not (ResultCode in [0..4]) then
    begin
      WriteNote('Critical Error Encounted');
      sound(3000); delay(1000); nosound;
      waitkey;
      fname := '';
    end
  else
    if ResultCode = 4 then
      begin
        WriteNote('Memory is deficient');
        sound(2000); delay(1000); nosound;
        waitkey;
        fname := '';
      end
  else
    if ResultCode = 2 then
      begin
        WriteNote('wrong path/no match file');
        sound(1000); delay(1000); nosound;
        waitkey;
        fname := ''
      end;
  WriteNote('VoiceSegAddr = $'+VoiceSeg);
end;

procedure MoveWindow(XDelta,YDelta,trial : integer);
var
  dummy   : Boolean;
begin
  if (trial < 1) then exit;
  repeat
    dummy := TpWindow.MoveWindow(XDelta,YDelta);
    Dec(trial);
    delay(50);
```

```
  until (trial = 0);
end;

function ValidNumberString(InputString : string) : Boolean;
var
  indx   : integer;
begin
  for indx := 1 to length(InputString) do
     if not (InputString[indx] in ['0'..'9']) then
       begin
         ValidNumberString := False;
         exit;
       end;

  if InputString[1] = '0' then
    ValidNumberString := False
  else
    ValidNumberString := True;
end;


procedure WriteWithChangeTextAttr(WriteAttr : byte ; WriteStr : string);
var
  SaveAttr  : byte;
begin
  SaveAttr := TextAttr;
  TextAttr := WriteAttr;
  write(WriteStr);
  TextAttr := SaveAttr;
end;

function IntToStr(i: Longint): string;
{ Convert any Integer type to a string }
var
  s: string[11];
begin
  Str(i, s);
  IntToStr := s;
end;

Procedure WriteSelectPointer(var SaveChar : char;
                             var SaveAttr : byte; I,J : integer);
begin
  gotoxy(I,J);
  SaveChar := ReadCharAtCursor;
  SaveAttr := ReadAttrAtCursor;
  FastWrite(#16,J,I,WhiteOnRed);
end;

procedure DeleteFile (fname : string);
var
  success  : integer;
```

```
begin
  success := ExecDos('Del ' + fname + ' > NUL',True,NIL);
end;


{-----------------------End of Supporting Routines----------------------}


(*****************************************************************************)
(*                      VRP-70 Voice Card Routines                           *)
(*****************************************************************************)

procedure CMDEXEC(s : anystring);
var
  ch            : byte;
  addr,i        : integer;
begin
  addr:=memw[0000:$182];       { get int$60 segment            }
  for i:=1 to ord(s[0]) do     { s[0] is the length of string s }
    mem[addr:i+15]:=ord(s[i]);

  { save AX, BX and ES into stack before executing the command }
  DisableInterrupts;
  inline($06/
         $50/
         $53/
         $cd/
         $60/
         $5B/
         $58/
         $07) ;
   EnableInterrupts;
end ;


function Voice_Driver_Exist : Boolean;
begin
  if (memw[0000:$180]<>0) and (memw[0000:$182]<>0) then {vrp_driver test}
    begin
      segment := memw[0000:$182] ;
      CMDEXEC('T');
      if mem[segment:16] = $31 then Voice_Driver_Exist := True
      else Voice_Driver_Exist := False;
    end
  else Voice_Driver_Exist := False;
end;

function GetHeapMemOkay(var p1,p2,p3 : pointer) : Boolean;
begin
  if (MaxAvail > 98304) then       { 96K block }
    begin
      GetMem(p1,_32K);
```

```
      GetMem(p2,_32K);
      GetMem(p3,_32K);
      GetHeapMemOkay := True;
    end
  else
    GetHeapMemOkay := False;
end;
(*
   1st byte of allocated heap is at
     memw[seg(ptr1):ofs(ptr1)+2]:memw[seg(ptr1):ofs(ptr1)]
   or simply
     seg(ptr1^) : ofs(ptr1^)
*)

procedure FreeAllocatedHeap;
begin
  FreeMem(ptr1,_32K);
  FreeMem(ptr2,_32K);
  FreeMem(ptr3,_32K);
end;

procedure SetVoiceDataAddr;
var
  i  : integer;
begin
  DecMemOfs := ofs(ptr1^);
  DecMemSeg := seg(ptr1^);

  if (DecMemOfs <> 0) then
    DecMemSeg := (((DecMemSeg shl 4) + DecMemOfs) shr 4 ) + 1;

  DecToHex(DecMemSeg);
  VoiceSeg := copy(HexDigit,1,length(HexDigit));
  VoiceOfs := '0000';

  if ((VoiceSeg[1]+'000') <= VoiceSeg)
    and (VoiceSeg <= (VoiceSeg[1]+'7FF')) then
    VoiceSeg := VoiceSeg[1]+'800'
  else VoiceSeg := char(byte(VoiceSeg[1])+1)+'000';

  CMDEXEC('A'+VoiceSeg);
end;

procedure SetSamplingRate(mode : char);
begin
  mode := UpCase(mode);
  CMDEXEC('M'+mode);
  SamplingRate := mode;
end;

function CalculateVoiceLength : real;
var
```

```
  I,J             : integer;
  VoiceLength     : real;
  BuffLength      : string[8];
  Rate            : real;
  buff            : real;
  power           : real;

begin
  VoiceLength := 0.0;
  BuffLength := '12345678';
  for I := 33 to 40 do begin
    BuffLength[I-33+1] := char(mem[segment:16+I-1]);
    if BuffLength[I-33+1] in ['0'..'9']  then
      buff := byte(BuffLength[I-33+1]) - byte('0')
    else
      buff := byte(BuffLength[I-33+1]) - byte('A') + 10;
    power := 1.0;
    for J := I+1 to 40 do
      power := power*16;
    VoiceLength := VoiceLength + buff*power;
  end;

  case SamplingRate of
    'Q' : Rate := 8*1024;
    'D' : Rate := 4*1024;
    'M' : Rate := 2*1024;
  end;
  VoiceLength := VoiceLength/Rate;          { in seconds }
  CalculateVoiceLength := VoiceLength;
end;

procedure VoiceStop;
begin
  CMDEXEC('E');
end;


function ExecError(code : byte) : string;
var
  ErrorString     : string;
begin
  ErrorString := '*** Saving Error ***';
  case code of
    0  : ErrorString := 'Voice Card Absent';
    1  : ErrorString := 'Invalid Function';
    2  : ErrorString := 'File Not Found';
    3  : ErrorString := 'Path Not Found';
    4  : ErrorString := 'Too Many Open File';
    5  : ErrorString := 'Access Denied';
    6  : ErrorString := 'Invalid Handle';
    12 : ErrorString := 'Invalid Access';
  end;
```

```
  ExecError := ErrorString;
end;

function LoadVoiceOkay(fname : string) : Boolean;
var
  buff      : byte;
  ErrStr    : string;
begin
  CMDEXEC('L'+fname);
  buff := mem[segment:16];
  if buff = 32 then
    begin
      VoiceLoadedFromDisk := True;
      LoadVoiceOkay := True;
      LoadedVoiceFile := fname;
    end
  else
    begin
      ErrStr := ExecError(buff);
      WriteNote(ErrStr);
      sound(1000); delay(2000); nosound;
      LoadVoiceOkay := False;
    end;
  WriteNote('VoiceSegAddr = $'+VoiceSeg);
end;


{----------------- End of VRP-70 Voice Card Routine ----------------------}

(*****************************************************************************)
(*                          Windows Routines                                 *)
(*****************************************************************************)

procedure DispNote(message : string);
var
  delta     : byte;
  SaveAttr  : byte;
begin
  SaveAttr := TextAttr;
  TextAttr := NWA;
  window(51,24,79,24);
  clrscr;
  delta := (29-length(message)) div 2;
  gotoxy(delta,wherey);
  if (length(message) > 28) then
    message := copy(message,1,28);
  write(message);
  TextAttr := SaveAttr;
end;

procedure WriteNote(message : string);
var
```

```
  SaveX,SaveY          : integer;
  WC                   : WindowCoordinates;
begin
  SaveX := wherex;
  SaveY := wherey;
  StoreWindowCoordinates(WC);
  DispNote(message);
  RestoreWindowCoordinates(WC);
  gotoxy(SaveX,SaveY);
end;

procedure DetermineWindowAttr;
begin
  if Mono then begin
    NWA := LightGray + Black*16;
    NFA := LightGray + Black*16;
    NHA := Black + LightGray*16;

    WindowAttr := White + Black*16;
    FrameAttr := LightGray + Black*16;
    HeaderAttr := Black + LightGray*16;
  end
  else begin
    NWA := White + Red*16;
    NFA := LightGray+ Red*16;
    NHA := Black + Green*16;

    WindowAttr := $0E + Cyan*16;
    FrameAttr := LightGray + Cyan*16;
    HeaderAttr := Black + Cyan*16;
  end;
end;

function InitNoteBoardWindow : Boolean;
begin
  if MakeWindow(NoteBoard,50,23,80,25,True,True,False,NWA,NFA,NHA,' Note ')
    then InitNoteBoardWindow := True
  else
    InitNoteBoardWindow := False;
end;

function InitRecordWindow : Boolean;
begin
  if MakeWindow(RecWindow,15,4,35,9,True,True,False,
              WindowAttr,FrameAttr,HeaderAttr,'Voice Record')
    then InitRecordWindow := True
  else InitRecordWindow := False;
end;

function InitPlayWindow : Boolean;
begin
  if MakeWindow(PlayWindow,32,6,72,9,True,True,False,
```

```
                WindowAttr,FrameAttr,HeaderAttr,'Voice PlayBack')
    then InitPlayWindow := True
  else InitPlayWindow := False;
end;

function InitSetFilePathWin : Boolean;
begin
  if MakeWindow(SetFilePathWin,20,6,78,9,True,True,False,
                WindowAttr,FrameAttr,HeaderAttr,'')
    then InitSetFilePathWin := True
  else InitSetFilePathWin := False;
end;

function InitDefineRecTimeWindow : Boolean;
begin
  if MakeWindow(DefineRecTimeWindow,45,7,75,10,True,True,False,
                WindowAttr,FrameAttr,HeaderAttr,'')
    then InitDefineRecTimeWindow := True
  else InitDefineRecTimeWindow := False;
end;

function InitDefinePrimaryNodeNameWindow : Boolean;
begin
  if MakeWindow(DefinePrimaryNodeNameWindow,45,8,75,11,True,True,False,
                WindowAttr,FrameAttr,HeaderAttr,'')
    then InitDefinePrimaryNodeNameWindow := True
  else InitDefinePrimaryNodeNameWindow := False;
end;

function InitDefineNodePrefixWindow : Boolean;
begin
  if MakeWindow(DefineNodePrefixWindow,45,11,75,14,True,True,False,
                WindowAttr,FrameAttr,HeaderAttr,'')
    then InitDefineNodePrefixWindow := True
  else InitDefineNodePrefixWindow := False;
end;


function InitDefineActionAfterAnsWindow : Boolean;
begin
  if MakeWindow(DefineActionAfterAnsWindow,45,9,75,12,True,True,False,
                WindowAttr,FrameAttr,HeaderAttr,'')
    then InitDefineActionAfterAnsWindow := True
  else InitDefineActionAfterAnsWindow  := False;
end;

function InitDefineGreetMessageWindow : Boolean;
begin
  if MakeWindow(DefineGreetMessageWindow,45,10,75,13,True,True,False,
                WindowAttr,FrameAttr,HeaderAttr,'')
    then InitDefineGreetMessageWindow  := True
  else InitDefineGreetMessageWindow  := False;
```

```
end;

function InitDispConfigInfoWindow : Boolean;
begin
  if MakeWindow(DispConfigInfoWindow,20,8,78,19,True,True,False,
                WindowAttr,FrameAttr,HeaderAttr,'Current Configuration')
    then InitDispConfigInfoWindow := True
  else InitDispConfigInfoWindow := False;
end;

function InitSaveVoiceMessage : Boolean;
begin
  if MakeWindow(SaveVoiceMessageWindow,50,6,80,10,True,True,False,
                WindowAttr,FrameAttr,HeaderAttr,'Save Voice')
    then InitSaveVoiceMessage := True
  else InitSaveVoiceMessage := False;
end;

function InitApplicationWindow : Boolean;
begin
  if MakeWindow(ApplicationWindow,1,7,80,22,True,True,False,
                BlackOnLtGray,YellowOnLtGray,WhiteOnRed,'Application Generator')
    then InitApplicationWindow := True
  else InitApplicationWindow := False;
end;

function InitSaveApplicationWindow : Boolean;
begin
  if MakeWindow(SaveApplicationWindow,20,10,60,13,True,True,False,
                YellowOnBlue,YellowOnBlue,YellowOnRed,'Save Application')
    then InitSaveApplicationWindow := True
  else InitSaveApplicationWindow := False;
end;

function InitGetApplicationWindow : Boolean;
begin
  if MakeWindow(GetApplicationWindow,10,8,70,18,True,True,False,
                WindowAttr,FrameAttr,HeaderAttr,'Get Application')
    then InitGetApplicationWindow := True
  else InitGetApplicationWindow := False;
end;

function InitWindowsOkay : Boolean;
begin
  if InitNoteBoardWindow and InitRecordWindow and InitPlayWindow
     and InitSetFilePathWin and InitDefineRecTimeWindow and
     InitDispConfigInfoWindow and InitSaveVoiceMessage and
     InitApplicationWindow and InitSaveApplicationWindow and
     InitGetApplicationWindow and InitDefinePrimaryNodeNameWindow and
     InitDefineNodePrefixWindow and InitDefineGreetMessageWindow and
     InitDefineActionAfterAnsWindow
    then InitWindowsOkay := True
```

```
  else InitWindowsOkay := False;
end;

{------------------------- End of Window Routines -------------------------}



(*****************************************************************************)
(*                  Main Menu Supporting Routines                            *)
(*****************************************************************************)


procedure RecordVoice;
var
  hr,min,sec,sec100  : word;
  remtime            : real;
  code               : integer;
  AvailTime          : real;
  key                : char;
begin
  if not DisplayWindow(RecWindow) then
    exit;

  Val(RecTime,AvailTime,code);

  gotoxy(2,1);
  write('* Press a key to');
  gotoxy(4,2);
  write('start ...');
  gotoxy(3,4);
  write('<ESC> to quit');
  repeat until keypressed;
  key := readkey;

  if key <> ESC then begin
    if keypressed then key := readkey;
    playflag := True;
    clrscr;
    gotoxy(3,1);
    write('# Remain Time #');
    GetTime(hr,min,sec,sec100);
    CMDEXEC('R'+RecTime);
    repeat
      { Show Remaining Record Time }
      gotoxy(4,3);
      remtime := AvailTime - StopWatch(hr,min,sec,sec100);
      if remtime < 0 then remtime := 0;
      write(remtime:4:2,' Seconds');
      clreol;
    until (mem[segment:16] = 32) or keypressed;
    if keypressed then begin
```

```
      waitkey;
      VoiceStop;
    end;
    SaveVoiceTime := CalculateVoiceLength;
    WriteNote('Press Any Key');
    PseudoMusic;
    waitkey;
  end;

  DummyWindow := EraseTopWindow;
  WriteNote('VoiceSegAddr = $'+VoiceSeg);
  VoiceLoadedFromDisk := False;
end;


procedure VoicePlayBack(keyword : string);
const
  filename : string = '';
var
  hr,min,sec,sec100   : word;
  remtime             : real;
  AvailTime           : real;
  Mask                : string;

begin
  if (keyword = 'Disk') then
    begin
      Mask := AppendPath(FilePath) + '*.*';
      SelectFile(Mask,filename);
      if (filename = '') or (not LoadVoiceOkay(filename)) then exit;
      playflag := True;
      PlayInterrupted := False;
      SaveVoiceTime := CalculateVoiceLength;
    end;
  if not playflag then
    begin
      gotoxy(1,1);
      WriteNote('No Voice Message in RAM !');
      sound(2000); delay(1000); nosound;
      WriteNote('VoiceSegAddr = $'+VoiceSeg);
      exit;
    end;
  if not DisplayWindow(PlayWindow) then begin
    WriteNote('PlayWindow Create Error');
    sound(5000); delay(2000); nosound;
    WriteNote('VoiceSegAddr = $'+VoiceSeg);
  end;
  clrscr;
  gotoxy(8,1);
  writeln(#19,' Remaining Play Time ',#19);
  WriteNote('Any Key To Stop Playing');
  if PlayInterrupted then
```

```
    AvailTime := SaveVoiceTime
  else
    AvailTime := CalculateVoiceLength;

  CMDEXEC('P');
  GetTime(hr,min,sec,sec100);
  repeat
    { Show Remaining Record Time }
    gotoxy(13,2);
    remtime := AvailTime - StopWatch(hr,min,sec,sec100);
    if remtime < 0 then remtime := 0;
    write(remtime:4:2,' Seconds');
    clreol;
  until (mem[segment:16] = 32) or keypressed;
  if keypressed then
    begin
      waitkey;
      VoiceStop;
      PlayInterrupted := True;
    end
  else
    PlayInterrupted := False;
  WriteNote('Hit Any Key');
  PseudoMusic;
  waitkey;
  DummyWindow := EraseTopWindow;
  WriteNote('VoiceSegAddr = $'+VoiceSeg);
end;

procedure SelectSamplingRate(mode : char);
begin
  SetSamplingRate(mode);
  TpMenu.EraseCurrentSubMenu(_M_);
end;


Procedure SetFilePath;
const
   trial        : byte    = 5;
   NewFilePath  : string  = '';
var
  dummy         : Boolean;
  Escaped       : Boolean;
  DispPath      : String;


begin
  if not DisplayWindow(SetFilePathWin) then
    exit;
  MoveWindow(-2,1,5);
  if FilePath = '' then
    writeln('Old File Path : Current Directory')
```

```
  else
    begin
      DispPath := copy(FilePath,1,40);
      writeln('Old File Path : ',DispPath);
    end;

  TpEdit.WindowRelative := True;
  TpEdit.EditSize := 40;
  TpEdit.ReadString('New File Path : ',
                    2,1,254,
                    BlackOnCyan,WhiteOnCyan,RedOnCyan,
                    Escaped,
                    NewFilePath);

  if not Escaped then FilePath := NewFilePath;

  delay(500);
  MoveWindow(2,-1,5);
  dummyWindow := EraseTopWindow;
end;

procedure DefineRecTime;
const
  NewRecTime  : String = '';
var
  Escaped   : Boolean;

begin
  if not DisplayWindow(DefineRecTimeWindow) then
    exit;
  MoveWindow(-3,2,4);
  MoveWindow(-1,-2,2);
  writeln('Old Record Time : ',RecTime,' Sec');

  TpEdit.WindowRelative := True;
  TpEdit.EditSize := 5;
  TpEdit.ReadString('New Record Time : ',
                    2,1,4,
                    BlackOnCyan,WhiteOnCyan,RedOnCyan,
                    Escaped,
                    NewRecTime);

  if not ValidNumberString(NewRecTime) then
    begin
      WriteNote('Invalid Record Time');
      sound(2000); delay(1000); nosound;
      WriteNote('VoiceSegAddr = $'+VoiceSeg);
    end
  else if (NewRecTime <> '') then
    RecTime := NewRecTime;

  delay(500);
```

```
  MoveWindow(1,2,2);
  MoveWindow(3,-2,4);
  dummyWindow := EraseTopWindow;
end;

Procedure DispConfigInfo;
var
  Rate      : char;
  path      : string;
  Escaped   : Boolean;
  fname     : string;
begin
  if not DisplayWindow(DispConfigInfoWindow) then
    exit;

  case SamplingRate of
    'S' : Rate := '2';
    'D' : Rate := '4';
    'Q' : Rate := '8';
  end;
  if FilePath = '' then
    path := 'Current Directory'
  else
    path := FilePath;

  MoveWindow(0,1,3);
  MoveWindow(-1,0,15);

  gotoxy(1,1);
  writeln('File Search Path     : ',path);
  writeln('Voice Sampling Rate  : ',Rate,'K Byte/Sec');
  writeln('Recording Time       : ',Rectime,' Seconds');
  writeln('Primary Node Name    : ',PrimaryNodeName);
  writeln('Node Prefix          : ',NodePrefix);
  writeln('Action After Ans.    : ',ActionAfterAns);
  writeln('Greeting Message     : ',GreetMessage);
  writeln;

  write('*** Press Any Key ***');
  waitkey;

  MoveWindow(0,-1,3);
  MoveWindow(1,0,15);
  DummyWindow := EraseTopWindow;
end;

procedure SaveVoiceMessage;
var
  fname     : string;
  buff      : byte;
  Escaped   : Boolean;
  WinMoved  : Boolean;
```

```
  WinLength : integer;
  Okay      : Boolean;
  Success   : integer;

begin
  if not playflag then begin
    WriteNote('No Voice In RAM');
    sound(2000); delay(1000); nosound;
    WriteNote('VoiceSegAddr = $'+VoiceSeg);
    exit;
  end;

  if not (DisplayWindow(SaveVoiceMessageWindow)) then exit;

  clrscr;
  fname := '';
  WinMoved := False;
  TpEdit.EditSize := 15;
  TpEdit.WindowRelative := True;
  TpEdit.ReadString('Save To '+#16+' ',
                    1,1,
                    12,                          { Max 12 char·}
                    WhiteOnCyan,BlackOnCyan,RedOnCyan,
                    Escaped,fname);
  if (not Escaped) and (fname <> '') then
    begin
      writeln;
      writeln('Thinking ... ');
      fname := AppendPath(FilePath) + fname;
      if VoiceLoadedFromDisk and ExistFile(fname) then
        begin
          TpEdit.WindowRelative := True;
          TpEdit.ShowReadChar := True;
          if (length('OverWrite '+fname+ '[N]   ') >= 29) then
            begin
              WinLength := length('OverWrite '+fname+ '[N]   ') + 2;
              MoveWindow(29-WinLength,0,1);
              Okay := ResizeWindow(WinLength-29,0,' ');
              WinMoved := True;
            end
          else
            WinMoved := False;
          if TpEdit.YesOrNo('OverWrite '+fname,
                            3,1,
                            RedOnCyan,
                            'N')  then
            begin
              gotoxy(1,wherey);
              clreol;
              write('In Processing . . . ');
              success := ExecDos('Copy '+LoadedVoiceFile+' !Voice.$D$ > NUL',
                                 True,Nil);
```

```
            success := ExecDos('Copy !Voice.$D$ '+fname+' > NUL',
                               True,Nil);
            success := ExecDos('Del !Voice.$D$ ',
                               True,Nil);
          end;
        end
      else begin
        fname := fname +'.msg';
        CMDEXEC('S'+fname);
        buff := mem[segment:16];
        if buff = 32 then
          write('Saving Success !')
        else
          begin
            fname := ExecError(buff);
            write(fname);
            sound(2000); delay(200); nosound;
          end;
      end;
    WriteNote('Press Any Key');
    PseudoMusic;
    waitkey;
  end;
  if WinMoved then begin
    Okay := ResizeWindow(29-WinLength,0,' ');
    MoveWindow(WinLength-29,0,1);
  end;
  DummyWindow := EraseTopWindow;
  WriteNote('VoiceSegAddr = $'+VoiceSeg);
end;

procedure DefinePrimaryNodeName;
const
  NewPrimaryNodeName : String = '';
var
  Escaped   : Boolean;

begin
  if not DisplayWindow(DefinePrimaryNodeNameWindow) then
    exit;
  MoveWindow(-3,2,4);
  MoveWindow(-1,-2,2);
  writeln('Primary Node : ',PrimaryNodeName);

  TpEdit.WindowRelative := True;
  TpEdit.EditSize := 13;
  TpEdit.ReadString('New Name : ',
                    2,1,12,
                    BlackOnCyan,WhiteOnCyan,RedOnCyan,
                    Escaped,
                    NewPrimaryNodeName);
```

```
  if (NewPrimaryNodeName <> '') then
    PrimaryNodeName := NewPrimaryNodeName;

  delay(500);
  MoveWindow(1,2,2);
  MoveWindow(3,-2,4);
  dummyWindow := EraseTopWindow;
end;

procedure DefineActionAfterAnswer;
const
  NewActionAfterAns : String = '';
var
  Escaped   : Boolean;

begin
  if not DisplayWindow(DefineActionAfterAnsWindow) then
    exit;
  MoveWindow(-3,2,4);
  MoveWindow(-1,-2,2);
  writeln('Action : ',ActionAfterAns);

  TpEdit.WindowRelative := True;
  TpEdit.EditSize := 13;
  TpEdit.ReadString('New Action : ',
                    2,1,12,
                    BlackOnCyan,WhiteOnCyan,RedOnCyan,
                    Escaped,
                    NewActionAfterAns);

  if (NewActionAfterAns <> '') then
    ActionAfterAns := NewActionAfterAns;

  delay(500);
  MoveWindow(1,2,2);
  MoveWindow(3,-2,4);
  dummyWindow := EraseTopWindow;
end;

procedure DefineGreetingMessage;
const
  NewGreetMessage : String = '';
var
  Escaped   : Boolean;

begin
  if not DisplayWindow(DefineGreetMessageWindow) then
    exit;
  MoveWindow(-3,1,4);
  MoveWindow(-1,-1,2);
  writeln('Greet Message : ',GreetMessage);
```

```
  TpEdit.WindowRelative := True;
  TpEdit.EditSize := 13;
  TpEdit.ReadString('New Message : ',
                    2,1,12,
                    BlackOnCyan,WhiteOnCyan,RedOnCyan,
                    Escaped,
                    NewGreetMessage);

  if (NewGreetMessage <> '') then
    GreetMessage := NewGreetMessage;

  delay(500);
  MoveWindow(1,1,2);
  MoveWindow(3,-1,4);
  dummyWindow := EraseTopWindow;
end;

procedure DefineNodePrefix;
const
  NewNodePrefix : String = '';
var
  Escaped   : Boolean;

begin
  if not DisplayWindow(DefineNodePrefixWindow) then
    exit;
  MoveWindow(-3,1,4);
  MoveWindow(-1,-1,2);
  writeln('Node Prefix : ',NodePrefix);

  TpEdit.WindowRelative := True;
  TpEdit.EditSize := 4;
  TpEdit.ReadString('New Prefix : ',
                    2,1,3,
                    BlackOnCyan,WhiteOnCyan,RedOnCyan,
                    Escaped,
                    NewNodePrefix);

  if (NewNodePrefix <> '') then
    NodePrefix := NewNodePrefix;

  delay(500);
  MoveWindow(1,1,2);
  MoveWindow(3,-1,4);
  dummyWindow := EraseTopWindow;
end;


procedure CreateNode;
var
  I  : integer;
begin
```

```
  Inc(Node.LevelIdentifier);
  Node.PreviousNode := Node.NodeID;
  Inc(SystemNodeNumber);
  Node.NodeID := SystemNodeNumber;
  PresentNodeName := NodePrefix + IntToStr(SystemNodeNumber);
  for I := 1 to 12 do begin
    Node.DigitToNode[I].NextNode := -1;    { ==> invalid next node }
    Node.DigitToNode[I].Action := 'Nil';
    Node.DigitToNode[I].VoiceFile := 'Nil';
  end;
end;

procedure CreatePrimaryNode;
var
  I     : integer;
begin
  PresentNodeName := PrimaryNodeName;
  Node.NodeID := 0;
  Node.PreviousNode := -1;
  Node.LevelIdentifier := 0;
  for I := 1 to 12 do begin
    Node.DigitToNode[I].NextNode := -1;    { ==> invalid next node }
    Node.DigitToNode[I].Action := 'Nil';
    Node.DigitToNode[I].VoiceFile := 'Nil';
  end;
end;

procedure ShowNodeStructure;
var
  I  : integer;
begin
  if not DisplayWindow(ApplicationWindow) then exit;
  writeln;
  writeln('  └─┐');
  writeln('    ├─ 1 - ');
  writeln('    ├─ 2 - ');
  writeln('    ├─ 3 - ');
  writeln('    ├─ 4 - ');
  writeln('    ├─ 5 - ');
  writeln('    ├─ 6 - ');
  writeln('    ├─ 7 - ');
  writeln('    ├─ 8 - ');
  writeln('    ├─ 9 - ');
  writeln('    ├─ 0 - ');
  writeln('    ├─ * - ');
  write  ('    └─ # - ');
  gotoxy(1,1);
  with node do begin
    if (PreviousNode = -1) and (LevelIdentifier = 0) then
      begin
        IsPrimaryNode := True;
        write('Primary Node : ');
```

```
        WriteWithChangeTextAttr(YellowOnCyan,PrimaryNodeName);
        gotoxy(32,1);
        write('Action : ');
        WriteWithChangeTextAttr(YellowOnCyan,ActionAfterAns);
        gotoxy(55,1);
        write('Voice : ');
        WriteWithChangeTextAttr(YellowOnCyan,GreetMessage);
      end
    else
      begin
        IsPrimaryNode := False;
        write('NODE : ');
        WriteWithChangeTextAttr(YellowOnCyan,PresentNodeName);
      end;
  end;
  gotoxy(15,2);
  WriteWithChangeTextAttr(BlueOnLtGray,'Voice');
  gotoxy(36,2);
  WriteWithChangeTextAttr(BlueOnLtGray,'Action');

  for I := 1 to 12 do
    with Node.DigitToNode[I] do
      begin
        gotoxy(15,2+I);
        WriteWithChangeTextAttr(YellowOnCyan,VoiceFile);
        gotoxy(36,wherey);
        WriteWithChangeTextAttr(YellowOnCyan,Action);
      end;

  gotoxy(66,14);
  WriteWithChangeTextAttr(YellowOnBlue,'Level :: ');
  WriteWithChangeTextAttr(YellowOnBlue,IntToStr(Node.LevelIdentifier));
end;

function SaveNodeToDisk : Boolean;
var
  f             : file;
  ResultWritten  : word;

begin
  assign(f,AppendPath(FilePath) + PresentNodeName);
  {$I-}
  rewrite(f,1);
  {$I+}
  if (IOResult = 0) then
    begin
      BlockWrite(f,node,sizeof(node),ResultWritten);
      close(f);
      SaveNodeToDisk := True;
    end
  else
    begin
```

```
      { Try to save to current directory }
      sound(500); delay(500); nosound;
      FilePath := '';
      assign(f,AppendPath(FilePath) + PresentNodeName);
      {$I-} rewrite(f,1); {$I+}
      if (IOResult = 0) then
        begin
          BlockWrite(f,node,sizeof(node),ResultWritten);
          close(f);
          SaveNodeToDisk := True;
        end
      else
        SaveNodeToDisk := False;
    end;
end;

function LoadNodeFromDisk(ID : longint) : Boolean;
var
  f          : file;
  fname      : string;
  ResultRead : word;
begin
  if (ID = 0) then
    fname := AppendPath(FilePath) + PrimaryNodeName
  else
    fname := AppendPath(FilePath) + NodePrefix + IntToStr(ID);
  assign(f,fname);
  {$I-}
  reset(f,1);
  {$I+}
  if (IOResult = 0) then
    begin
      BlockRead(f,node,sizeof(node),ResultRead);
      close(f);
      LoadNodeFromDisk := True;
    end
  else
    LoadNodeFromDisk := False;
end;

function StepForwardOrBackward(Item : Word) : string;  (* Must use far call *)
begin
  case Item of
    1 : StepForwardOrBackward := ' Step Forward '  ;
    2 : StepForwardOrBackward := ' Back One Level ';
  end;
end;

function VoiceInMemoryOrDisk (Item : Word) : String;   (* Must use far call *)
begin
  case Item of
    1 : VoiceInMemoryOrDisk := 'Memory';
```

```
    2 : VoiceInMemoryOrDisk := 'Disk';
  end;
end;

function ChooseSamplingRate (Item : Word) : string; (* Must be far call *)
begin
  case Item of
    1 : ChooseSamplingRate := '2K Bytes/Sec';
    2 : ChooseSamplingRate := '4K Bytes/Sec';
    3 : ChooseSamplingRate := '8K Bytes/Sec';
  end;
end;

function SaveApplication(ToFile : string) : Boolean;
var
  SaveOrNot    : Boolean;
  Escaped      : Boolean;
  Config       : ConfigStructure;
  filename     : string;
  f1,f2        : file;
  index        : longint;
  NumWritten   : integer;
  NumRead      : integer;
  logo         : string;

  procedure ReportIOError(ErrorIndex : longint);
  var
    ErrorMessage : string;
  begin
    if (ErrorIndex <= 0) then
      case ErrorIndex of
        -1  : ErrorMessage := ' ## Unable To Open '+filename;
         0  : ErrorMessage := ' ## Primary Node File Not Found';
      end
    else
      ErrorMessage := ' ## File of Node #'+IntToStr(ErrorIndex)+' Not Found.';
    Write(ErrorMessage);
    sound(3000); delay(1000); nosound;
    WriteNote('Press Any Key');
    waitkey;
  end;

begin
  if NothingToSave then
    begin
      WriteNote('No Unsaved Application');
      PseudoMusic;
      waitkey;
      WriteNote('VoiceSeg = $'+VoiceSeg);
      exit;
    end;
```

```
if not DisplayWindow(SaveApplicationWindow) then exit;

filename := ToFile;

TpEdit.WindowRelative := True;
TpEdit.EditSize := 15;
SaveOrNot := YesOrNo('Save Application To Disk ?',1,2,YellowOnBlue,'Y');

if SaveOrNot then
  begin
    writeln;
    if filename = '' then begin
      ReadString('Save To : ',2,2,
                 12,                    { Max 12 char }
                 YellowOnBlue,LtCyanOnBlue,WhiteOnBlue,
                 Escaped,filename);
      if Escaped or (filename = '') then
        begin
          DummyWindow := EraseTopWindow;
          SaveApplication := False;
          exit;
        end;
      filename := AppendPath(FilePath) + filename;
    end
    else
      write(' Saving '+ filename + '...');
    with Config do
      begin
        _FilePath        := FilePath;
        _SamplingRate    := SamplingRate;
        _RecTime         := RecTime;
        _PrimaryNodeName := PrimaryNodeName;
        _NodePrefix      := NodePrefix;
        _ActionAfterAns  := ActionAfterAns;
        _GreetMessage    := GreetMessage;
        _SystemNodeNumber := SystemNodeNumber;
      end;
    assign(f1,filename);
    {$I-} rewrite(f1,1); {$I+}
    if (IOResult <> 0) then
      begin
        ReportIOError(-1);
        NothingToSave := False;
        DummyWindow := EraseTopWindow;
        SaveApplication := False;
        exit;
      end;

    logo := signature + EOF;
    blockwrite(f1,logo,sizeof(logo),NumWritten);
    blockwrite(f1,Config,sizeof(Config),NumWritten);
```

```
      assign(f2,AppendPath(FilePath) + PrimaryNodeName);
      {$I-} reset(f2,1); {$I+}
      if (IOResult <> 0) then
        begin
          ReportIOError(0);
          NothingToSave := False;
          DummyWindow := EraseTopWindow;
          close(f1);
          SaveApplication := False;
          exit;
        end
      else
        begin
          blockread(f2,node,sizeof(node),NumRead);
          blockwrite(f1,node,sizeof(node),NumWritten);
          close(f2);
        end;

      for index := 1 to SystemNodeNumber do
        begin
          assign(f2,AppendPath(FilePath)+NodePrefix+IntToStr(index));
          {$I-} reset(f2,1); {$I+}
          if (IOResult = 0) then
            begin
              blockread(f2,node,sizeof(node),NumRead);
              blockwrite(f1,node,sizeof(node),NumWritten);
              close(f2);
            end
          else
            begin
              ReportIOError(index);
              NothingToSave := False;
              DummyWindow := EraseTopWindow;
              close(f1);
              SaveApplication := False;
              exit;
            end;
        end;

      close(f1);
    end;

  (* Up to here, everything is okay *)
  DummyWindow := EraseTopWindow;
  NothingToSave := True;
  SaveApplication := True;
end;

procedure NewApplication;
const
  Left    = #75;
  Down    = #80;
```

```
  Right   = #77;
  UP      = #72;
  Alt_P   = #25;
  Alt_R   = #19;
  Alt_S   = #31;


  PickColor : PickColorArray =
    ( $07,              { unselected item }
      YellowOnBlue,     { Window Frame    }
      $70,              { Window Title    }
      BlackOnGreen,     { Selected Item   }
      $07,              { Alt unselected item   }
      $0F               { Alternate select item }
    );

  PickColorMono : PickColorArray =
    ( $07,              { unselected item }
      YellowOnBlue,     { Window Frame    }
      $70,              { Window Title    }
      BlackOnLtGray,    { Selected Item   }
      $07,              { Alt unselected item   }
      $0F               { Alternate select item }
    );

var
  key1,key2   : Char;
  I,J,X,Y     : integer;
  P,Q,R       : integer;
  SaveLength  : integer;
  SaveChar    : Char;
  SaveAttr    : byte;
  strptr      : ^string;
  Escaped     : Boolean;
  choice      : Word;
  quit        : Boolean;

  { Nested Function/Procedure }
  procedure NewApplicationScreen;
  var
    index : integer;
  begin
    ShowNodeStructure;
    Window(1,1,80,25);
    gotoxy(1,8);
    Y := 8;
    for index := 1 to 80 do
      begin
        WriteSelectPointer(SaveChar,SaveAttr,index,Y);
        delay(5);
        FastWrite(SaveChar+'',Y,index,SaveAttr);
      end;
```

```
  I := 16;
  J := 8;
  X := I;
  Y := J;
  WriteSelectPointer(SaveChar,SaveAttr,I,J);
  WriteNote('Use Arrow Keys To Select');
end;

procedure NextPointerPosition (key : char; var newx,newy : integer);
begin
  Case key of
    Left    : begin
                if (wherey = 8) then            { for primary node }
                  begin
                    if IsPrimaryNode then
                      case wherex of
                        16 : newx := 63;
                        41 : newx := 16;
                        63 : newx := 41;
                      end;
                  end
                else
                  case wherex of
                    9  : newx := 36;
                    15 : newx := 9;
                    36 : newx := 15;
                  end;
              end;
    Down    : begin
                if (wherey = 8) then
                  begin
                    newx := 9;
                    newy := 10;
                  end
                else
                  begin
                    Inc(newy);
                    if (newy = 22) then
                      begin
                        if IsPrimaryNode then
                          begin
                            newx := 16;
                            newy := 8;
                          end
                        else
                          begin
                            newx := wherex;
                            newy := 10;
                          end;
                      end;
                  end;
              end;
```

```
    Right   : begin
                if (wherey = 8) then
                  begin
                    if IsPrimaryNode then
                       case wherex of
                        16 : newx := 41;
                        41 : newx := 63;
                        63 : newx := 16;
                      end;
                  end
                else
                  case wherex of
                    9  : newx := 15;
                    15 : newx := 36;
                    36 : newx := 9;
                  end;
              end;
    Up      : begin
                if (wherey = 8) then
                  begin
                    newx := 9;
                    newy := 21;
                  end
                else
                  begin
                    Dec(newy);
                    if (newy = 9) then begin
                      if IsPrimaryNode then
                        begin
                          newx := 16;
                          newy := 8;
                        end
                      else
                        begin
                          newx := wherex;
                          newy := 21;
                        end;
                    end;
                  end;
              end;
  end;
end;

procedure VoiceRecordInApplicationWindow;
begin
  RecordVoice;
end;

procedure VoicePlayBlackInApplicationWindow;
begin
  if PickWindow(@VoiceInMemoryOrDisk,
                2,                      { 2 choices only }
```

```
                wherex+1,wherey+1,wherex+10,wherey+5,
                True,                   { Draw Frame }
                PickColor,
                '',                     { No Title }
                choice) then
    if PickCmdNum = PKSSelect then
      begin
        case choice of
          1 : VoicePlayBack('Memory');
          2 : VoicePlayBack('Disk');
        end;
      end;
end;

procedure SaveVoiceMessageInApplicationWindow;
begin
  SaveVoiceMessage;
end;

procedure StepOneLevelForward(SaveRow : integer);
var
  SaveDigitToNode    : longint;
begin
  SaveDigitToNode := Node.DigitToNode[SaveRow-10+1].NextNode;
  if (SaveDigitToNode < 0) then
    Node.DigitToNode[SaveRow-10+1].NextNode := SystemNodeNumber + 1;

  if not SaveNodeToDisk then begin
    WriteNote('Current Node Save Error');
    sound(2000); delay(2000); nosound;
    WriteNote('Use Arrows Keys To Select');
    Node.DigitToNode[SaveRow-10+1].NextNode := SaveDigitToNode;
    exit;
  end;

  if (SaveDigitToNode > 0) then
    begin
      if not LoadNodeFromDisk(SaveDigitToNode) then
        begin
          WriteNote('Node '+ IntToStr(SaveDigitToNode) + ' Load Error');
          sound(2000); delay(2000); nosound;
          WriteNote('Use Arrows Keys To Select');
          exit;
        end
      else
        begin
          if (Node.LevelIdentifier = 0) and (Node.PreviousNode = -1) then
            PresentNodeName := PrimaryNodeName
          else
            PresentNodeName := NodePrefix + IntToStr(Node.NodeID);
        end;
    end
```

```
    else
      CreateNode;
    DummyWindow := EraseTopWindow;
    ShowNodeStructure;
    window(1,1,80,25);
    I := 9;
    J := 10;
    X := I;
    Y := J;
    WriteSelectPointer(SaveChar,SaveAttr,I,J);
    WriteNote('Use Arrows Keys To Select');
  end;

  procedure BackOneLevel;
  begin
    if not SaveNodeToDisk then begin
      WriteNote('File I/O Error');
      sound(2000); delay(2000); nosound;
      WriteNote('Use Arrows Keys To Select');
      exit;
    end;
    if not LoadNodeFromDisk(Node.PreviousNode) then
      exit;
    if (Node.LevelIdentifier = 0) and (Node.PreviousNode = -1) then
      PresentNodeName := PrimaryNodeName
    else
      PresentNodeName := NodePrefix + IntToStr(Node.NodeID);
    DummyWindow := EraseTopWindow;
    ShowNodeStructure;
    window(1,1,80,25);
    I := 9;
    J := 10;
    X := I;
    Y := J;
    WriteSelectPointer(SaveChar,SaveAttr,I,J);
    WriteNote('Use Arrows Keys To Select');
  end;

begin                    { Main routine of NewApplication }
  if Mono then
    PickColor := PickColorMono;

  if ANewApplication then
    begin
      SystemNodeNumber := 0;
      CreatePrimaryNode;
    end;

  NewApplicationScreen;
  quit := False;
  NothingToSave := False;
  repeat
```

```
HiddenCursor;
key1 := readkey;
if key1 = #0 then begin
  key2 := readkey;
  case key2 of
    Alt_P : VoicePlayBlackInApplicationWindow;
    Alt_R : VoiceRecordInApplicationWindow;
    Alt_S : SaveVoiceMessageInApplicationWindow;
  end;
  NextPointerPosition(key2,X,Y);
  FastWrite(SaveChar+'',J,I,SaveAttr);
  I := X;
  J := Y;
  WriteSelectPointer(SaveChar,SaveAttr,I,J);
end;
if (key1 = CR) then begin
  if (wherex <> 9) then         { Field Update }
    begin
      case wherex of
        15 : strptr := addr(Node.DigitToNode[wherey-10+1].VoiceFile);
        36 : strptr := addr(Node.DigitToNode[wherey-10+1].Action);
        16 : strptr := addr(PresentNodeName);
        41 : strptr := addr(ActionAfterAns);
        63 : strptr := addr(GreetMessage);
      end;
      SaveLength := length(strptr^);
      P := I + 1;
      Q := J;
      if MakeWindow(EditWindow,
                    wherex+1,wherey+1,
                    wherex+15,wherey+3,
                    True,True,False,
                    LtCyanOnBrown,LtCyanOnBrown,LtCyanOnBrown,
                    '') then
        if DisplayWindow(EditWindow) then begin
          NormalCursor;
          TpEdit.WindowRelative := True;
          ReadString('',1,1,12,TextAttr,TextAttr,BlackOnLtGray,
                     Escaped,strptr^);
          DummyWindow := EraseTopWindow;
          if not Escaped then begin
            gotoxy(P,Q);
            TextAttr := BlackOnLtGray;
            for R := 1 to SaveLength do
              write(' ');
            gotoxy(P,Q);
            TextAttr := YellowOnCyan;
            write(strptr^);
            gotoxy(X,Y);
            if IsPrimaryNode then
              PrimaryNodeName := PresentNodeName;
          end;
```

```
          end;
        end
      else      { Digit Selected }
        begin
          choice := 1;
          if PickWindow(@StepForwardOrBackward,
                        2,                          { 2 choices only }
                        wherex+1,wherey+1,wherex+18,wherey+5,
                        True,                       { Draw Frame }
                        PickColor,
                        '',                         { No Title }
                        choice) then
            if PickCmdNum = PKSSelect then
              begin
                case choice of
                  1 :  StepOneLevelForward(J);
                  2 :  BackOneLevel;
                end;
              end;
        end;
    end;
    if (key1 = ESC) then
      if SaveNodeToDisk then        (* save current node to disk *)
        if ANewApplication then
          quit := SaveApplication('')
        else
          quit := True;
  until (key1 = ESC) and quit;

  DummyWindow := EraseTopWindow;
  WriteNote('VoiceSeg = $'+VoiceSeg);
  NormalCursor;
  strptr := nil;
end;

procedure ErrorHandling(var ErrorFlag : Boolean ; ErrorMessage : string);
begin
  writeln;
  writeln(ErrorMessage);
  writeln;
  writeln('*** Hit Any Key ***');
  sound(1000); delay(500); nosound;
  waitkey;
  ErrorFlag := True;
end;


procedure GetApplicationFile;
var
  Mask          : string;
  f1,f2         : file;
  sign          : string;
```

```
  NumRead       : word;
  NumWritten    : word;
  Config        : ConfigStructure;
  I             : longint;
  TestString    : string;
  ErrorOccurred : Boolean;
  fname         : string;
  Appfname      : string;

begin
  ANewApplication := False;
  ErrorOccurred := False;

  { Chosen Application By User }
  Appfname := '';
  Mask := AppendPath(FilePath) + '*.*';
  SelectFile(Mask,Appfname);
  if (Appfname = '') or not DisplayWindow(GetApplicationWindow) then
    begin
      ANewApplication := True;
      exit;
    end;

  clrscr;
  writeln('Loading Application File '+ Appfname);
  assign(f1,Appfname);
  {$I-}
  reset(f1,1);
  {$I+}
  if (IOResult = 0) and not ErrorOccurred then begin
    writeln('Verifying File Signature ...');
    blockread(f1,sign,sizeof(signature),NumRead);
    TestString := Signature + EOF;
    if (sign = TestString) and not ErrorOccurred then begin
      writeln('Retrieving Configuration from file ...');
      blockread(f1,Config,sizeof(Config),NumRead);
      with Config do
        begin
          FilePath            := _FilePath;
          SamplingRate        := _SamplingRate;
          RecTime             := _RecTime;
          PrimaryNodeName     := _PrimaryNodeName;
          NodePrefix          := _NodePrefix;
          ActionAfterAns      := _ActionAfterAns;
          GreetMessage        := _GreetMessage;
          SystemNodeNumber    := _SystemNodeNumber;
        end;
      blockread(f1,node,sizeof(node),NumRead);
      fname := AppendPath(FilePath)+PrimaryNodeName;
      assign(f2,fname);
      {$I-}
      Rewrite(f2,1);
```

```
    {$I+}
    if (IOResult = 0) and not ErrorOccurred then
      begin
        writeln('Creating Primary Node File ...');
        blockwrite(f2,node,sizeof(node),NumWritten);
        close(f2);
      end
    else (* Primary Node File Create Error *)
      begin
        close(f1);
        ErrorHandling(ErrorOccurred,fname + ' creation error !');
        ANewApplication := True;
        DummyWindow := EraseTopWindow;
        exit;
      end;

    for I := 1 to SystemNodeNumber do
      begin
        fname := AppendPath(FilePath) + NodePrefix + IntToStr(I);
        assign(f2,fname);
        {$I-}
        rewrite(f2,1);
        {$I+}
        if (IOResult = 0) and not ErrorOccurred then
          begin
            writeln('Creating Node '+ IntToStr(I) + ' File ...');
            blockread(f1,node,sizeof(node),NumRead);
            blockwrite(f2,node,sizeof(node),NumWritten);
            close(f2);
          end
        else  (* Node #I File Create Error    *)
          begin
            close(f1);
            ErrorHandling(ErrorOccurred,fname + 'creation error !');
            DummyWindow := EraseTopWindow;
            ANewApplication := True;
            exit;
          end;
      end;
  end
  else
    (* Not A Valid Application File *)
    ErrorHandling(ErrorOccurred,'Not a valid application file !!!');
end
else
  (* Application File Open Error  *)
  ErrorHandling(ErrorOccurred,fname + ' open error !');

close(f1);

if not ErrorOccurred then
  begin
```

```
      writeln('Application Loading Completed.');

      { Points To PrimaryNode }
      assign(f2,AppendPath(FilePath)+PrimaryNodeName);
      {$I-}
      Reset(f2,1);
      {$I+}
      if (IOResult = 0) then
        begin
          writeln('Loading Primary Node Info ...');
          blockread(f2,node,sizeof(node),NumRead);
          close(f2);
          PresentNodeName := PrimaryNodeName;
        end
      else
      ErrorHandling(ErrorOccurred,fname);

      if not ErrorOccurred then
        begin
          writeln;
          writeln('System ready, wait a moment.......');
          delay(1000);
        end;
    end;

  DummyWindow := EraseTopWindow;

  if not ErrorOccurred then
    begin
      NewApplication;

      { Prompt User Whether Saving The Application Or Not }
      SaveOkay := SaveApplication(Appfname);
    end;

  { Before Leaving This SubProgram, Set flag ANewApplication to True }
  ANewApplication := True;
end;

procedure Browse;
begin
end;


{--------------- End of Main Menu Supporting Routines ----------------------}

(* Main Program *)
begin
  clrscr;
  VideoDetect;
  DetermineWindowAttr;
```

```
if not GetHeapMemOkay(ptr1,ptr2,ptr3) then
  begin
    Sound(1000); delay(100); nosound;
    writeln('*** Not Enough Memory To Run The Program. ***');
    exit;
  end;

if not InitWindowsOkay then
  begin
    Sound(1500); delay(200); nosound;
    writeln('*** Windows Creation Error Due To Insufficient Memeory ***');
    exit;
  end;

if not Voice_Driver_Exist then
  begin
    sound(2000); delay(300); nosound;
    writeln('*** Cannot Find Voice Driver ! Please Load It First ***');
    exit;
  end;

PopAppGenMenu;
SetVoiceDataAddr;
SetSamplingRate(SamplingRate);

if DisplayWindow(NoteBoard) then
  DispNote('VoiceSegAddr = $'+VoiceSeg)
else {nothing};

repeat
  choice := MenuChoice(_M_,ch);
  if ch <> ESC then
    case choice of
      2   : RecordVoice;
      4   : Browse;
      9   : Shell;
      10  : NewApplication;
      11  : GetApplicationFile;
      12  : VoicePlayBack('Memory');
      13  : VoicePlayBack('Disk');
      14  : SetFilePath;
      16  : DefineRecTime;
      17  : DispConfigInfo;
      18  : SaveVoiceMessage;
      19  : SaveOkay := SaveApplication('');
      20  : SelectSamplingRate('S');
      21  : SelectSamplingRate('D');
      22  : SelectSamplingRate('Q');
      23  : DefinePrimaryNodeName;
      24  : DefineActionAfterAnswer;
      25  : DefineGreetingMessage;
      26  : DefineNodePrefix;
```

```
    end;
  until (choice = 8) or (ch = ESC);

  { Before Quit }
  FreeAllocatedHeap;
  repeat
    DummyWindow := EraseTopWindow;
    Dispose(DummyWindow);
  until DummyWindow = nil;
  DisposeMenu(_M_);

  DeleteFile(AppendPath(FilePath) + PrimaryNodeName);
  DeleteFile(AppendPath(FilePath) + NodePrefix + '*');

  clrscr;
end.
```

```
unit M_AppGen;

interface
uses
  TPString,
  TPCrt,
  TPCmd,
  TPWindow,
  TPMenu,
  Dos;

var
  _M_           : Menu;
  Mstack        : MenuStackP;

procedure PopAppGenMenu;


implementation
function Monochrome : Boolean;
var
  regs          : registers;
  disp_mode     : byte;
begin
  { Detect Display Mode }
  regs.ah := 15;
  intr($10,regs);
  disp_mode := regs.al;

  { Select Display Mode }
  if (disp_mode = 2) or (disp_mode = 3) then
    begin
      TextMode(C80);
      Monochrome := False;
    end
  else if (disp_mode = 7) then Monochrome := True;
end;


procedure InitMenu(var M : Menu);
const
  ColorC : MenuColorArray = ($17, $4E, $1B, $7E, $1E, $0E, $19, $78);
  ColorM : MenuColorArray = ($1E, $1E, $03, $78, $1E, $0E, $19, $78);
  Frame1 : FrameArray = '╔╚╝═║';
  Frame2 : FrameArray = '┌└┘─│';

var
  color  : MenuColorArray;
begin
  {Customize this call for special exit characters and custom item displays}
  M := NewMenu([], nil);
```

```
  if Monochrome then
    color := ColorM
  else
    color := ColorC;

  SubMenu(1,2,1,Horizontal,Frame1,Color,'TeleVoice Application Generator');
    MenuItem('FileRetrieve',2,1,1,'Retrieve Application File');
    SubMenu(2,4,1,Vertical,Frame2,Color,'');
      MenuItem(' New Application ',1,2,10,'Create A New Application');
      MenuItem(' Get Application File ',2,2,11,'Retrieve Stored Application');
      PopSublevel;
    MenuItem('Record',16,1,2,'Voice Recording');
    MenuItem('PlayBack',26,1,3,'Voice PlayBack');
    Submenu(25,4,1,Vertical,Frame2,Color,'');
      MenuItem('Voice in Memory',1,10,12,'PlayBack Voice Data in RAM');
      MenuItem('Voice in Disk',2,10,13,'PlayBack Voice Message Stored in Disk');
      PopSublevel;
    MenuItem('Browse ',40,1,4,'Test/View application file');
    MenuItem('Environment',51,1,5,'Set Default path/Directory ');
    SubMenu(45,4,1,Vertical,Frame2,Color,'');
      MenuItem('Set File Path',1,1,14,'Declare File In/Out Path');
      MenuItem('Adjust Voice Sampling Rate',2,8,15,'Select Voice Quality');
      SubMenu(46,7,1,Vertical,Frame2,Color,'');
        MenuItem('2K Byte/Sec',1,1,20,'Acceptable Sampling Rate and The Smallest Voice File Thus Created');
        MenuItem('4K Byte/Sec',2,1,21,'The Preferred Value');
        MenuItem('8K Byte/Sec',3,1,22,'The Best Voice Quality At The Expense of Large Voice File');
        PopSublevel;
      MenuItem('Define Recording Time',3,1,16,'Default Time is 60 Sec/Message');
      MenuItem('Primary Node Name',4,1,23,'Edit Primary Node Name');
      MenuItem('Action After Call Answer',5,1,24,'Define Appropriate Action Right After Call Answer');
      MenuItem('Greeting Message',6,1,25,'Define Greet Message To Be Played Right After Call Answer');
      MenuItem('Node Prefix',7,1,26,'Edit Node File Prefix, Default is $$');
      MenuItem('Display Configuration Info',8,23,17,'Show Configuration Information');
      PopSublevel;
    MenuItem('Save',66,1,6,'Save Works');
    SubMenu(65,4,1,Vertical,Frame2,Color,'');
      MenuItem('Voice in RAM',1,1,18,'Store Current Voice Data in RAM to Disk');
      MenuItem('Application',2,1,19,'Save Entire Application');
      PopSublevel;
    MenuItem('Quit',73,1,7,'Exit The Program');
    SubMenu(72,4,1,Vertical,Frame2,Color,'');
      MenuItem('Exit',1,1,8,'Exit The Program');
      MenuItem('Shell',2,1,9,'Shell To DOS ');
      PopSublevel;
    PopSublevel;

  ResetMenu(M);
end;

procedure PopAppGenMenu;
begin
  InitMenu(_M_);
```

```
end;

end.
```

```
unit datastru;

interface

type
  ConfigStructure = record
    _FilePath        : String     ;
    _SamplingRate    : Char       ;
    _RecTime         : string     ;
    _PrimaryNodeName : string     ;
    _NodePrefix      : string     ;
    _ActionAfterAns  : string     ;
    _GreetMessage    : string     ;
    _SystemNodeNumber : longint   ;
  end;

  Next_N = record
             VoiceFile : string[12];
             NextNode  : longint;          { Points to next node }
             Action    : string[20];       { Specify action to be exec }
           end;

  NodeStructure = record
                    NodeID           : longint;
                    PreviousNode     : longint;
                    LevelIdentifier  : integer;
                    DigitToNode      : array[1..12] of Next_N;
                  end;

implementation

end.
```

```
Unit Colordef;

Interface

const

        {Color constants:
         Black     = 0; Blue   = 1; Green   = 2; Cyan   = 3; Red   = 4;
         Magenta   = 5; Brown  = 6; LtGray  = 7;
         DkGray    = 8; LtBlue = 9; LtGreen = A; LtCyan = B; LtRed = C;
         LtMagenta = D; Yellow = E; White   = F
         }

        {Screen color constants}
        BlackOnBlack       = $00;        BlueOnBlack       = $01;
        BlackOnBlue        = $10;        BlueOnBlue        = $11;
        BlackOnGreen       = $20;        BlueOnGreen       = $21;
        BlackOnCyan        = $30;        BlueOnCyan        = $31;
        BlackOnRed         = $40;        BlueOnRed         = $41;
        BlackOnMagenta     = $50;        BlueOnMagenta     = $51;
        BlackOnBrown       = $60;        BlueOnBrown       = $61;
        BlackOnLtGray      = $70;        BlueOnLtGray      = $71;

        GreenOnBlack       = $02;        CyanOnBlack       = $03;
        GreenOnBlue        = $12;        CyanOnBlue        = $13;
        GreenOnGreen       = $22;        CyanOnGreen       = $23;
        GreenOnCyan        = $32;        CyanOnCyan        = $33;
        GreenOnRed         = $42;        CyanOnRed         = $43;
        GreenOnMagenta     = $52;        CyanOnMagenta     = $53;
        GreenOnBrown       = $62;        CyanOnBrown       = $63;
        GreenOnLtGray      = $72;        CyanOnLtGray      = $73;

        RedOnBlack         = $04;        MagentaOnBlack    = $05;
        RedOnBlue          = $14;        MagentaOnBlue     = $15;
        RedOnGreen         = $24;        MagentaOnGreen    = $25;
        RedOnCyan          = $34;        MagentaOnCyan     = $35;
        RedOnRed           = $44;        MagentaOnRed      = $45;
        RedOnMagenta       = $54;        MagentaOnMagenta  = $55;
        RedOnBrown         = $64;        MagentaOnBrown    = $65;
        RedOnLtGray        = $74;        MagentaOnLtGray   = $75;

        BrownOnBlack       = $06;        LtGrayOnBlack     = $07;
        BrownOnBlue        = $16;        LtGrayOnBlue      = $17;
        BrownOnGreen       = $26;        LtGrayOnGreen     = $27;
        BrownOnCyan        = $36;        LtGrayOnCyan      = $37;
        BrownOnRed         = $46;        LtGrayOnRed       = $47;
        BrownOnMagenta     = $56;        LtGrayOnMagenta   = $57;
        BrownOnBrown       = $66;        LtGrayOnBrown     = $67;
        BrownOnLtGray      = $76;        LtGrayOnLtGray    = $77;

        DkGrayOnBlack      = $08;        LtBlueOnBlack     = $09;
        DkGrayOnBlue       = $18;        LtBlueOnBlue      = $19;
```

```
    DkGrayOnGreen      = $28;        LtBlueOnGreen      = $29;
    DkGrayOnCyan       = $38;        LtBlueOnCyan       = $39;
    DkGrayOnRed        = $48;        LtBlueOnRed        = $49;
    DkGrayOnMagenta    = $58;        LtBlueOnMagenta    = $59;
    DkGrayOnBrown      = $68;        LtBlueOnBrown      = $69;
    DkGrayOnLtGray     = $78;        LtBlueOnLtGray     = $79;

    LtGreenOnBlack     = $0A;        LtCyanOnBlack      = $0B;
    LtGreenOnBlue      = $1A;        LtCyanOnBlue       = $1B;
    LtGreenOnGreen     = $2A;        LtCyanOnGreen      = $2B;
    LtGreenOnCyan      = $3A;        LtCyanOnCyan       = $3B;
    LtGreenOnRed       = $4A;        LtCyanOnRed        = $4B;
    LtGreenOnMagenta   = $5A;        LtCyanOnMagenta    = $5B;
    LtGreenOnBrown     = $6A;        LtCyanOnBrown      = $6B;
    LtGreenOnLtGray    = $7A;        LtCyanOnLtGray     = $7B;

    LtRedOnBlack       = $0C;        LtMagentaOnBlack   = $0D;
    LtRedOnBlue        = $1C;        LtMagentaOnBlue    = $1D;
    LtRedOnGreen       = $2C;        LtMagentaOnGreen   = $2D;
    LtRedOnCyan        = $3C;        LtMagentaOnCyan    = $3D;
    LtRedOnRed         = $4C;        LtMagentaOnRed     = $4D;
    LtRedOnMagenta     = $5C;        LtMagentaOnMagenta = $5D;
    LtRedOnBrown       = $6C;        LtMagentaOnBrown   = $6D;
    LtRedOnLtGray      = $7C;        LtMagentaOnLtGray  = $7D;

    YellowOnBlack      = $0E;        WhiteOnBlack       = $0F;
    YellowOnBlue       = $1E;        WhiteOnBlue        = $1F;
    YellowOnGreen      = $2E;        WhiteOnGreen       = $2F;
    YellowOnCyan       = $3E;        WhiteOnCyan        = $3F;
    YellowOnRed        = $4E;        WhiteOnRed         = $4F;
    YellowOnMagenta    = $5E;        WhiteOnMagenta     = $5F;
    YellowOnBrown      = $6E;        WhiteOnBrown       = $6F;
    YellowOnLtGray     = $7E;        WhiteOnLtGray      = $7F;

Implementation

end.
```

Appendix 3

Parameters of Running the AEM

The AEM program run by all VRUs is the LAppExec.EXE program. The syntax of running it is shown below :-

```
LAppExec [ApplicationFileName] [ [MailDefFile] ]
```

where ApplicationFileName specifies the full path location of the Application if it does not exist in the same directory of AEM. As for the optional parameter MailDefFile, it actually tells the AEM where to load mail box information. The MailDefFile may be generated by dBase IV software and the database structure is as follows

Field Name	Type	Length	Remark
UserID	Char	3	1 - 999
UserName	Char	30	Not used by system
Password	Char	6	Use 0 - 9, * and # only
Directory	Char	50	Specifies where to place mail for this user

After editing, the data base must be exported using Crtl-A (ASC II code 01 or the SOH/☺ symbol) as delimiter. Otherwise, the AEM may not be able to decode the definition file properly.

```
{$M 16384,0,100000}
program LanApplicationExecuteManager;

uses
  Dos,
  TpCrt,
  ToneCard,
  VRPSupp,
  MiscUtil,
  DataStru,
  LanMStru;


const
  ESC = #27;
  EOF = #26;   (* $1A *)

  CallEnded : Boolean = False;
  ByPass    : Boolean = False;

  (* Default Config Values *)
  FilePath       : String    = ''           ;          { current directory }
  SamplingRate   : Char      = 'D'          ;          { 4K bytes per sec }
  RecTime        : string    = '60'         ;          { 60 Seconds }
  PrimaryNodeName : string   = 'P-Node'     ;
  NodePrefix     : string    = '■'          ;
  ActionAfterAns : string    = 'Welcome'    ;
  GreetMessage   : string    = 'Greeting.msg';

  signature      : string    = 'WCSUM'      ;

  ErrorOccurred  : Boolean   = False        ;

  TimeOut        : Boolean   = False        ;
  TimeOutPeriod  : Real      = 10.0 { Sec } ;


  InCorrectTimes : integer   = 0;
  MaxMailBox     = 200;
  FaultAllowed   = 3;
  MaxMailPerUser = 10;


var
  node           : NodeStructure;
  FileRec        : SearchRec;
  key            : char;
  ToneDigitString : string;
  DigitRec       : byte;
  LoadOK         : Boolean;
  LoadPNodeOK    : Boolean;
  MailBox        : array[1..MaxMailBox] of LanMailStructure;
```

```
  MailBoxIndex    : integer;

procedure ErrorHandling(var ErrorFlag : Boolean ; ErrorCode : longint);
var
  ErrorMessage : string;
begin
  if (ErrorCode <= 0) then
    case ErrorCode of
      -8 : ErrorMessage := '** Dos Critical Error Handler Amend Failure';
      -7 : ErrorMessage := '## Not A Valid Mail Definition File';
      -6 : ErrorMessage := '## Mail Definition File Read Error';
      -5 : ErrorMessage := '** Voice Driver Not Found, Please Load It First';
      -4 : ErrorMessage := '** Not Enough Memory to Run the Program';
      -3 : ErrorMessage := '## Primary Node File Read Error';
      -2 : ErrorMessage := '## Application File Open Error';
      -1 : ErrorMessage := '## Not A Valid Application File';
       0 : ErrorMessage := '## Primary Node File Handling Error';
    end
  else
    ErrorMessage := '## Error in handling file of Node No. '+
                    IntToStr(ErrorCode);
  writeln;
  writeln(ErrorMessage);
  writeln;
  writeln('*** Hit Any Key ***');
  sound(3000); delay(1000); nosound;
  waitkey;
  ErrorFlag := True;
end;


function LoadNodeFromDisk(ID : longint) : Boolean;
var
  f         : file;
  fname     : string;
  ResultRead : word;
begin
  if (ID = 0) then
    fname := AppendPath(FilePath) + PrimaryNodeName
  else
    fname := AppendPath(FilePath) + NodePrefix + IntToStr(ID);
  assign(f,fname);
  {$I-}
  reset(f,1);
  {$I+}
  if (IOResult = 0) then
    begin
      BlockRead(f,node,sizeof(node),ResultRead);
      close(f);
      LoadNodeFromDisk := True;
    end
  else
```

```
    LoadNodeFromDisk := False;
end;

procedure GetApplicationFile (fname : string);
var
  f1,f2           : file;
  sign            : string;
  NumRead         : word;
  NumWritten      : word;
  Config          : ConfigStructure;
  I               : longint;
  TestString      : string;
  SystemNodeNumber : longint;

begin
  if (fname = '') then
   exit;

  writeln('Loading Application File '+fname);
  assign(f1,fname);
  {$I-}
  reset(f1,1);
  {$I+}
  if (IOResult = 0) and not ErrorOccurred then begin
    writeln('Verifying File Signature ...');
    blockread(f1,sign,sizeof(signature),NumRead);
    TestString := Signature + EOF;
    if (sign = TestString) and not ErrorOccurred then begin
      writeln('Retrieving Configuration from file ...');
      blockread(f1,Config,sizeof(Config),NumRead);
      with Config do
        begin
          FilePath            := _FilePath;
          SamplingRate        := _SamplingRate;
          RecTime             := _RecTime;
          PrimaryNodeName     := _PrimaryNodeName;
          NodePrefix          := _NodePrefix;
          ActionAfterAns      := _ActionAfterAns;
          GreetMessage        := _GreetMessage;
          SystemNodeNumber    := _SystemNodeNumber;
        end;
      blockread(f1,node,sizeof(node),NumRead);
      assign(f2,AppendPath(FilePath)+PrimaryNodeName);
      {$I-}
      Rewrite(f2,1);
      {$I+}
      if (IOResult = 0) and not ErrorOccurred then
        begin
          writeln('Creating Primary Node File ...');
          blockwrite(f2,node,sizeof(node),NumWritten);
          close(f2);
        end
```

```
    else (* Primary Node File Create Error *)
      ErrorHandling(ErrorOccurred,0);

    for I := 1 to SystemNodeNumber do
      begin
        assign(f2,AppendPath(FilePath) + NodePrefix + IntToStr(I));
        {$I-}
        rewrite(f2,1);
        {$I+}
        if (IOResult = 0) and not ErrorOccurred then
          begin
            writeln('Creating Node '+ IntToStr(I) + ' File ...');
            blockread(f1,node,sizeof(node),NumRead);
            blockwrite(f2,node,sizeof(node),NumWritten);
            close(f2);
          end
        else  (* Node #I File Create Error    *)
          ErrorHandling(ErrorOccurred,I);
      end;
  end
  else
    ErrorHandling(ErrorOccurred,-1);     (* Not A Valid Application File *)
end
else
  ErrorHandling(ErrorOccurred,-2);       (* Application File Open Error  *)

{$I-} close(f1); {$I+}

if not ErrorOccurred and (IOResult = 0) then
  begin
    writeln('Application Loading Completed.');

    { Points To PrimaryNode }
    assign(f2,AppendPath(FilePath)+PrimaryNodeName);
    {$I-}
    Reset(f2,1);
    {$I+}
    if (IOResult = 0) then
      begin
        writeln('Loading Primary Node Info ...');
        blockread(f2,node,sizeof(node),NumRead);
        close(f2);
      end
    else
    ErrorHandling(ErrorOccurred,-3);

    writeln;
    writeln('System ready, wait a moment.......');
    delay(1000);
  end;
end;
```

```
procedure LoadMailDef(filename : string);
const
  Delimiter  = #1;
  LF         = #10;
  CR         = #13;

var
  f          : file of Char;
  ch         : char;
  BeginPos   : longint;
  EndPos     : longint;
  InfoStart  : longint;
  InfoEnd    : longint;
  InfoStr    : String;
  ValidID    : Boolean;
  ID,code    : integer;
  PassFile   : file;
  TempPass   : string;
  NumWritten : word;

  procedure FindDelimiterPosition(SearchStartPos : longint ;
            var _start,_end : longint);
  var
    _ch : char;

  begin
    seek(f,SearchStartPos);
    repeat
      read(f,_ch);
    until (_ch = Delimiter);
    _start := FilePos(f);

    repeat
      read(f,_ch);
    until (_ch = Delimiter);
    _end := FilePos(f);
  end;

  function InfoBetwDelimiter(beginpoint,endpoint : longint) : string;
  var
    template : string;
    _ch      : char;
    I        : integer;
  begin
    if ((endpoint - beginpoint) <= 1) then
      begin
        InfoBetwDelimiter := '';
        exit;
      end;

    FillChar(template,sizeof(template),' ');
    seek(f,beginpoint);
```

```
    I := 1;
    repeat
      read(f,_ch);
      template[I] := _ch;
      Inc(I);
    until (FilePos(f) = endpoint-1);
    InfoBetwDelimiter := copy(template,1,I-1);
  end;

begin
  clrscr;
  writeln('Reading Mail Definition File ...');
  assign(f,filename);
  {$I-} reset(f); {$I+}
  if (IOResult <> 0) then
    begin
      ErrorHandling(ErrorOccurred,-6);
      exit;
    end;
  read(f,ch);
  if (ch <> Delimiter) then
    begin
      ErrorHandling(ErrorOccurred,-7);
      {$I-} Close(f); {$I+}
      exit;
    end;

  { reset file pointer }
  seek(f,0);
  writeln('Reading Mail Info ... ');
  repeat
    read(f,ch);
    if (ch <> EOF) then
      begin
        BeginPos := FilePos(f) - 1;
        repeat read(f,ch); until (ch = CR);
        EndPos := FilePos(f) - 1;
        { Extract Information }
        (* UserID *)
        FindDelimiterPosition(BeginPos,InfoStart,InfoEnd);
        InfoStr := InfoBetwDelimiter(InfoStart,InfoEnd);

        ValidID := False;
        if (ValidNumberString(InfoStr)) then
          begin
            Val(InfoStr,ID,code);
            if (ID <= MaxMailBox) then
              ValidID := True;
          end;

        if ValidID then begin
          (* UserName *)
```

```
        FindDelimiterPosition(InfoEnd,InfoStart,InfoEnd);
        InfoStr := InfoBetwDelimiter(InfoStart,InfoEnd);
        MailBox[ID].UserName := InfoStr;
        writeln('Loading '+ InfoStr + ' Data ...');

        (* PassWord *)
        FindDelimiterPosition(InfoEnd,InfoStart,InfoEnd);
        InfoStr := InfoBetwDelimiter(InfoStart,InfoEnd);
        MailBox[ID].Password := InfoStr;

        (* Directory *)
        FindDelimiterPosition(InfoEnd,InfoStart,InfoEnd);
        InfoStr := InfoBetwDelimiter(InfoStart,InfoEnd);
        MailBox[ID].Directory := InfoStr;

        (* Save PassWord to disk in unprotected form *)
        TempPass := MailBox[ID].Password;
        assign(PassFile,AppendPath(MailBox[ID].Directory)+'Password.Fil');
        {$I-} rewrite(Passfile,1); {$I+}
        if (IOResult <> 0) then
          begin
            writeln;
            writeln(MailBox[ID].UserName + ' Data Load Error !!!');
            writeln;
            sound(1000); delay(500); nosound;
            writeln('Press <Esc> to quit or any other key to continue ...');
            repeat until keypressed;
            ch := readkey;
            if (ch = #27) then
              begin
                close(f);
                halt(1);
              end;
          end
        else
          begin
            blockwrite(PassFile,TempPass,sizeof(TempPass),NumWritten);
            close(PassFile);
          end;
      end;

      { Dummy Read CR and LF pair }
      Seek(f,EndPos);
      read(f,ch);
      read(f,ch);
    end;
  until (ch = EOF);

  delay(1000);
  clrscr;
  {$I-} close(f); {$I+}
end;
```

```
procedure AppExecInit;
begin
  { ToneCard }
  ToneCardInit;
  SetPlayMode;
  DisableToneRec;
  DigitAvailable := False;

  { VRP-70 Voice Card }
  if not Voice_Driver_Exist then
    begin
      ErrorHandling(ErrorOccurred,-5);
      exit;
    end;

  if not GetHeapMemOkay(ptr1,ptr2,ptr3) then
    begin
      ErrorHandling(ErrorOccurred,-4);
      exit;
    end;

  SetVoiceDataAddr;
  SetSamplingRate(SamplingRate);

  { Load MailDef File }
  if (ParamCount >= 2) then
    LoadMailDef(ParamStr(2));
  if ErrorOccurred then exit;

  { Load Application File }
  GetApplicationFile(ParamStr(1));

end;


procedure AppExecByebye;
begin
  ToneCardByebye;
  FreeAllocatedHeap;
end;


(*****************************************************************************)
procedure ClearUnWantedToneDigit;
var
  hr,min,sec,sec100  : word;
  TimeElapsed        : real;
begin
  EnableToneRec;
  GetTime(hr,min,sec,sec100);
  repeat
```

```
    while DigitAvailable do DigitAvailable := False;
  until (StopWatch(hr,min,sec,sec100) >= 0.5);
  DisableToneRec;
end;

procedure VoicePlay(fname : string);
begin
  if LoadVoiceOkay(fname) then
    VRP_Play
  else
    writeln('Voice File : ' + fname + ' Not Found, Play Denied. ');
end;

procedure WaitTone(NumOfDigit : byte ; var ToneDigitString : string);
var
  I                : integer;
  hr,min,sec,sec100  : word;
  TimeElapsed      : real;

begin
  FillChar(ToneDigitString,sizeof(ToneDigitString),'X');
  EnableToneRec;
  GetTime(hr,min,sec,sec100);
  for I := 1 to NumOfDigit do
    begin
      repeat
        TimeElapsed := StopWatch(hr,min,sec,sec100);
        if (TimeElapsed >= TimeOutPeriod) then
          begin
            TimeOut := True;
            I := NumOfDigit;
          end;
      until DigitAvailable or TimeOut;
      DigitAvailable := False;
      ToneDigitString[I] := ToneDigit(Digit);
    end;
  DisableToneRec;
  ToneDigitString := copy(ToneDigitString,1,NumOfDigit);
end;

procedure GoBackward;
var
  NodeID  : longint;
begin
  if (node.PreviousNode = -1) then
    NodeID := 0
  else
    NodeID := Node.PreviousNode;

  if LoadNodeFromDisk(NodeID) then
    ByPass := True
  else
```

```
    writeln('Node '+ IntToStr(node.PreviousNode) + ' Load Failure');
end;

procedure EndCall;
begin
  CallEnded := True;
end;

procedure StayAtThisNode;
begin
  ByPass := True;
end;


function GetDigitString(EndString : string) : string;

(*
   Note carefully that if EndString is null, then returned string is the
   1st digit pressed ==> same as the WaitTone(1,DummyString)
*)

var
  I                : integer;
  Buff,Comp        : string;
  matched          : Boolean;
  hr,min,sec,sec100  : word;
  NoResponse       : Boolean;

begin
  matched := False;
  Buff := '';
  EnableToneRec;
  repeat
    GetTime(hr,min,sec,sec100);
    repeat
      NoResponse := TimeUp(TimeOutPeriod,hr,min,sec,sec100);
    until DigitAvailable or NoResponse;
    if DigitAvailable then
      begin
        Buff := Buff + ToneDigit(Digit);
        if (length(Buff) >= length(EndString)) then
          begin
            Comp := copy(Buff,length(Buff)-length(EndString)+1,length(EndString));
            if (Comp = EndString) then
              matched := True;
          end;
      end;
    DigitAvailable := False;
  until matched or (length(Buff) = sizeof(Buff)) or NoResponse;
  DisableToneRec;

  if matched then
```

```
   Buff := Copy(Buff,1,length(Buff)-length(EndString));
  GetDigitString := Buff;
end;

function TimeFileName : string;
const
  MonthString : array[1..12] of string[2] = ('JA','FE','MA','AP',
                'MY','JE','JY','AU','SE','OC','NV','DE');

var
  Year,Month,Day,DayOfWeek : word;
  hr,min,sec,sec100        : word;
  DayStr,MonStr            : string;
  HrStr,MinStr,SecStr      : string;

  procedure AdjustDayTimeString (var Num : word ; var NumStr : string);
  begin
    NumStr := IntToStr(longint(Num));
    if (Num <= 9) then NumStr := '0' + NumStr;
  end;

begin
  GetDate(Year,Month,Day,DayOfWeek);
  GetTime(hr,min,sec,sec100);
  AdjustDayTimeString(Day,DayStr);
  MonStr := MonthString[Month];
  AdjustDayTimeString(hr,HrStr);
  AdjustDayTimeString(min,MinStr);
  AdjustDayTimeString(sec,SecStr);
  if (length(SecStr) = 1) then
    SecStr := '00' + SecStr
  else if (length(SecStr) = 2) then
    SecStr := '0' + SecStr;
  TimeFileName := DayStr + MonStr + HrStr + MinStr + '.' + SecStr;
end;

procedure Retry;
begin
  Inc(InCorrectTimes);
  if (InCorrectTimes >= FaultAllowed) then
    begin
      DigitAvailable := False;
      VoicePlay(AppendPath(FilePath) + 'ForceEnd.msg');
      writeln('Too many wrong keys, Call is forced to terminate ... ');
      CallEnded := True;
    end;
  StayAtThisNode;
end;

procedure RetryAndClearFaults;
begin
  InCorrectTimes := 0;
```

```
  StayAtThisNode;
end;

procedure FindMailID;
begin
  MailBoxIndex := integer(DigitRec);
end;

function RetrievePassWordFromFile(FromFile : string) : string;
var
  PassFile  : file;
  NumRead   : word;
  TempPass  : string;
begin
  assign(PassFile,FromFile);
  {$I-} reset(Passfile,1); {$I+}
  if (IOResult <> 0) then
    RetrievePassWordFromFile := '@'
  else
    begin
      blockread(PassFile,TempPass,sizeof(TempPass),NumRead);
      close(PassFile);
      RetrievePassWordFromFile := TempPass;
    end;
end;

function PassWordOkay (PassWord,PassEndString : string) : Boolean;
var
  pass   : string;
begin
  pass := GetDigitString(PassEndString);
  if (pass = PassWord) then
    PassWordOkay := True
  else
    PassWordOkay := False;
end;

function AskDeleteJustListenVoiceMail : Boolean;
var
  ToneKey : string;
begin
  ToneKey := GetDigitString('');
  if (ToneKey[1] = '*') then
    AskDeleteJustListenVoiceMail := True
  else
    AskDeleteJustListenVoiceMail := False;
end;

procedure ForceEndOfCurrentCall;
begin
  ClearUnWantedToneDigit;
  VoicePlay(AppendPath(FilePath) + 'ForceEnd.msg');
```

```
  EndCall;
end;

procedure ForceEndDueToSystemFailure;
begin
  ClearUnWantedToneDigit;
  VoicePlay(AppendPath(FilePath) + 'SysDown.msg');
  EndCall;
end;

procedure TeleVoiceMail;
var
  fname : string;
begin
  (* Check for mail full *)
  if (NoOfFiles(AppendPath(MailBox[MailBoxIndex].Directory) + '*.*',
                AnyFile,
                FileRec) >= MaxMailPerUser+1) then
    begin
      ClearUnWantedToneDigit;
      VoicePlay(AppendPath(FilePath) + 'MailFull.msg');
      VoicePlay(AppendPath(FilePath) + 'ThankUse.msg');
      EndCall;
      exit;
    end;

  (* Okay, allow to leave VMail *)
  SetRecordMode;
  ClearUnWantedToneDigit;
  VRP_Record(RecTime);
  SetPlayMode;
  fname := AppendPath(MailBox[MailBoxIndex].Directory) + TimeFileName;
  VRP_SaveVoiceMessage(fname);
  delay(200);
  VoicePlay(AppendPath(FilePath) + 'ThankUse.msg');
  EndCall;
end;

procedure ListenToVoiceMail(PassEnd,EndVoice : string);
var
  FileNameArray   : array[1..10] of string;
  DeleteVoiceMail : Boolean;
  Index           : integer;
  Pass            : string;

begin
  delay(200);
  Pass := RetrievePassWordFromFile(AppendPath(MailBox[MailBoxIndex].Directory)
          + 'Password.Fil');

  if (Pass = '@') then
    begin
```

```
      ForceEndDueToSystemFailure;
      exit;
    end;

  if not PassWordOkay(Pass,PassEnd) then
    begin
      ForceEndOfCurrentCall;
      exit;
    end;

  for index := 1 to 10 do
    FileNameArray[index] := '';

  FindFirst(AppendPath(MailBox[MailBoxIndex].Directory) + '*.*',
            AnyFile,
            FileRec);

  index := 0;
  while (DosError = 0) do begin
    if (FileRec.Name <> '.') and (FileRec.Name <> '..') and
       (ToUpperCase(FileRec.Name) <> 'PASSWORD.FIL') then
      begin
        Inc(index);
        FileNameArray[index] := AppendPath(MailBox[MailBoxIndex].Directory)
                                + FileRec.Name;
      end;
    FindNext(FileRec);
  end;

  for index := 1 to 10 do
    if (FileNameArray[index] <> '') then
      begin
        ClearUnWantedToneDigit;
        VoicePlay(FileNameArray[index]);

        (* Prompt Mail Box Owner to delete the mail *)
        VoicePlay(AppendPath(FilePath) + 'AskDelVM.msg');
        DeleteVoiceMail := AskDeleteJustListenVoiceMail;
        if DeleteVoiceMail then
           if (ToneDigit(Digit) = '*' ) then begin
             SwapVectors;
             Exec(GetEnv('COMSPEC'),'/C del '+FileNameArray[index]+' > NUL');
             SwapVectors;
           end;
      end;

  ClearUnWantedToneDigit;
  VoicePlay(EndVoice);
  StayAtThisNode;
end;

procedure ChangePassWord(PromptPassWordMessage,PassEndString : string);
```

```
var
  NewPassWord1   : string;
  NewPassWord2   : string;
  Pass           : string;
  PassFile       : file;
  NumWritten     : word;
begin
  delay(200);
  Pass := RetrievePassWordFromFile(AppendPath(MailBox[MailBoxIndex].Directory)
          + 'Password.Fil');

  if (Pass = '@') then
    begin
      ForceEndDueToSystemFailure;
      exit;
    end;

  if PassWordOkay(Pass,PassEndString) then
    begin
      VoicePlay(PromptPassWordMessage);
      NewPassWord1 := GetDigitString(PassEndString);
      VoicePlay(PromptPassWordMessage);
      NewPassWord2 := GetDigitString(PassEndString);
      if (NewPassWord1 = NewPassWord2) then
        begin
          assign(PassFile,AppendPath(MailBox[MailBoxIndex].Directory)+'Password.Fil');
          {$I-} Rewrite(PassFile,1); {$I-}
          ClearUnWantedToneDigit;
          if (IOResult <> 0) then
            begin
              VoicePlay(AppendPath(FilePath) + 'SysDown.msg');
              ForceEndDueToSystemFailure;
              exit;
            end
          else
            begin
              blockwrite(PassFile,NewPassWord1,sizeof(NewPassWord1),NumWritten);
              close(PassFile);
              VoicePlay(AppendPath(FilePath) + 'Success.msg');
              StayAtThisNode;
              exit;
            end;
        end;
    end;

  ForceEndOfCurrentCall;
end;

procedure ChangeLatestMessage(PromptPassWordMessage,
                              PassEndString,
                              StoreToFile            : string);
var
```

```
  Pass  : string;

begin
  delay(200);
  Pass := RetrievePassWordFromFile(AppendPath(MailBox[MailBoxIndex].Directory)
          + 'Password.Fil');

  if (Pass = '@') then
    begin
      ForceEndDueToSystemFailure;
      exit;
    end;

  if PassWordOkay(Pass,PassEndString) then
    begin
      VoicePlay(AppendPath(FilePath) + 'ChLatIns.msg');
      SetRecordMode;
      ClearUnWantedToneDigit;
      VRP_Record(RecTime);
      SetPlayMode;
      VRP_SaveVoiceMessage (StoreToFile);

      ClearUnWantedToneDigit;
      VoicePlay(StoreToFile);
      StayAtThisNode;
      exit;
    end;

  ForceEndOfCurrentCall;
end;

(* ----------  Demo2 procedures  ------------- *)
procedure MCheungChangePassWord;
begin
  ChangePassWord(AppendPath(FilePath) + 'MCGetP.msg','***');
end;

procedure MCheungListenToVoiceMail;
begin
  ListenToVoiceMail('***',AppendPath(FilePath) + 'MCLatest.msg');
end;

procedure MCheungChangeLatestMessage;
begin
  ChangeLatestMessage(AppendPath(FilePath) + 'MCGetP.msg',
                      '***',
                      AppendPath(FilePath) + node.DigitToNode[1].VoiceFile);
end;

procedure PYumChangePassWord;
begin
  ChangePassWord(AppendPath(FilePath) + 'PYumGetP.msg','###');
```

```
end;

procedure PYumListenToVoiceMail;
begin
  ListenToVoiceMail('###',AppendPath(FilePath) + 'PYLatest.msg');
end;

procedure PYumChangeLatestMessage;
begin
  ChangeLatestMessage(AppendPath(FilePath) + 'PYumGetP.msg',
                      '###',
                      AppendPath(FilePath) + node.DigitToNode[1].VoiceFile);
end;

procedure DeptChangePassWord;
begin
  ChangePassWord(AppendPath(FilePath) + 'DeptGetP.msg','###');
end;

procedure DeptListenToVoiceMail;
begin
  ListenToVoiceMail('###',AppendPath(FilePath) + 'DpLatest.msg');
end;

procedure DeptChangeLatestMessage;
begin
  ChangeLatestMessage(AppendPath(FilePath) + 'DeptGetP.msg',
                      '###',
                      AppendPath(FilePath) + node.DigitToNode[1].VoiceFile);
end;

(* -------- End of Demo2 Procedures ----------- *)

procedure ExecuteAction(ActionString : string);
begin
  ActionString := ToUpperCase(ActionString);

  if ActionString = 'GOBACKWARD' then
    GoBackWard;
  if ActionString = 'END' then
    EndCall;
  if ActionString = 'FINDMAILID' then
    FindMailID;
  if ActionString = 'RETRY' then
    Retry;
  if ActionString = 'RETRY&CF' then
    RetryAndClearFaults;
  if ActionString = 'STAY' then
    StayAtThisNode;
  if ActionString = 'VMRECORD' then
    TeleVoiceMail;
```

```
  if ActionString = 'MCCHPASS' then
    MCheungChangePassWord;
  if ActionString = 'MCLISTEN' then
    MCheungListenToVoiceMail;
  if ActionString = 'MCCHLATEST' then
    MCheungChangeLatestMessage;

  if ActionString = 'PYCHPASS' then
    PYumChangePassWord;
  if ActionString = 'PYLISTEN' then
    PYumListenToVoiceMail;
  if ActionString = 'PYCHLATEST' then
    PYumChangeLatestMessage;

  if ActionString = 'DPCHPASS' then
    DeptChangePassWord;
  if ActionString = 'DPLISTEN' then
    DeptListenToVoiceMail;
  if ActionString = 'DPCHLATEST' then
    DeptChangeLatestMessage;

end;


(*****************************************************************************)

begin
  writeln; writeln;
  writeln('***  LAN Version of Application Execute Manager  ***');
  writeln; writeln;
  PseudoMusic;
  delay (1000);

  if (ParamCount <= 0) then begin
    writeln('Syntax : Appexec [ApplicationFileName] [ [MailDefFile] ]');
    exit;
  end;

  clrscr;
  AppExecInit;
  if ErrorOccurred then exit;

  clrscr;
  key := #0;
  repeat
    writeln;

    writeln('[ Hit <ESC> to quit ] ');
    writeln('Waiting for an incoming call ... ');
    while MachineIdle and (key <> ESC) do
      begin
        if keypressed then
```

```
      key := readkey;
    if (RingCount <> 0) then
      begin
        write('A call comes !!! ');
        write(' RingCount = ',RingCount);
        gotoxy(1,wherey);
      end;
  end;

{ 0.8 second delay for switch to accept the call &
  clear all unwanted tone }
delay (300);
ClearUnwantedToneDigit;  (* this routine will introduce 0.5 sec delay *)

writeln;
LoadPNodeOK := LoadNodeFromDisk(0);
if (key <> ESC) and LoadPNodeOK then begin
  VoicePlay(AppendPath(FilePath) + GreetMessage);
  ExecuteAction(ActionAfterAns);
  repeat
    writeln('Now in node '+IntToStr(node.NodeID));
    WaitTone(1,ToneDigitString);
    if not TimeOut then
      begin
        DigitRec := ToneDigitToNumeric(ToneDigitString[1]);
        VoicePlay(AppendPath(FilePath) + node.DigitToNode[DigitRec].VoiceFile);
        ExecuteAction(node.DigitToNode[DigitRec].Action);
        if not ByPass then
          if (node.DigitToNode[DigitRec].NextNode = -1) then
            CallEnded := True
          else begin
            (* Avoid disk contention with resident program *)
            delay(600);
            LoadOK := LoadNodeFromDisk(node.DigitToNode[DigitRec].NextNode);
            if not LoadOK then
              begin
                CallEnded := True;
                DigitAvailable := False;
                VoicePlay(AppendPath(FilePath) + 'SysDown.Msg');
              end;
          end;
        ByPass := False;
      end;
  until CallEnded or TimeOut;
end;

if not LoadPNodeOk then
  begin
    ClearUnWantedToneDigit;
    VoicePlay(AppendPath(FilePath) + 'SysDown.msg');
    writeln('*** Primary Node Load failed !!! ***');
  end;
```

```
    if TimeOut then
      begin
        DigitAvailable := False;
        VoicePlay(AppendPath(FilePath) + 'TimeOut.Msg');
        writeln('WaitTone time out, Call is to be disconnected ... ');
      end;

    { Test if user wants to exit }
    if keypressed then
      key := readkey;

    { Reinit variables }
    MachineIdle := True;
    RingCount := 0;
    OpenHookFlashRelay;
    ClearRingFF;
    CallEnded := False;
    ByPass := False;
    TimeOut := False;
    DigitAvailable := False;
    InCorrectTimes := 0;

  until (key = ESC);

  { Before quit }
  AppExecByebye;
end.
```

```
unit ToneCard;

interface
uses
  Dos,TpCrt;

const
  MachineIdle : Boolean = TRUE; { Machine Idle ==> awaiting for I/C call }
  RingCount   : integer = 0;
  MaxRingCount = 10;

var
  DigitAvailable   : Boolean;
  Digit            : byte;

procedure DisableInterrupts;
procedure EnableInterrupts;
procedure EnableToneRec;
procedure DisableToneRec;
procedure OpenHookFlashRelay;
procedure CloseHookFlashRelay;
procedure ClearRingFF;
procedure SetPlayMode;
procedure SetRecordMode;
procedure ToneCardByebye;
procedure ToneCardInit;

function ToneDigit(input : byte) : char;


implementation
const
  base_address      = $270;

  Vector            = $0B;      { IRQ 3 }
  INTFB             = $02;

  RecordOrPlay      = $03;      { Pin 3 of Port C of 8255 }

  ClrFF             = $04;      { Pin 4 of Port C of 8255 }
                                { Clear D Flip Flop of Ringing Circuitry }

  HookFlashRelay    = $05;      { Pin 5 of Port C of 8255 }

var
  PORTA,PORTB,PORTC,ControlPort : integer;
  OldVector                     : pointer;


procedure PIA_Init (mode : byte);
begin
  PORTA := base_address;
```

```
  PORTB := PORTA + 1;
  PORTC := PORTB + 1;
  ControlPort := PORTC + 1;
  port[ControlPort] := mode;
end;


procedure DisableInterrupts;
begin
  inline($FA);
end;


procedure EnableInterrupts;
begin
  inline($FB);
end;

procedure EnableToneRec;
begin
  port[ControlPort] := (INTFB shl 1) + 1;
end;

procedure DisableToneRec;
begin
  port[ControlPort] := (INTFB shl 1);
end;

procedure OpenHookFlashRelay;
begin
  port[ControlPort] := (HookFlashRelay shl 1) + 1;
end;

procedure CloseHookFlashRelay;
begin
  port[ControlPort] := (HookFlashRelay shl 1);
end;

procedure ClearRingFF;
begin
  port[ControlPort] := (ClrFF shl 1);
  delay(100);
  port[ControlPort] := (ClrFF shl 1) + 1;
end;

procedure SetPlayMode;
begin
  port[ControlPort] := (RecordOrPlay shl 1) + 1;
end;

procedure SetRecordMode;
begin
```

```
  port[ControlPort] := (RecordOrPlay shl 1);
end;

procedure ToneRecHandler(Flags,CS,IP,AX,BX,CX,DX,SI,DI,DS,ES,BP : word);
interrupt;
begin
  DisableInterrupts;

  if MachineIdle then begin
    RingCount := RingCount + 1;
    ClearRingFF;
    if (RingCount > MaxRingCount) then begin
      CloseHookFlashRelay;
      MachineIdle := False;
      RingCount := 0;
    end;
  end
  else begin
    Digit := port[PortB] and $0F;
    DigitAvailable := True;
  end;

  EnableInterrupts;

  port[$20] := $20;
end;


procedure ToneCardByebye;
begin
  DisableToneRec;
  SetIntVec(Vector,OldVector);
  dispose(OldVector);
  OldVector := nil;

  { Mask corresponding Interrupt Enable bit of 8259 }
  port[$21] := port[$21] or ($01 shl (Vector - 8));

  OpenHookFlashRelay;
  RingCount := 0;
end;

procedure ToneCardInit;
begin
  EnableInterrupts;

  { Init 8255 }
  PIA_Init($86);              (* Mode 1 with strobed Port B input *)

  { Clear Ring FF and Enable Ring }
  port[ControlPort] := (ClrFF shl 1 ) + 1;
```

```
  { Initialize Interrupt Vetcor }
  new(OldVector);
  GetIntVec(Vector,OldVector);
  SetIntVec(Vector,@ToneRecHandler);

  { Init 8259 to receive Tone arrival interrupt }
  port[$21] := port[$21] and (not ($01 shl (Vector -8)));

  OpenHookFlashRelay;
end;

function ToneDigit(input : byte) : char;
var
  buffer : byte;
begin
  case input of
    1..9        :  buffer := input + byte('0');
    10          :  buffer := byte('0');
    11          :  buffer := byte('*');
    12          :  buffer := byte('#');
    13          :  buffer := byte('A');
    14          :  buffer := byte('B');
    15          :  buffer := byte('C');
    0           :  buffer := byte('D');
  end;
  ToneDigit := char(buffer);
end;

end.
```

```
unit VRPSupp;

interface
uses TpCrt,ToneCard;

var
  ptr1,ptr2,ptr3       : pointer;

procedure FreeAllocatedHeap;
procedure SetVoiceDataAddr;
procedure SetSamplingRate(mode : char);
procedure VoiceStop;
procedure VRP_Play;
procedure VRP_Record (RecordTime : String);
procedure VRP_SaveVoiceMessage (ToFile : string);

function LoadVoiceOkay (fname : string)          : Boolean;
function Voice_Driver_Exist                      : Boolean;
function GetHeapMemOkay (var p1,p2,p3 : pointer) : Boolean;

implementation
type
  anystring = string[80];


const
  _32K      = 32*1024;

  VoiceSeg  : string[4] = '****';
  VoiceOfs  : string[4] = '****';
  HexDigit  : string[4] = '****';


var
  segment   : word;


procedure DisableInterrupts;
begin
  inline($FA);
end;

procedure EnableInterrupts;
begin
  inline($FB);
end;

function power(value,index : integer) : word;  { index should +ve }
var
  i      : integer;
  result : word;
begin
```

```
  result := 1;
  for i := 1 to index do
    begin
      result := result*value;
    end;
  power := result;
end;

procedure DecToHex(decimal : word);
var
  addr    : string[4];
  rem     : word;
  i,buff  : integer;
begin
  rem := decimal;
  addr := '****';
  for i := 1 to 4 do
    begin
      buff := trunc(rem/power(16,4-i));
      if (buff > 9) then addr[i] := char(integer('A')+buff-10)
      else addr[i] := char(integer('0')+buff);
      rem := rem - buff*power(16,4-i);
    end;
  HexDigit := copy(addr,1,length(addr));
end;
(* Call to DecToHex causes result to be stored in hexdigit
   which is of type string[4] *)

function HexToDec(hex : string) : word;
var
  result     : word;
  i,buff     : integer;
begin
  result := 0;
  for i := 1 to 4 do
    begin
      if hex[i] in ['A'..'F'] then
        buff := integer(hex[i]) - integer('A') + 10
      else
        buff := integer(hex[i]) - integer('0');

      result := result + buff*power(16,4-i);
    end;
  HexToDec := result;
end;


(*****************************************************************************)
(*                    VRP-70 Voice Card Routines                             *)
(*****************************************************************************)

procedure CMDEXEC(s : anystring);
```

```
var
  ch            : byte;
  addr,i        : integer;
begin
  addr:=memw[0000:$182];       { get int$60 segment            }
  for i:=1 to ord(s[0]) do     { s[0] is the length of string s }
    mem[addr:i+15]:=ord(s[i]);

  { save AX, BX and ES into stack before executing the command }
  inline($06/
         $50/
         $53/
         $cd/
         $60/
         $5B/
         $58/
         $07) ;
end ;


function Voice_Driver_Exist : Boolean;
begin
  if (memw[0000:$180]<>0) and (memw[0000:$182]<>0) then {vrp_driver test}
    begin
      segment := memw[0000:$182] ;
      CMDEXEC('T');
      if mem[segment:16] = $31 then Voice_Driver_Exist := True
      else Voice_Driver_Exist := False;
    end
  else Voice_Driver_Exist := False;
end;

function GetHeapMemOkay(var p1,p2,p3 : pointer) : Boolean;
begin
  if (MaxAvail > 98304) then       { 96K block }
    begin
      GetMem(p1,_32K);
      GetMem(p2,_32K);
      GetMem(p3,_32K);
      GetHeapMemOkay := True;
    end
  else
    GetHeapMemOkay := False;
end;
(*
   1st byte of allocated heap is at
     memw[seg(ptr1):ofs(ptr1)+2]:memw[seg(ptr1):ofs(ptr1)]
   or simply
     seg(ptr1^) : ofs(ptr1^)
*)

procedure FreeAllocatedHeap;
```

```
begin
  FreeMem(ptr1,_32K);
  FreeMem(ptr2,_32K);
  FreeMem(ptr3,_32K);
end;

procedure SetVoiceDataAddr;

var
  i                    : integer;
  DecMemSeg,DecMemOfs  : word;

begin
  DecMemOfs := ofs(ptr1^);
  DecMemSeg := seg(ptr1^);

  if (DecMemOfs <> 0) then
    DecMemSeg := (((DecMemSeg shl 4) + DecMemOfs) shr 4 ) + 1;

  DecToHex(DecMemSeg);
  VoiceSeg := copy(HexDigit,1,length(HexDigit));
  VoiceOfs := '0000';

  if ((VoiceSeg[1]+'000') <= VoiceSeg)
    and (VoiceSeg <= (VoiceSeg[1]+'7FF')) then
    VoiceSeg := VoiceSeg[1]+'800'
  else VoiceSeg := char(byte(VoiceSeg[1])+1)+'000';

  CMDEXEC('A'+VoiceSeg);
end;

procedure SetSamplingRate(mode : char);
begin
  mode := UpCase(mode);
  CMDEXEC('M'+mode);
end;

procedure VoiceStop;
begin
  DisableInterrupts;
  CMDEXEC('E');
  EnableInterrupts;
end;

function LoadVoiceOkay(fname : string) : Boolean;
begin
  DisableInterrupts;
  delay(300);
  CMDEXEC('L'+fname);
  if (mem[segment:16] = 32) then
    LoadVoiceOkay := True
  else
```

```
    LoadVoiceOkay := False;
  EnableInterrupts;
end;

procedure VRP_Play;
begin
  DisableInterrupts;
  CMDEXEC('P');
  EnableInterrupts;
  EnableToneRec;
  repeat
    if DigitAvailable then
      VoiceStop;
  until (mem[segment : 16] = 32) or DigitAvailable;
  DisableToneRec;
end;


procedure VRP_Record (RecordTime : string);
var
  SaveDigit  : byte;
begin
  SaveDigit := digit;
  digit := 255;          { invalid value }

  EnableToneRec;
  CMDEXEC('R'+RecordTime);
  repeat
  until (mem[segment:16] = 32) or (ToneDigit(digit) = '#');
  if (ToneDigit(digit) = '#') then
    begin
      DigitAvailable := False;
      VoiceStop;
    end;
  DisableToneRec;
  digit := SaveDigit;
end;

procedure VRP_SaveVoiceMessage (ToFile : string);
begin
  CMDEXEC('S'+ToFile);
end;

end.
```

```
unit MiscUtil;

interface
uses TpCrt,Dos,ToneCard,TpInt;

procedure waitkey;
procedure PseudoMusic;
procedure Restore_Int24H;

function StopWatch(hr,min,sec,sec100 : word) : real;
function AppendPath(path : string) : string;
function ValidNumberString(InputString : string) : Boolean;
function IntToStr(i: Longint): string;
function ToUpperCase(InputString : string) : string;
function ToneDigitToNumeric(InputDigit : Char) : byte;
function TimeUp(period : real; hr,min,sec,sec100 : word) : Boolean;
function Install_Int24H : Boolean;
function NoOfFiles(path : string; Attr : byte; Frec : SearchRec) : longint;

implementation
const
  MyIsrHandle  = 16;

procedure waitkey;
var
  ch    : char;
begin
  repeat until keypressed;
  while keypressed do ch := readkey;
end;


function StopWatch(hr,min,sec,sec100 : word) : real;
var
  hour,minute  : word;
  second       : word;
  second100    : word;
  t1,t2        : real;
begin
  GetTime(hour,minute,second,second100);
  if (hour = 0) and (hr <> 0) then hour := 24;
  t1 := hour*3600 + minute*60 + second + second100*0.01;
  t2 := hr*3600 + min*60 + sec + sec100*0.01;
  StopWatch := t1 - t2;
end;

procedure PseudoMusic;
var
  freq       : integer;
  i          : integer;
begin
  for i := 1 to 6 do begin
```

```
    randomize;
    freq := random(1000);
    sound(freq);
    delay(50);
    nosound
  end;
end;

function AppendPath(path : string) : string;
begin
  if path = '' then
    begin
      AppendPath := '';
      exit;
    end;
  if (path[length(path)] <> '\') then
    AppendPath := path + '\'
  else
    AppendPath := path;
end;

function ValidNumberString(InputString : string) : Boolean;
var
  indx   : integer;
begin
  for indx := 1 to length(InputString) do
     if not (InputString[indx] in ['0'..'9']) then
       begin
         ValidNumberString := False;
         exit;
       end;

  if InputString[1] = '0' then
    ValidNumberString := False
  else
    ValidNumberString := True;
end;


function IntToStr(i: Longint): string;
{ Convert any Integer type to a string }
var
  s: string[11];
begin
  Str(i, s);
  IntToStr := s;
end;

function ToUpperCase(InputString : string) : string;
var
  index  : integer;
begin
```

```
  for index := 1 to length(InputString) do
    InputString[index] := upcase(InputString[index]);
  ToUpperCase := InputString;
end;

function ToneDigitToNumeric(InputDigit : Char) : byte;
var
  buffer : byte;
begin
  case InputDigit of
    '1'..'9' : buffer := byte(InputDigit) - byte('1') + 1;
    '0'      : buffer := 10;
    '*'      : buffer := 11;
    '#'      : buffer := 12;
    'A'..'D' : buffer := byte(InputDigit) - byte('A') + 13;
  end;
  ToneDigitToNumeric := buffer;
end;

function TimeUp(period : real; hr,min,sec,sec100 : word) : Boolean;
var
  TimePassed : real;

begin
  TimePassed := StopWatch(hr,min,sec,sec100);
  if (TimePassed >= period) then
    TimeUp := True
  else
    TimeUp := False;
end;

procedure Int24H (Flags,CS,IP,AX,BX,CX,DX,SI,DI,DS,ES,BP : word);
interrupt;
var
  Reg : registers;

begin
  Reg.ax := 0;
end;

function Install_Int24H : Boolean;
begin
  if not InitVector($24,MyIsrHandle,@Int24H) then
    Install_Int24H := False
  else
    Install_Int24H := True;
end;

procedure Restore_Int24H;
begin
  RestoreVector(MyIsrHandle);
end;
```

```
function NoOfFiles(path : string; Attr : byte; Frec : SearchRec) : longint;
var
  NumFiles  : longint;
begin
  NumFiles := 0;
  FindFirst(path,Attr,Frec);
  while (DosError = 0) do
    begin
      if (Frec.Name <> '.') and (Frec.Name <> '..') then
        Inc(NumFiles);
      FindNext(Frec)
    end;

  case DosError of
    3,18 : NoOfFiles := NumFiles;
  else
    NoOfFiles := -1;
  end;
end;

end.
```

```
unit datastru;

interface

type
  ConfigStructure = record
    _FilePath        : String     ;
    _SamplingRate    : Char       ;
    _RecTime         : string     ;
    _PrimaryNodeName : string     ;
    _NodePrefix      : string     ;
    _ActionAfterAns  : string     ;
    _GreetMessage    : string     ;
    _SystemNodeNumber : longint   ;
  end;

  Next_N = record
             VoiceFile : string[12];
             NextNode  : longint;          { Points to next node }
             Action    : string[20];       { Specify action to be exec }
           end;

  NodeStructure = record
                    NodeID           : longint;
                    PreviousNode     : longint;
                    LevelIdentifier  : integer;
                    DigitToNode      : array[1..12] of Next_N;
                  end;

implementation

end.
```

```
unit LanMStru;

interface
type
  LanMailStructure = record
                       Password          : string[6];
                       UserName          : string[30];
                       Directory         : string;
                     end;

implementation

end.
```

Appendix 4

VOICE I/O CARD VRP-70

FOR ADVANCED APPLICATIONS

Second Edition on July 4th, 1990
Compunic Electronics Co., Ltd.

The Instruction Manual of the VRP-70 Voice I/O Card

TABLE OF CONTENTS

Chapter 1 INTRODUCTION

1.1 FEATURES

The VRP-7Ø voice I/O card of the Compunic is the latest model of the voice cards after VRP-3Ø and VRP-3ØA. It enables the IBM PC/XT/AT/AT386 and their compatibles to perform the functions of voice recording/playing and has the following features:

a. The software for voice applications can be written in any high level language and linked to any existent application software easily.
b. The voice data is handled by DMA and interrupt echnology, so you can run any program (including the graphics) and record or playback the sound concurrently with high efficiency.
c. The voice data rate can be set by software at the speed of 8K or 4K or 2K bytes/second.
d. It can be operated under LAN or WINDOWS/386 (refer to appendix 1 and appendix 3).
e. User-selectable I/O Port Addresses, DMA channels, and IRQ ports (refer to 1.4).
f. A socket is preserved for users to insert a PAL IC to protect the application software.
g. An additional input jack is to connect with an external keypad which has two keys with it. When the computer keyboard is occupied by other programs, e.g. under multi-tasking operation systems, the keypad can be used as an input device to control the functions of recording/playing. The input jack can also be connected to another external device such as various types of sensor which is necessary to set up an auto brief system.

1.2 THE APPLICATION AND FUTURE'S OUTLOOK OF THE VRP-7Ø CARD

Since the introduction of the VRP series, many software companies and research institutes have developed various applications for them. Following are some examples:

a. The voice card has been used in accompany with CAI programs,e.g. Mandarin Phonetic Symbols (see demo diskette), English pronounciational learning software, English conversational learning software, Malayan pronounciational learning software, German pronounciational learning software, and other CAI programs with voice in English, Malayan, German etc.
b. The voice card has been used in accompany with a CAI teaching editing program developed by the Institute for Information Industry enabling those teachers who are never trained to program to generate CAI software with sound as well as with pictures through the use of the editing program.
c. There is a news company combining the voice card with LAN and telephone network to offer the inquiry/answer service system for checking the result of the University Entrance Examination.
d. The inquiry/answer service is also applied for offering the instant information of the stock market.

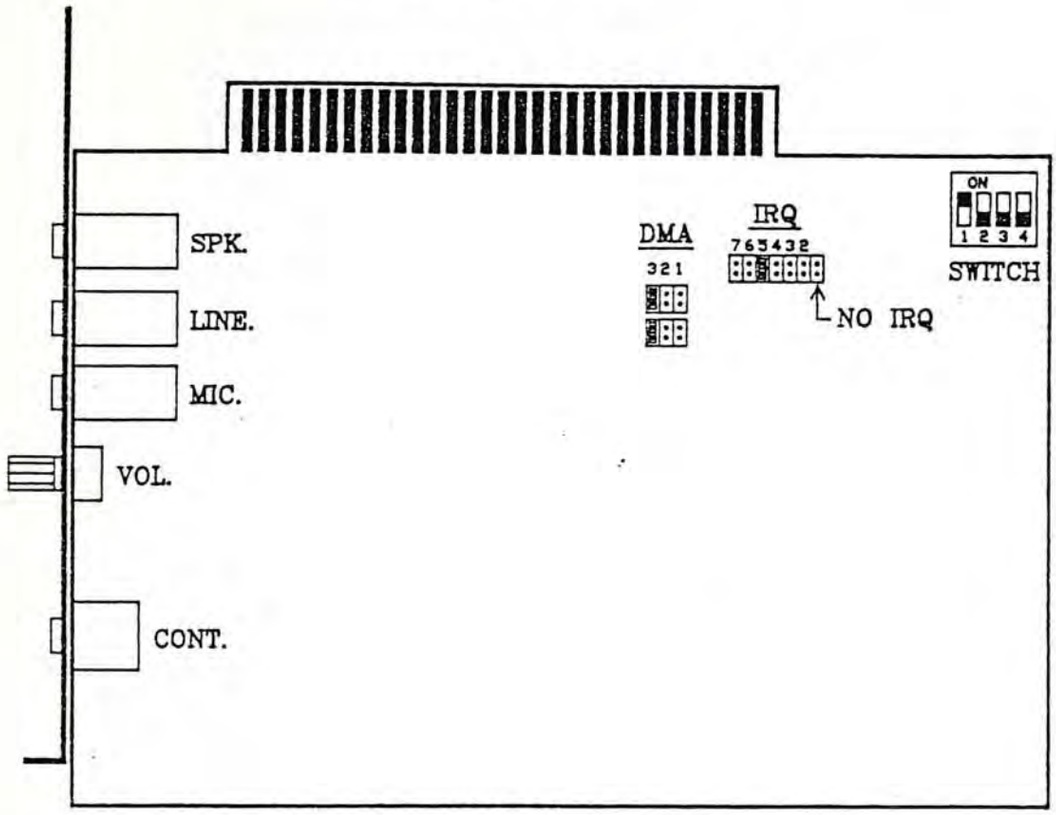
It can be forseen that the future application of the voice card will be more extensive, miscellaneous and popular. There are several other directions of development.

a. The voice card can be used in connection with image processing/graphic card, laser disk, videotape recorder/camera, disk camera, scanner etc. to develop multi-media integration system and to be applied in brief system, commercial advertisement system, and information inquiry/answer system.

b. The voice card can be used in accompany with tele-communication system to develop the voice mail system.And it can be further used in accompany with LAN to develop the software such as scheduler with voice, voice recognition, as well as dictation management which can be applied in offices to make office automation thorough.

1.3 A QUICK LOOK AT THE HARDWARE OF THE VRP-7Ø CARD

[Figure 1]



Model:VRP-70

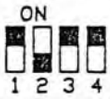
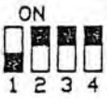
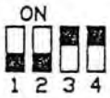
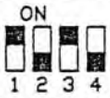
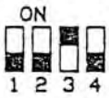
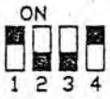
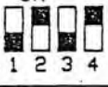
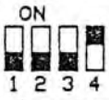
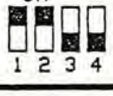
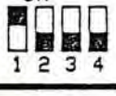
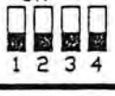
1. SPK. : a speaker jack.
2. LINE : a jack connected to line output or earphone output of tape recorders or another type of audio system.
3. MIC. : a microphone jack.
4. VOL. : a knob for adjusting the volume.
5. CONT. : A control input jack. The structure of this jack is the same as a stereo jack, but it's not applied for sound input. Inside the jack, there are two input contacts and a ground contact. Cooperating with "I" instruction of the driving routine, the application software can detect the "Input shorted to ground" signal of this jack to excute the functions (such as recording, playing ..., etc) at right time. It is very useful when the application software relative to voice runs under multi-tasking operation systems, such as Windows/386 or OS/2, and the whole keyboard may be occupied by another application programs. Besides connected to a key pad, this jack can also be connected to another external devices with open collector C.K.T. such as various type of sensor. It is useful for auto brief systems or security systems.

1.4 THE RESOURCES OF PC OCCUPIED BY THE VRP-70 CARD

a. INT 60H VECTOR: the addresses from 0000:0180H to 0000:0183H.
b. DATA BUFFER : the addresses from 7000:0000H to 8000:FFFFH. The total size is 128K bytes which can be changed by software. Please refer to chapter 3.
c. DMA : channel 1 to channel 3 selectable. The default is channel 3.
d. INTERRUPT : IRQ2 to IRQ7 selectable or without IRQ. The default is IRQ5.
e. I/O PORT : the I/O port address can be selected by DIP switch. The following figure describes the correspondent position of DIP switch and I/O port address.

**** After changing the hardware setting of VRP-70, you have to excute the SR9.COM to change the setting up table inside the SR9.COM .*

[Figure 2]

DIP SWITCH	I/O PORT ADDRESS	DIP SWITCH	I/O PORT ADDRESS	DIP SWITCH	I/O PORT ADDRESS	DIP SWITCH	I/O PORT ADDRESS
	209H		289H		309H		389H
	229H		2B9H		329H		3B9H
	249H		2D9H		349H		3D9H
	269H		* 2E9H		369H		3E9H

1. ▯ is 0, ▮ is 1, and the default is ▯▮▮▮ (0111) which represents the I/O port address of 2E9H.
2. If the above items, c, d, and e conflict with other interface cards, this voice card could not operate properly. Please change the position of the related jumper pin and run SR9.COM correspondently.

1.5 THE ELECTRICAL SPECIFICATION OF THE VRP-7Ø CARD

a. Microphone input: 1Ø mVpp max.
b. Line input: 5ØØ mVpp max.
c. Frequency response: 2ØØHZ - 3KHZ (±3dB).
d. S/N ratio: 35dB.
e. THD: 2%.
f. Speaker output: 2W max.

1.6 THE CONTENTS OF THE DISKETTES

1.6.1 DEMO Diskettes

a. VRP-7Ø Demo: This diskette contains some demonstration programs. Put the diskette into driver A, and key in "DEMO"; then you'll be ushered in the world of VRP-7Ø.
b. Audio & Color Graphics Demo: If your PC equipped with VGA display, you can excute this demonstration to see excellent effects of graphics and voice to be delivered synchronously. Which can lead you to the door of future Multi-Media computer!

1.6.2 UTILITY Diskette:

a. R7.EXE: a driving routine for the VRP-7Ø voice card that can run in a PC either with a hard disk or with a floppy disk. Please refer to chapter 3.

b. RPC7.EXE: a driving routine for the VRP-7Ø voice card that can run in a PC with hard disk only. Please refer to chapter 3.

c. SR9.COM: after the setting of I/O port address, DMA channels, and IRQ ports on the VRP-7Ø is changed, please excute this program.

d. VOICE.DAT: Voice data file.

e. BAS <DIR > :
BRECORD1.BAS: a program written in GWBASIC language for voice recording and playing.
CMD.ASM: an interface program written in assembly language for the driving routine and application software.
CMD.OBJ: the object file of CMD.ASM.
BRECORD2.BAS: a program written in QUICK BASIC language ver.4.5 for voice recording and playing.
BRECORD2.EXE: the execution file of BRECORD2.BAS.

f. PAS <DIR> :
PRECORD1.PAS: a program written in TURBO PASCAL language ver.5.Ø for voice recording and playing.
PRECORD1.EXE: the exectuion file of PRECORD1.PAS.

g. C <DIR> :

CRECORD2.C: a program written in MICROSOFT C language ver.5.1 or TURBO C language ver. 2.Ø for voice recording and playing.

CRECORD2.EXE: the execution file of CRECORD2.C.

CR6.ASM: an interface program written in assembly language for the driving routine and application software.

CR6.OBJ: the object file of CR6.ASM.

CRECORD1.C: a program written in TURBO C language ver.2.Ø using inline assembly language method for voice recording and playing.

CRECORD1.EXE: the execution file of CRECORD1.C.

h. DBASE <DIR> :

DRECORD.PRG: a program written in DBASE III PLUS language for voice recording and playing.

DBSET.ASM: an interface program written in assembly language for the driving routine and application software.

DBSET.BIN: the binary file of DBSET.ASM.

DBEXEC.ASM: an interface program written in assembly language for the dirving routine and application software.

DBEXEC.BIN: the binary file of DBEXEC.ASM.

DBGET.ASM: an interface program written in assembly language for the driving routine and application software.

DBGET.BIN: the binary file of DBGET.ASM.

i. CLIP <DIR> :

CLRECORD.PRG: a program written in CLIPPER language for voice recording and playing.

CL.ASM: an interface program written in assembly language for the driving routine and application software.

CL.OBJ: the object file of CL.ASM.

j. README.DOC: this file containing recent updates not included in this manual.

Chapter 2 ENTER THE WORLD OF THE VRP-7Ø CARD

2.1 HOW TO INSTALL THE VRP-7Ø CARD

Installation Procedure:

a. Power off the computer.
b. Plug in the VRP-7Ø card to any one of the expansion slots.
c. Connect the microphone and speaker to the specified jack on the VRP-7Ø card.
d. Power on the computer.
(Notic that DOS is not contained in the utility diskette.)

2.2 HOW TO BACKUP THE UTILITY DISKETTE

(Notice that DOS is not contained in the utility diskette.)

2.2.1 The PCs with Hard Disks

If you have a hard disk in your PC, please do as following:

a. Please select the hard disk driver as the default driver (usually C:) and make sure that the "XCOPY.EXE" file resides in the hard disk.
b. Insert the utility diskette into driver A.
C. Key in:

C> XCOPY A: C:/e/s <CR>

2.2.2 The PCs without Hard Disks

If you don't have a hard disk in your PC, please backup a diskette according to the following procedure:

a. Insert the DOS diskette into driver A.
b. Key in:

A> DISKCOPY A: B: <CR>

c. Insert the utility diskette into driver A and a blank one unformatted into driver B and press any key to start backup after the diskcopy message displayed on the screen.

2.3 HOW TO LOAD THE DRIVING ROUTINE OF THE VRP-7Ø

Please make sure that the current directory contains the driving routine and setting up file, SR9.COM.

2.3.1 The PCs with Hard Disks

If you have a hard disk in your PC, please key in:

A> RPC7 (or R7) <CR>

2.3.2 The PCs without Hard Disks

If you don't have any hard disk in your PC, please key in:

A> R7 <CR>

** After you have keyed in the above command, the interface between the VRP-7Ø card and various high level languages is established.*

2.3.3 The Command Line Parameter of Driving Routine

The driving routine resides in memory after being loaded. If you want to quit it then to release the memory, please key in:

A>R7 Q <CR>

or

A>RPC7 Q <CR>

2.3.4 The Screen Message Displayed after Driving Routine Is Loaded

a. "Driving Routine: R7.EXE V1.01"
" Date: 5-8-1991 "
"Copyright Compunic Electronics Co.. Ltd."

It means the program is loaded successfully and the speaker will echo with a sound "POP".

b. "load voice program fail"

It means the program is loaded unsuccessfully.

c. "voice program already exists."

It means the program has already been loaded.

2.3.5 A Setting up Program SR9.COM

SR9.COM is a setting up program for the driving routine. When the jumper for I/O port address, DMA channels and IRQ ports of your VRP-70 card is changed, you have to run SR9.COM to change the setting up table inside the SR9.COM. When the driving routine is loaded, the table inside the SR9.COM will be checked by the driving routine. Wherefor SR9.COM need to exist with the driving routine under the same directory. The default value for the I/O port address is 2E9H, DMA channel is channel 3 and IRQ port is 5. If the other interface cards use the same values, they will conflict with the VRP-70 card. You have to change the jumper or DIP switch and excute SR9.COM.

2.4 · HOW TO RUN THE DEMONSTRATION PROGRAMS

1. The demonstration diskette contains a batch file and some example programs.
   a. Put **"VRP-70 Demo"** diskette into driver A.
   b. Key in : A:>DEMO <CR>
      The DEMO.BAT will invoke the computer to run these demonstration programs.

2. If your PC equipped with VGA display, the **"Audio & Color Graphics Demo"** diskette will show you the excellent effects of voice and graphics to be delivered synchronously.
   a. Put **"Audio & Color Graphics Demo"** diskette into drive A.
      (If your PC equipped with a hard disk, It's better that copy all files on the diskette into hard disk.)
   b. Key in : A:>DEMO <CR>
      The DEMO.BAT will invoke the computer to run these demonstration programs.

2.5 HOW TO REDUCE THE BACKGROUND NOISE

When you are recording voices, it's unavoidable to produce background noise which has intensive relationship with the quality of switching power supply and the grounding condition of your PC. To improve the quality of recording, please do as following:

a. Please don't touch the metal head of the mic when you are holding it to record voices.
b. Please get in touch with any part of the metal shield of your PC with the other hand when you are recording voices.
c. Please reduce the background noise of the recording environment to the less.

Chapter 3: THE DESCRIPTION OF DRIVING ROUTINES R7.EXE AND RPC7.EXE

3.1 THE FUNCTIONS OF THE DRIVING ROUTINE

RPC7.EXE or R7.EXE provides the functions of recording, playing, stopping the voice, changing the voice data rate, setting the memory blocks as the data buffers, and storing or retrieving the voice data from the mass storage media stored on either a diskette or a hard disk.

3.2 THE ENTRY POINT OF THE DRIVING ROUTINE

The entry point of the driving routine is the same as that of "INT 60H". If the driving routine has been loaded successfully, we can find the entry point stored at the addresses 0000:0180-0000:0183; otherwise the contents of these four bytes will all be 0's. The voice application program can check whether the driving routine has been loaded successfully or not by checking these four bytes. Please refer to the source files of the example programs written in various kinds of languages contained in the utility diskette to get the usage to develop your own program.

3.3 THE COMMAND BUFFER

The application programs written in any high level language can initiate the driving routine, RPC7.EXE or R7.EXE to execute the functions of recording, stopping, ...etc. by issuing the interrup instruction "INT 60H" or calling the entry point. The application programs should put the command in the command buffer (abbr. CMDBUF) in a correct format prior to issuing "INT 60H" or calling the entry point. The segment address of the CMDBUF is the same as that of the vector of "INT 60H".

Saying that
data occupy the address 0000:0182H as S2S1 (Hex), and
data occupy the address 0000:0183H as S4S3 (Hex),
the address of the first byte of the CMDBUF is

S4 S3 S2 S1:0010H.

After executing the commands, the application software can also get some information from the CMDBUF. The first byte of the CMDBUF will be filled with the result of some of the commands. From the 33rd byte to the 40th byte of the CMDBUF will be filled with the length information of the voice data after the completion of the commands, "Record", "Play", "Stop", and "Load". These eight bytes of the CMDBUF contain a hex number in ASCII form.

For example:

```
                          1st ..... 33th ---------------- 40th
value of the CMDBUF : XX ..... 30 30 30 31 43 41 37 32
```

The length of the voice data is 0001CA72H bytes.

3.4 THE COMMANDS OF R7.EXE AND RPC7.EXE

Following are the foramts of the commands for putting commands into the CMDBUF and the functional description of the commnads.

a. Test : T
b. Addr. : An000 or An800
c. Record : Rnnnn
d. Size : AA
e. Record : R — these two instructions for R7.EXE only.
f. Play : P
g. Stop : E
h. Save : Sfname
i. Load : Lfname
j. Mode : Mn
k. Input : I
1. Monitor : On or Of
m. Karaoke . : OKn
n. Echo : ECn — these two instructions support VSM-75K only.

a. Test command: (TEST)
Format : T
Description : T is an identifier.
Function: This instruction can distinguish the version of the driving routine. After the execution of this command, the driving routine is R7.EXE if the first byte of the CMDBUF is 30H, or the driving routine is RPC7.EXE if the first byte of the CMDBUF is 31H.

b. Address command: (ADDR)
Format : An000 or An800
Description : A is an identifier.
n000(Hex) stands for the starting segment address of the voice data buffer.
Function : This instruction changes the present starting segment address 7000H of the voice data buffer to address n000H. The length of the voice data buffer will be set to 64K bytes automatically.

c. Recording command: (RECORD)
Format: Rnnnn
Description: R is an identifier.
nnnn(Decimal) stands for the total seconds of recording time and has to be between 0000 and 9999 seconds.
Function: The driving routine initiates the voice card and begins to record voices after the issue of this command. When recording comes to an end, the driving routine will fill a code 20H in the first byte of the CMDBUF and the length information of the voice data in the bytes from the 33rd byte to the 40th byte of the CMDBUF.
By issuing this command to the RPC7.EXE, the maximum continuous recording/playing time is limited to the capacity of your hard disk. For example, a 20MB hard disk allows you to have a period of 85-minute recording time at the voice data rate of 4K bytes/sec. or 170-minute recording time at the voice data rate of 2K bytes/sec. If this command is issued to the R7.EXE and the voice data buffer is 64K bytes, the maximum recording/playing time is 16 sec. at the voice data rate of 4K bytes/sec. or 32 sec. at the voice data rate of 2K bytes/sec.

d. Size command: (SIZE)
Format: AAwwww:xxxx yyyy:zzzz
or
AAwwww:xxxx,yyyy:zzzz
Description: AA is an identifier.
wwww:xxxx is the starting address of the voice data buffer.
yyyy:zzzz is the ending address of the voice data buffer.
Function: This command can be accepted by the R7.EXE driving routine only.
By issuing this command, the address and size of the voice data buffer can be reassigned.

e. Recording command: (RECORD)
Format : R
Description : R is an identifier.
Function: This command can be accepted by R7.EXE driving routine only.
It instructs the driving routine to initate the voice card to record and place the voice data into the voice data buffer denoted by "AA" command. When recording is completed, the driving routine will fill a code 20H in the first byte of the CMDBUF and the length information of the voice data in the bytes from the 33rd byte to the 40th byte of the CMDBUF.
The length of recording time depends on the size of the data buffer you set previously. Instructing "AA7000:0000 8000:FFFF" first, then issuing this command will enable you to have a period of recording/playing time of 32 sec. (at the voice data rate of 4K bytes/sec.) or 64 sec. (at the voice data rate of 2K bytes/sec.).

If the "AA" and "R" commands and the length information of the voice data are used properly and efficiently, the multiple pieces (or paragraphs) of the voice data can be placed into a single data buffer or different voice data buffers. Each piece (or paragraph) of the voice data can be played or recorded randomly and in-stantly. Please make sure that the RAM space available is large enough.

f. Play command: (PLAY)
Format: P
Description: P is an identifier.
Function: The driving routine initiates the voice card to read and play the voice data after the issue of this command. The playback will stop at the end of this paragraph even though it does not fully occupy the voice data buffer which is set by "A" or "AA" command. When playback is completed, the the driving routine will fill a code 20H in the first byte of the CMDBUF and the length information of the voice data in the bytes from the 33rd byte to the 40th byte of the CMDBUF.

g. Stop command: (STOP)
Format: E
Description: E is an identifier.
Function: This command is to stop recording or playing.
After this action, the driving routine will fill the length information of the voice data in the bytes from the 33rd byte to the 40th byte of the CMDBUF.

h. Saving command: (SAVE)
Format: Sfname
Description: S is an identifier.
"fname" is a file name. Please consult the rules of the file name from MS-DOS manual (ver. 2.0 or above).
Function: This command is to save the voice data onto the disk under the file name you assigned. The previous data with the same file name will be overwritten.
After the execution of this command, if the first byte of the CMDBUF is changed into 20H, it means the file has been saved successfully; otherwise the error code will be placed in the first byte of the CMDBUF. Please refer to TABLE 1 to identify and check the error.

Error Codes:

00H. VRP70 Card absent	04H. Too many open files
01H. Invalid functions	05H. Access denied
02H. File not found	06H. Invalid handle
03H. Path not found	0CH. Invalid access

TABLE 1

i. Loading command: (LOAD)
Format: Lfname
Description: L is an identifier.
"fname" is a file name. Please consult the rules of the file name from MS-DOS manual (ver. 2.0 or above).
Function: This command is to load the data of the assigned file into the voice data buffer. If the first byte of the CMDBUF is changed into 20H, it means the loading process is successful. The length information of the voice data will be filled in the bytes from the 33rd byte to the 40th byte of the CMDBUF. If loading fails, the error code will be filled in the first byte of the CMDBUF. Please refer to TABLE 1 to identify and check the error.

j. Mode command: (MODE)
Format: Mn
Description: M is an identifier.
n represents Q, D, or S.
Function: The voice data rate can be changed with commands MQ, MD or MS. The preset voice data rate is 4K bytes/sec.
By issuing the MQ command, the driving routine sets the voice data rate at the speed of 8K bytes/sec.
By issuing the MD command, the driving routine sets the voice data rate at the speed of 4K bytes/sec.
By issuing the MS command, the driving routine sets the voice data rate at the speed of 2K bytes/sec.

Please notice that the sound quality at a high voice data rate is better than that at a low voice data rate.

k. Input command: (INPUT)
Format: I
Description: I is an identifier.
Function: Issuing this command to detect the "short to ground" signal of the control input (ref. 1.3). the control input can be connected to an external contacting devices, for example, a key pad with two keys, saying key Ø and key 1. By issuing this command, the 3Øth byte of the CMDBUF will be filled with a code. The bit Ø of this code represents key Ø, while the bit 1 represents key 1. If bit Ø is 1, it means key Ø had been pressed. If bit 1 is 1, it means key 1 has been pressed.

1. Monitoring command: (MONITOR)
Format: On or Of
Description: O is an identifier.
Function: This command is to switch the monitoring function to "On" or "Off" position before proceeding the recording. If you record the voice from the tape recorder, you may issue "On" command to active the monitoring function. But if you record the voice with microphone, please issuing "Of" command to switch the monitoring function "Off" to keep from howling effect that you may hear from the speaker. The monitoring function is default to "Off" position.

m. Karaoke command: (KARAOKE)
Format: OKn
Description: OK is a keyword,
n is between Ø to 15, the default value is Ø.
Function: Control the Karaoke On/Off and the delay time of echo effects of VSM-75K. n = Ø means truning Karaoke off.
n = 1 to 15 means truning Karaoke on and define the delay time of echo effects. Set the value at about 1Ø to 12, you will get better sound quality.

n. Echo Effects command: (ECHO)
Format: ECn
Description: EC is a keyword,
n is between Ø to 15, the default value is Ø.
Function: Control the echo effects of VSM-75K. The "n" value is same as OK command. This command is similar to OK command. The difference between OK and EC is that OK command not only controls the delay time of echo effects, but also performs On and Of (refer to MONITOR command). You can get echo effects when you are playing voice by using EC command.

** The KARAOKE and ECHO commands are only available on VSM-75K.*

5 THE DIFFERENCES BETWEEN RPC7.EXE AND R7.EXE

RPC7.EXE:

a.This driving routine is designed for PCs with hard disks exclusively or with floppy disks on some high speed PCs like 80386 or NEAT. It can accepts two commands, "AnØØØ" and "Rnnnn".
b.The command "AnØØØ" instructs RPC7.EXE to set the starting segment address of the voice data buffer which occupies a RAM size of 64K bytes beginning at this segment address.
c.The command "Rnnnn" instructs it to set the the length of recording time by second and activates the recording function of the voice card.
d.During recording or playing, the voice data is swapped between the 64K RAM buffer and the hard disk, so the maximum continuous recording/playing time depends on the capacity of the hard disk.

R7.EXE

a.This driving routine can be used on PCs with either hard disks or floppy disks. It can accepts two commands, "AA" and "R".
b.The command "AA" instructs R7.EXE to set the starting and ending addresses of the voice data buffer randomly in RAM where the addresses are not occupied by other programs or data.
c.The command "R" instructs it to initiate the voice card to perform recording function; then the voice card will place the voice data in the voice data buffer set by the "AA" command.
d.Unlike RPC7.EXE using the swapping technology, it supports only the voice card to write/read the voice data into/from the RAM buffer during recording or playing, so the continuous recording/playing time is in accordance with the size of the voice data buffer set by users.
e.It can accepts the commands "AnØØØ" and "Rnnn" too, but the maximum voice data buffer is fixed to 64K bytes and the parameter "nnnn" of the command "Rnnnn" has to be a number between ØØØØ and ØØ16.

*The rest of the commands such as Play, Stop, Save, Load,...etc. are functionally exactly the same in both of them.

Chapter 4 SOFTWARE EXAMPLES WRITTEN IN FOUR KIND OF HIGH LEVEL LANGUAGE

A11 the source codes of the programs witten in BASIC, PASCAL, C, and DBASE III PLUS can be found in the utility diskette.Fo11owing are brief exp1anations for each program.

4.1 BASIC LANGUAGE

4.1.1 Examp1e Program BRECORD1.BAS

BRECORD1.BAS resides in the "BAS" subdirectory of VRP-7Ø utility diskette. It is a recording/playing examp1e program written in GWBASIC 1anguage.

. Line 13Ø to 17Ø: statements for finding out the segment 1ocation of the CMDBUF and the starting address of the driving routine.
. 13Ø DEF SEG=Ø
. 15Ø SL=PEEK(&H182): SH=PEEK(&H183): S=SH*256+SL; get segment
. 16Ø CL=PEEK(&H18Ø): CH=PEEK(&H181): CMD=CH*256+CL; get offset
. 17Ø DEF SEG=S
. Line 117Ø to 121Ø: statements for putting instructions into the CMDBUF.
. 117Ø FOR I=1 TO LEN (CMD$)
. 118Ø X=ASC(MID$(CMD$,I,1))
. 119Ø POKE I+15,X
. 12ØØ NEXT I
. 121Ø RETURN
. Line 17Ø to 3ØØ: a subroutine for checking if the driving routine has been 1oaded. If the driving routine has been 1oaded, it will identify what has been 1oaded (R7.EXE or RPC7.EXE).
. Line 31Ø to 42Ø: the main program.
. Line 1ØØØ to 116Ø: a subroutine for setting up screen.
. Line 2ØØØ to 259Ø: a subroutine for recording.
. Line 3ØØØ to 318Ø: a subroutine for playing.
. Line 4ØØØ to 4Ø4Ø: a subroutine for stopping recording or playing.
. Line 5ØØØ to 512Ø: a subroutine for saving voice data.
. Line 6ØØØ to 618Ø: a subroutine for 1oading voice data.
. Line 7ØØØ to 734Ø: a subroutine for changing the starting segment address of the voice data.
. Line 8ØØØ to 824Ø: a subroutine for changing the voice data rate.
. Line 9ØØØ: program to exit.
. Line 1ØØØØ to 1ØØ7Ø: a subroutine for timer.
. Line 11ØØØ to 1253Ø: a subroutine for transferring the starting and ending addresses of the voice date buffer to a voice data length (byte).
. Line 13ØØØ to 13Ø7Ø: a subroutine for getting the 1ength of the voice data.
. Line 14ØØØ to 14Ø5Ø: a subroutine for transferring a string of hexadecima1 address to a decima1 number.

4.1.2 How to Compile BRECORD1.BAS

1. MICROSOFT BASIC compiler ver. 2.Ø
   A. Delete the statement "CMD=CH*256+c1" in program.
   B. Key in: BASCOM filename/o/v <CR>
   C. Key in: LINK filename+CMD <CR>
   D. Then the execution file is created. (* CMD.OBJ is the object file of CMD.ASM and you can find CMD.ASM in the utility diskette.)

2. TURBO BASIC compiler ver. 2.ØØ
   A. Please don't delete any statement.
   B. Subtitue "CALL CMD" for "CALL ABSOLUET CMD" in the whole program.

4.1.3 Example program BRECORD2.BAS

BRECORD2.BAS resides in the "BAS" subdirectory of the VRP-7Ø utility diskette written in QUICK BASIC language ver. 4.5.

- Line 1Ø to 22Ø : statements for getting the segment address of the CMDBUF and the entry point of INT 6ØH, setting the initial value of the voice data rate and the the voice data buffer, identifying which ver. of the driving routine has been loaded, displaying the main menu, and setting function keys.
- Line 2ØØØ to 3ØØØ : a subroutine for recording.
- Line 3ØØØ to 4ØØØ : a subroutine for playing.
- Line 4ØØØ to 5ØØØ : a subroutine for stopping recording or playing.
- LIne 5ØØØ to 6ØØØ : a subroutine for saving voice data.
- Line 6ØØØ to 7ØØØ : a subroutine for loading voice data.
- Line 7ØØØ to 8ØØØ : a subroutine for reassigning the voice data buffer.
- Line 8ØØØ to 9ØØØ : a subroutine for changing the voice data rate (MODE$). If MODE$ is S, the voice data rate is 2*1Ø24 bytes/sec. If MODE$ is D, the voice data rate is 4*1Ø24 bytes/sec. If MODE$ is Q, the voice data rate is 8*1Ø24 bytes/sec.
- Line 12ØØØ to RETURN: a subroutine for transferring the starting and ending addresses of the voice data buffer to a voice data length (byte).
- CMDSET : a subroutine for putting commands into the CMDBUF.
- FUNCTION atv ($x) : a subroutine for transferring a string of hexadecimal address to a decimal number.
- SUB dispmsg : a subroutine for displaying the main menu.
- SUB tcal (t) : a subroutine for displaying the countdown of time during recording or playing voices.

* How to Compile BRECORD2.BAS Using QUICK BASIC Compiler Ver. 4.5

a. Key in: QB brecord2/L <CR>

b. Select the "Run" function in the main_menu and select the "Make Exec File " function in its sub_menu, and start to compile.

c. The execution file is created.

4.2 PASCAL LANGUAGE

4.2.1 Example Program PRECORD1.PAS

The example program PRECORD1.PAS residing in the "PAS" subdirectory of the VRP-70 utility diskette is a recording/playing example program written in TURBO PASCAL language ver. 5.0.

- procedure CMDEXEC : a subroutine for calling the entry point of INT 60H to initiate the driving routine RPC7.EXE (or R7.EXE).
- procedure CMDSET : a subroutine for putting instructions into the CMDBUF.
- Procedure SHOW_CURRENT_DATE_AND_TIME : a subroutine for displaying date and time currently.
- Procedure BACK_TIME : a subroutine for the countdown of time.
- Procedure READ_CHECK : a subroutine for checking the input data of procedure V_ADDR.
- Procedure DECIMAL_CONVERT_HEXDECIMAL : a subroutine for converting decimal into hexadecimal.
- Procedure HEXDECIMAL_CONVERT_DECIMAL : a subroutine for converting hexadecimal into decimal.
- Procedure GET_BUFFER_SEGMENT_AND_OFFSET : a subroutine for getting segment and offset addresses of the voice data buffer.
- Procedure COUNT_VOICE_DATA_LENGTH : a subroutine for calculating the length of the voice data.
- Procedure RESET_VOICE_DATA_BUFFER : a subroutine for procedure V_RECORD or procedure V_LOAD to adjust the voice data buffer.
- Procedure DRAW_TITLE : a subroutine for displaying the name of the function key.
- Procedure INIT : a subroutine for setting up screen and displaying time.
- Procedure V_STOP : a subroutine for stopping playing or recording.
- Procedure V_SAVE : a subroutine for saving voice data.
- Procedure V_RECORD : a subroutine for recording.
- Procedure V_PLAY : a subroutine for playing.
- Procedure V_LOAD : a subroutine for loading voice data.
- Procedure V_ADDR : a subroutine for setting the starting or ending segment address of the voice data.
- Procedure V_MODE : a subroutine for setting the voice data rate.

4.2.2 How to Complie and Link PRECORD1.PAS Using TURBO PASCAL Complier Ver. 5.Ø

a. Under MS-DOS system prompt, key in:
   X:> TURBO <CR>
b. TURBO PASCAL main screen contains Main menu, Edit window, Output window and Bottom line. The Main menu has seven items of functions which are File, Edit, Run, Compile, Options, Debug and Break/Watch.
c. When a program has been edited, users can select the "Compile" function on main menu and there will display a subwindow below. Set Destination to Disk on the subwindow to compile the program.
d. The execution file PRECORD1.EXE is created after compiling.

4.3 C LANGUAGE

4.3.1 Example Program CRECORD2.C

CRECORD2.C residing in the "C" subdirectory of the VRP-7Ø utility diskette is a recording/playing example program written in MICROSOFT C language ver. 5.Ø/5.1 or TURBO C language ver. 1.5/2.Ø.

. main() : main program.
. counter(int x,int y) : a subroutine for counting time.
. init() : a subroutine for setting up screen.
. v_save() : a subroutine for saving voice data.
. v_load() : a subroutine for loading voice data.
. v_record() : a subroutine for recording.
. v_play() : a subroutine for playing.
. v_stop() : a subroutine for stopping recording or playing.
. v_addr() : a subroutine for setting the starting or ending segment address of the voice data.
. v_mode() : a subroutine for setting voice data rate.

CR6.ASM is an interface program written in assembly language for the driving routine and application software.

. check() : a subroutine for checking if the driver routine is existent.
. cmdset(char *cmdname) : a subroutine for putting instructions into the CMDBUF.
. cmdexec() : a subroutine for executing INT 6ØH (or calling the entry point) to initiate the driving routine RPC7.EXE or R7.EXE.
. cmdget(int i) : a subroutine for getting the data of the 1st byte of the CMDBUF.

4.3.2 How to Create the Execution File of CRECORD2.C

1. Assemble CR6.ASM using MICROSOFT Assembler ver. 5.1. There are three models to create CR6.OBJ. Choose the same model as that of your MICROSOFT C compiler.

   a. Large Model : Key in

   ```
   MASM /MX /Dmodel=large /Dlang=C CR6
   ```

   b. Medium Model : Key in

   ```
   MASM /MX /Dmodel=medium /Dlang=C CR6
   ```

   c. Small Model : Key in

   ```
   MASM /Mx /Dmodel=small /Dlang=C CR6
   ```

   * Make sure the file, MIXED.INC is in your diskettebefore assembling.

2. Compile and link CRECORD2.C using MICROSOFT C compiler ver. 5.0/5.1.

   a. Large Model : Key in

   ```
   CL /AL CRECORD2.C CR6.OBJ
   ```

   b. Medium Model : Key in

   ```
   CL /AM CRECORD2.C CR6.OBJ
   ```

   c. Small Model : Key in

   ```
   CL /AS CRECORD2.C CR6.OBJ
   ```

3. Compile and link CRECORD2.C using TURBO C compiler ver. 1.5/2.0.

   a. Large Model : Key in

   ```
   TCC -ml CRECORD2.C CR6.OBJ
   ```

   b. Medium Model : Key in

   ```
   TCC -mm CRECORD2.C CR6.OBJ
   ```

   c. Small Model : Key in

   ```
   TCC CRECORD2.C CR6.OBJ
   ```

4.3.3 How to Use the Inline Assembly of TURBO C to Modify the Example Program

CRECORD1.C residing in the "C" subdirectory of the VRP-70 utility diskette is an alternative example program using inline assembly language of TURBO C 1.5/2.0. It is necessary to copy TASM.EXE or MASM.EXE into the same directory including CRECORD1.C before compiling and linking. Compile as following, Key in :

TCC -B CRECORD1 (note : small model)

4.4 DBASE III PLUS LANGUAGE

4.4.1 Example Program DRECORD.PRG

DRECORD.PRG residing in the "DBASE" subdirectory of VRP-70 utility diskette is a recording/playing example program written in DBASE III PLUS language. It can be executed with the R7.EXE driving routine only. If you select RPC7.EXE as the driving routine, please refer to the command "R" and "A" in section 3.4 to modify your programs.

. The DO WHILE LOOP displays the MENU first; then the user is requested to input a function key to select the function he needs.

0. Return to DBASE.
1. Start recording.
2. Start playing.
3. Stop recording/playing.
4. Save voice data to a floppy or hard disk.
5. Load voice data from a floppy or hard disk.
6. Display imformation including the segment address of the voice data buffer, voice data rate, and time.
7. Input the starting and ending segment addresses of the voice data buffer.
8. Input the voice data rate.

. CON : a subroutine for displaying the message "press any key to return".
. ASCII : a subroutine for converting a string of hexadecimal address to a decimal number.
. SEG : a subroutine for setting the starting or ending segment address of the voice data.
. MOD : a subroutine for setting voice data rate.

4.4.2 DBGET.ASM, DBSET.ASM and DBEXEC.ASM Subroutines

The subroutines, DBGET.ASM, DBSET.ASM, and DBEXEC.ASM residing in the "DBASE" subdirectory of the VRP-7Ø utility diskette are interface programs written in assembly language for the driving routine and application software.

- DBGET.ASM
  a subroutine for getting the first byte of the CMDBUF.

- DBSET.ASM
  a subroutine for putting instructions into the CMDBUF.

- DBEXEC.ASM
  a subroutine for initiating the driving routine RPC7.EXE or R7.EXE.

- These assembly language programs, through the processing of MASM, LINK, and EXE2BIN, are used to generate binary files for DBASE III PLUS.

4.5 CLIPPER LANGUAGE

4.5.1 Example Program CLRECORD.PRG

CRECORD.PRG residing in the "CLIP" subdirectory of the VRP-7Ø utility diskette is a recording/playing example program written in CLIPPER language. It can be executed with the R7.EXE or RPC7.EXE driving routine.

The DO WHILE LOOP displays the MENU first; then the user is requested to move the LIGHT-BAR to select a function.

1. RECORD : Start recording.
2. PALY : Start playing.
3. STOP : Stop recording/playing.
4. SAVE : Save voice data to a floppy or hard disk.
5. LOAD : Load voice data from a floppy or hard disk.
6. INFORMATION : Display the imformation including the segment address of voice data buffer, voice data rate, and time.
7. SEGMENT : Input the starting and ending segment addresses of the voice data buffer.
8. MODE : Input the voice data rate.
9. EXIT : Return to DOS system.

- CON : a subroutine for displaying information after the execution of every function.
- ASCII : a subroutine for converting a string of hexadecimal address to a decimal number.
- SEG : a subroutine for setting the starting or ending segment address of the voice data.
- MOD : a subroutine for setting voice data rate.

* CL.ASM containing four subroutines is an interface program written in assembly language for the driving routine and application software.

1. CLGET:
   a subroutine for getting the first byte of the CMDBUF.

2. CLSET:
   a subroutine for putting instructions into the CMDBUF.

3. CLEXEC:
   a subroutine for initiating the driving routine, R7.EXE or RPC7.EXE.

4. CLCHK:
   a subroutine for checking if the driver routine has been existent.

4.5.2 How to Compile CLRECORD.PRG Using CLIPPER Language

1. Assemble CL.ASM to CL.OBJ using MICROSOFT Assembler ver. 5.1 or above.

2. Compile CLRECORD.PRG to CRECORD.OBJ using CLIPPER complier.

   Key in : CLIPPER CLRECORD <CR>

3. Link CLRECORD.OBJ and CL.OBJ using PLINK86.EXE to create the execution file.

   Key in : PLINK86 FI CLRECORD, CL LIB CLIPPER.EXTEND <CR>

4.5.3 How to Compile And Link DRECORD.PRG Written in DBASE III PLUS Directly Using CLIPPER

1. Omit "LOAD" and "RELEASE" statements in DRECORD.PRG when calling subroutines written in assembly language.

2. Declare "PUBLIC" in the assembly language program. (* Please refer to the CLIPPER manual about "call" command.)

4.5.4 The Differences between DRECORD.PRG And CLRECORD.PRG

These two programs are almost all of the same except calling subroutines written in assembly language and their functions.

a. CLRECORD.PRG uses the "T" command of the driveing routine to check its version (R7.EXE or RPC7.EXE) to decide to use "An000", "Rnnnn" or "AA", and "R" commands while DRECORD.PRG can not.

b. CLRECORD.PRG can check the first byte of the CMDBUF when recording/playing is performing. If the first byte of the CMDBUF is 20H, it will stop recording/playing. DRECORD.PRG cannot do so. It just controls recording/playing according to the timer.(* CLIPPER can get the return code by assembly language programs, but DBASE III plus can not.)

c. CLRECORD.PRG can get the bytes from the 33rd through the 40th of the COMBUF to decide the length of the voice data after any one of the commands, "RECORD", "PLAY", "STOP", and "LOAD" is commpleted while DRECORD.PRG can not.

Appendix 1: HOW TO OPERATE THE VRP-7Ø CARD UNDER LAN

LAN has been applied widly to connect PCs in office environment to achieve the goal of resource sharing.

VRP-7Ø voice card is an interface card with powerful functions. It can work easily with network in conjunction with well written programs.

Following are the conditions that VRP-7Ø card can work with network.

1. Hardware: Any kind of network hardware compatible with either ETHERNET card or ARCNET card.

2. Software: DLINK, NOVEL ELS LEVEL II, ADVANCED V2.X, ...etc.

If the PC works as a non-dedicated file server under network, it can't be installed with the VRP-7Ø card and run voice application programs.

Executing voice programs under network is the same as that under MS-DOS system except in one point which is that all the voice application programs and relevant data have to be stored in the file server hard disk to achieve the aim of resource sharing.

Appendix 2: HOW TO ALLOCATE THE VOICE DATA BUFFER THROUGH MS-DOS OPERATING SYSTEM

To avoid the voice data buffers conflict with the memory area that the programs may use, the best choice is to allocate buffers through MS-DOS operating system.

When requested for allocating a block of voice data buffer, the DOS operating system will send back the allocated area if the free memory is available. The value of this starting address is used as a reference value for the starting address of the voice data buffer.

If the starting segment address of the allocated memory is 27C0H, the starting address of the data buffer can be set at 27C0H for R7.EXE driving routine or at 2800H for RPC7.EXE driving routine.

The program listed below is an example of allocating and releasing memory program written in TURBO PASCAL. (You may use other languages as well.)

```
program memaddr;
 type mem_ok=0;
 var
     ch, mode :char;
     Mem_size,Mem_start  :word;

function Mem_allocate(MSize:word; var Mem_start:word):byte;
var {the subroutine for allocating voice data buffer}
regs :registers;
Cflag :byte;
Mem_Free_str :string;
begin
       with regs do
        begin
           BX := msize*64;
           AH := $48;
           MSDOS(regs);
           Cflag := Flags and $01;

           if Cflag=0 then Mem_start := AX
           else case AX of
              7 : Mem_msg := 'MCB(Memory Control Block) BAD.';
              8 : begin
                   Mem_Free := BX;
                   str(BX/64:5:1,Mem_free_str);
                   Mem_msg := 'Not enough memory. ' + 'Current free memory: ' + Mem_free_str + ' KBytes'
                  end;
             end;
           Mem_allocate := Cflag;
       end;
end;
```

```
function Mem_Release(M_start:word):byte;
var {the subroutine for releasing voice data buffer}
 regs :registers;
 Cflag :byte;
 begin
     with regs do
      begin
          ES := M_start;
          AH := $49;
          MSDOS(regs);
          Cflag := flags and $01;
          if Cflag <> 0 then
          case AX of
               7 : Mem_msg := 'MCB(Memory Control Block) BAD.';
               8 : Mem_msg := 'Release memory error.';
          end;
      end;
      Mem_Release := Cflag;
end;

BEGIN  { main program }
          { 1. allocating memory  }
     writeln('MEMORY ALLOCATE.');
     write('How many Kbytes do you need ? ');
     readln(M_size); { input x value , x KBytes if x =  64 : 64K  Bytes 128 : 128k Bytes  }
     if Mem_allocate(M_size,Mem_start)=Mem_OK
     then writeln('OK.  Mem_start: ', Mem_start)
     else writeln(Mem_msg);

          { 2. releasing  memory }
     write('Memory Release.  ',Mem_start,' Right ?(y) ');
     ch := readkey;
     if (ch='N') or (ch='n') then
     begin
          write(' Input: ');
          readln(Mem_start);
     end;
     if Mem_release(Mem_start)=Mem_ok then
     begin
          writeln('OK.');
          M_size := 0;
     end
          else writeln(mem_msg);
END.
```

Appendix 3: HOW TO OPERATE THE VRP-7Ø CARD UNDER WINDOWS/386

To ensure that the voice application programs will run properly under WINDOWS/386, it's necessary to get voice buffers through DOS operating system. The example and demonstration programs metioned in the previous chapters didn't discuss this problem for the readability because they can run without any disturbance only if they could get voice data buffers without any conflict with other programs or data blocks under DOS operating system. The environment under WINDOWS/386 is more complicated than that under DOS operating system, so we strongly recommend that any voice application program to be executed under WINDOWS/386 should get safe blocks of data buffers through DOS. Any information about the allocation of voice data buffer, please refer to appendix 2. After a voice application program whose data buffer is allocated through DOS has been developed properly under DOS, we can try the program under WINDOWS/386. Do the procedure as following:

1. Initiate WINDOWS/386 under MS-DOS system.
2. Enter MS-DOS exective after the initiation of WINDOWS/386.
3. Determine how to execute an external program through Program Information File (PIF) owing to the voice application program is not an internally designed program of WINDOWS. For instance, if WINDOWS wants to know how much memory to allocate for a certain program, the PIF has to be used.
4. Set up PIF if it has not been set up in the voice program file. Do as following to set up the PIF:
   A. Execute PIFEDIT.EXE file under MS-DOS executive to initiate PIF edit program, entering all the information according to the screen of the editor.
   B. There are two points concerning the use of memory in connection with the voice application program.
      a. KB required.
      b. KB desired.
   C. Set the default value of KB required and KB desired. Please refer to the following:
      Voice application program: 64K bytes (including the driving routine).
      Voice data buffer: 128K bytes, driving routine R7.EXE.
      Voice data buffer: 64K bytes, driving routine RPC7.EXE.

      When the driving routinre is R7.EXE, the default value of KB required and KB desired is 192K or 2ØØK bytes.

      When the driving routine is RPC7.EXE, the default value of KB required and KB desired is 128K or 14ØK bytes.

      If normal operation cannot be obtained, please set the value at a higher level to achieve the proper operation.

D. Example:
Voice application program file name is RECORD.EXE and voice driving routine is R7.EXE.

a. Set up a batch file named VM.BAT.

```
X:>COPY CON: VM.BAT
R7
RECORD <F6>
```

b. Set up a PIF based upon the above illustration. Its file name is VM.PIF and .PIF is an extension file name.

c. Execute VM.PIF under MS-DOS executive to execute the the driving routine and voice application programs.