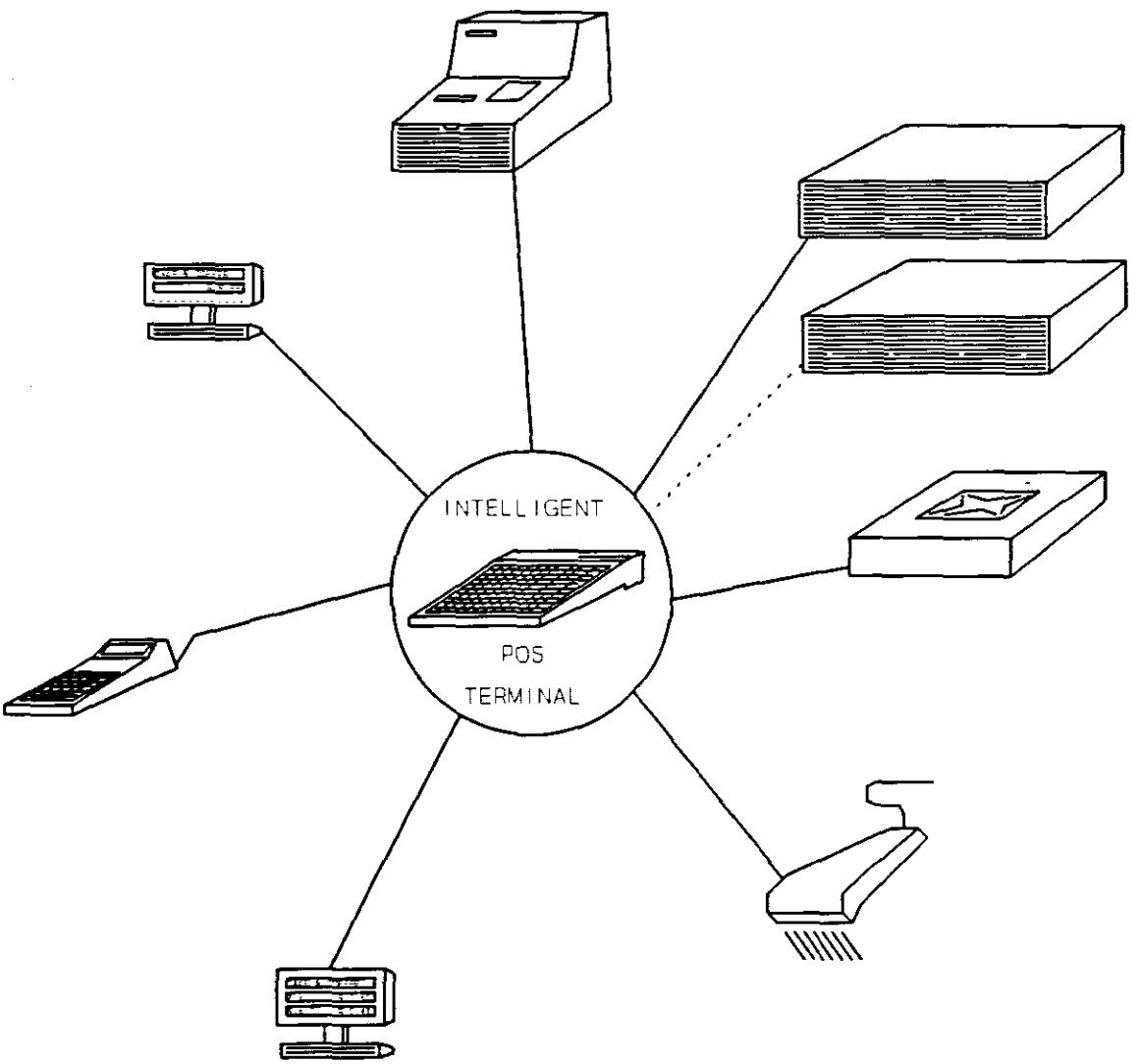
INTELLIGENT POINT OF SALE TERMINAL

Document Name: T5.DOC

Original: December 1991

Prepared by: H.Keown

Revised (_____)



INTELLIGENT POINT OF SALE TERMINAL THESIS

by **HARVEY KEOWN**

Thesis submitted in part fulfilment of the requirements for the Masters Diploma in the School of Electrical Engineering at the Cape Technikon.

Date of submission: December 1991.

DECLARATIONS:

I declare that the contents of this thesis has been entirely prepared by myself and represents my own work. The opinions contained herein are my own and not necessarily those of the Technikon.

Signed: R Keown Date: 08 : 06 : 92

ACKNOWLEDGMENTS:

I would like to use this opportunity to thank the following people, all have assisted in shaping my career:

Mr. Gerhard Kopatz (Managing Director of Ankerdata) for giving me the opportunity to further my studies and mainly having faith in the projects that I have generated.

Mr. Suliman Harnekar (Electrical Engineer, Hardware and Software) and Mr. Gary Robas (Electrical Engineer, Hardware) for unselfishly imparting their knowledge and experience.

Mr. Eric Seiderer (Production Manager) for supporting me through the various design stages.

To my wife, Gillian, who has always been the main support. (thanks for doing the dishes)

Finally, I would like to thank those who have proof read this thesis.

To the above mentioned people, **THANK-YOU** for the role each of you had in moulding my career.

TABLE OF CONTENTS

TABLE OF CONTENTS

TABLE OF CONTENTS

TABLE OF CONTENTS

SUMMARY

The main reason for this project was "import replacement", as all our existing Point of Sale and Electronic Equipment had to be imported from Japan. After the Government's steps to curb imports by placing extremely high levies on imported goods, it was decided to produce a completely local product.

From past experience it was obvious that customer requirements varied greatly. This gave rise to the inception of a modular system, enabling the customer to "mix and match" modules to their requirements.

The concept is to use a HOST computer controlling a differential communications line with a maximum of 255 terminals which are all individually addressable. Each individual terminal would in turn control an internal differential communications line, called PNET, which is an acronym for "Peripheral Network".

A decision was made to make all the peripherals intelligent, thereby alleviating the processor of all menial tasks. All peripherals local to the terminal would be connected to this network. The configuration can be seen graphically by refering to Figure 0-1. The communications protocol used is more sophisticated than that used for RS232 devices. The protocol has a POLL - ACKNOWLEDGE structure, where the HOST has complete control of the loop.

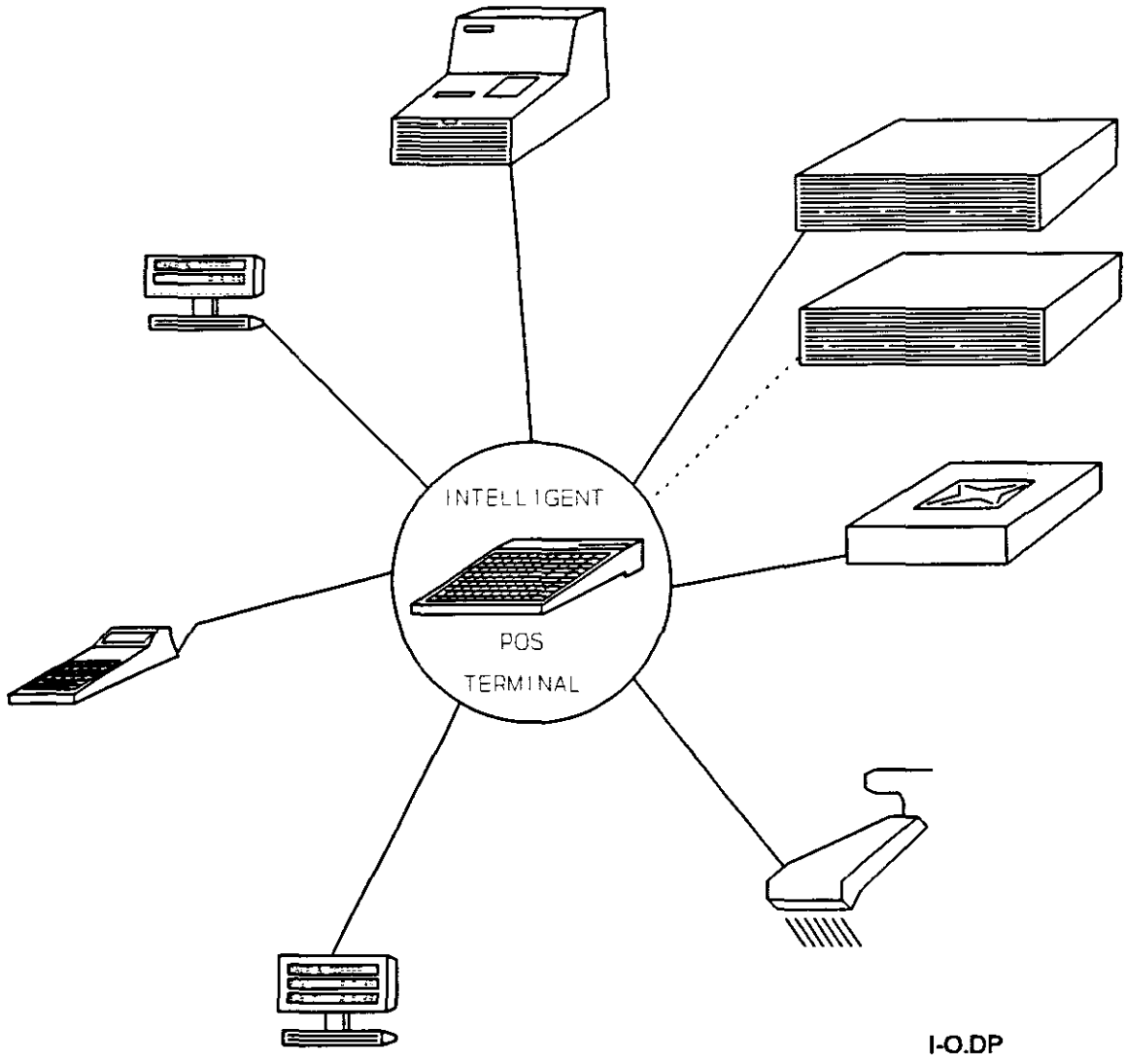


Figure 0-1. *INTELLIGENT TERMINAL*

This thesis is therefore a detailed description of the INTELLIGENT TERMINAL which forms an integral sub-section of each terminal. This can be seen graphically, by referring to the Figure 1-1, Chapter 2, page 2.

The operation of the terminal had to be very similar to that of the imported POS terminals and had to meet the following specifications:

Addressable up to a maximum of 255 terminals
Maximum 128 key keyboard
Magnetic card reader
Two, 2 lines X 16 character Liquid Crystal Displays (LCD's)
Buzzer
Two cash drawers
Sixteen status Light Emitting Diodes (LED's)
4 Bit control lock
Two standard serial RS232 ports
1 X Laser Hand Scanner Interface
1 X Laser Desk Top Scanner Interface
2 Line X 16 character vacuum fluorescent display
2 Line X 20 character vacuum fluorescent display
3 Line X 16 character vacuum fluorescent display
Pin pad for electronic fund transfer (EFT) At the Point of Sale (EFT POS)

The terminal had to be capable of interfacing to all the above devices as well as controlling all of them. The INTELLIGENT TERMINAL essentially handles the interface to the above devices, and communicates the data to the HOST (XT motherboard) via PNET. The HOST had to be capable of beeping the buzzer, opening the drawer, displaying messages on the LCD's etc., as well as

receiving data from any of the input devices. It was decided to use an Intel 8032 micro controller as the "heart" of the INTELLIGENT TERMINAL. All the software was written on an IBM AT, using Intel's Macro Assembler ASM51, and the Relocatable Linker RL51.

INTRODUCTION

CORPORATE PROFILE

Ankerdata, formerly known as Anker Data Systems (ADS) has always been a market leader in the field of Point Of Sale Equipment (POS), Cash Registers (both mechanical and electronic), Electronic Scales and Mainframe Computing.

The driving force behind this extremely prosperous corporate power is the Managing Director, Gerhard Kopatz. As a direct result of his insight and business skills, Ankerdata has developed since the mid 60's, into a highly competitive company. At present there are in excess of 50 branches country wide, as well as branches in foreign countries.

Ankerdata has a long history of local Research and Development. During the early 70's various niches were found in the local Receipting, Hotel and Point of Sale markets. There was no hardware available to satisfy customer requirements, and this was the basis for the inception of the Ankerdata R&D Department.

The first model was a Receipting machine called the 4900, based on an Intel 8085 microprocessor. This machine was directed at customers such as the Post Office, Customs and Excise and Inland Revenue. A lucrative niche was also apparent in the hotel industry, where a machine for the Front Desk was required. This

led to the development of the second generation of machines for the Front Desk, called the 5900. This extremely flexible and powerful machine satisfied all the requirements, and was still installed in major hotels such as the Capetonian, Heerengraght and President Hotels until a few years ago. The final model in this particular series was the 7900. This was designed to satisfy the requirements of various universities, with regard to the student accounts. This system could perform full stock control and manage the various student accounts. An installation at Stellenbosch University was upgraded two years ago to our newer generation of terminals. Valuable design and marketing experience was gained from this development phase at Ankerdata. This together with exorbitant import costs, prompted the Managing Diréctor to start the next generation of Point Of Sale Equipment.

During 1988 the Government revised the tax structure of imported goods. This meant that all electronic equipment that Ankerdata was importing was subjected to extremely high import costs. A direct result of this, was an unrealistically high unit cost to the end user. This prompted the Managing Director to investigate market requirements, which led to the decision to locally design and develop a Point Of Sale Terminal. This machine would be geared for the middle to upper market segments, with full communications to a store controller, enabling full Back Office control with extensive reporting capabilities.

PROJECT INCEPTION

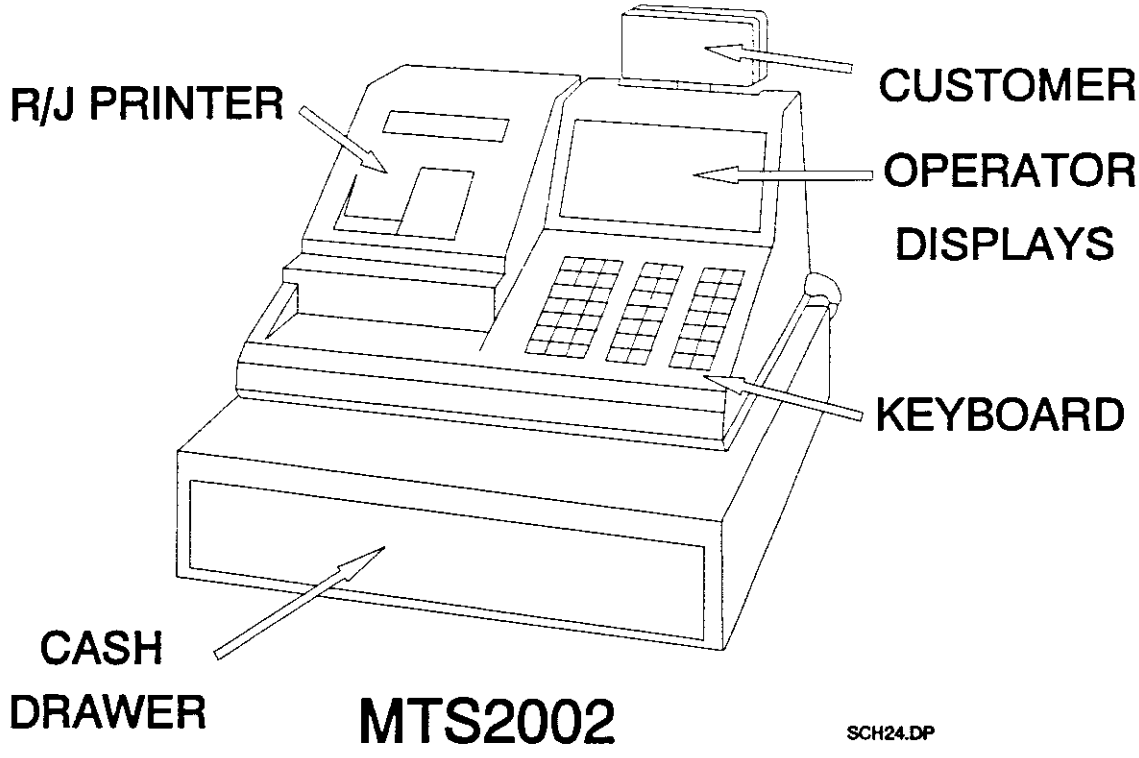


Figure 0-2. Integrated Terminal - MTS2002

The project was started in early 1988 and I was extremely fortunate to be a member of the initial development team. With the advent of the IBM PC, design philosophies changed, and Ankerdata, being in the forefront of technology, noted the swing toward an open-ended design architecture. This was the starting point for the MTS2002 Point Of Sale Terminal. The terminal was based on a standard XT motherboard, with a propriety power supply, expansion I/O card and various peripherals. The greatest advantage of choosing this architecture was that all software could be developed on a standard PC, and advantage could be taken of the vast number of debuggers, compilers and utilities that

were available. The availability of various expansion cards meant that standard cards could be used, greatly reducing the design and debugging phase. This meant that valuable design time could be more effectively spent, designing specific I/O cards to satisfy our hardware requirements.

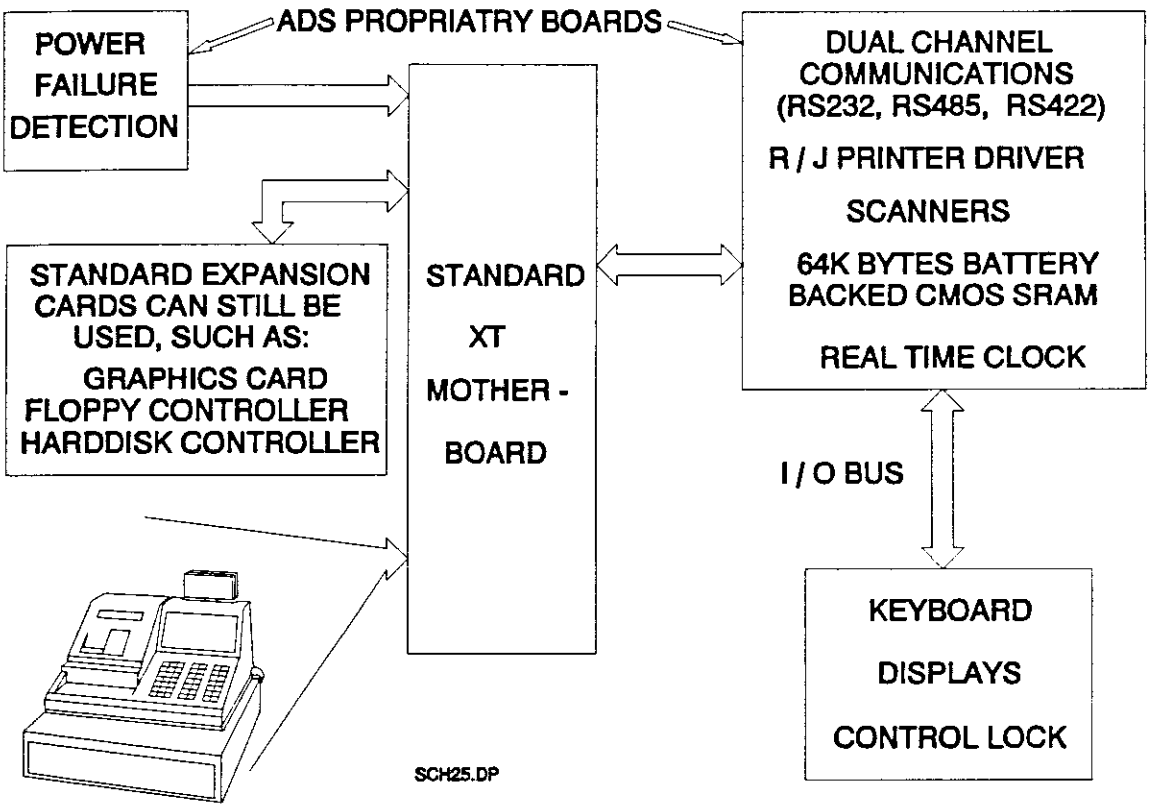


Figure 0-3. MTS2002 Configuration

The MTS2002 was an integrated design, having the processing, keyboard, printer and displays all housed in a single cabinet. This, unfortunately would prove to be a major stumbling block in the future. Due to the power of the terminal, all software requirements could be met relatively easily, the converse did not

however apply to the hardware. Each customer, having their own idiosyncrasy, would require slight changes to the existing terminal base. This led to many versions of the MTS2002, combined with this was the associated headache of maintaining so many variations. This was an extremely popular machine, being able to satisfy customer requirements, but was extremely costly to maintain.

The obvious solution was to design a modular system, which would enable the customer to "mix 'n match" the hardware to their requirements. The standard XT motherboard would still be retained, enabling the existing software base to be utilised, as well as the previously mentioned advantages. Unknown to us at the time was that this terminal would eventually be the most powerful, flexible, popular and greatly exported machine that Ankerdata has thus far developed.

The communications to the Back Office was maintained, but the structure of the Input/Output devices was radically changed. The communications to the Back Office is based on a 7 bit ASCII protocol using the RS485 bus type electrical connection. This meant that a maximum of 255 terminals could be connected to the HOST computer, which could either be a 286 or 386.

This bus type communications structure was now extended one more level, to the terminal. A second communications bus was designed for the internal operation of the terminal, and this was called

the Peripheral Network (PNET). The XT motherboard, internal to the terminal would control this network, which would have the flexibility of having any type of peripheral device connected to it. Initially the terminal needed the same functionality as the MTS2002, with the possibility of upgrading and adding more devices in future. A minimal system would therefore comprise of a keyboard, receipt/journal printer, two 2 line X 16 character vacuum fluorescent displays (operator and customer) and various support peripherals. Each device would be addressable, enabling the HOST to have total control of the communications bus. The following peripherals which are attached to PNET have been in full production since mid-1990:

Intelligent 2 line X 16 Character Vacuum Fluorescent Display

Intelligent 3 line X 16 Character Vacuum Fluorescent Display

Intelligent 2 line X 20 Character Vacuum Fluorescent Display

Intelligent 2 line X 16 Character LCD

RS232 to RS485 Protocol Converter for the SW1 Receipt/ Journal Printer

Intelligent Point Of Sale Termianl, which forms the basis of this thesis.

The first phase was to design a ROM Emulator card. We decided to design this card to fit into an expansion slot of the PC. This meant that the binary file could be directly loaded into the target system, where the code could be tested within seconds. The only debugging tools available were a digital oscilloscope, LCD, LED's and the port pins. This made debugging extremely difficult and tedious, especially when coding at chip level. Ankerdata has since purchased an In-Circuit Emulator, which greatly speeds up debugging.

The INTELLIGENT TERMINAL had to interface to the devices, as specified in Chapter 2, "INTELLIGENT TERMINAL SPECIFICATIONS". The INTELLIGENT TERMINAL had to transmit any incoming information to the HOST, and output any information that the HOST might transmit to it. The INTELLIGENT TERMINAL had to report any change in status to the HOST. All software was written in assembler, which is a lot more tedious than writing in a high level language. Intel's Macro Assembler and Relocatable Linker were used for the generation of the code. The rest of the thesis is dedicated to the INTELLIGENT TERMINAL, and therefore a brief description of the various chapters is necessary.

CHAPTER 1

The communications protocol is described, as well as the commands that control the INTELLIGENT TERMINAL.

CHAPTER 2

This contains the hardware description for the INTELLIGENT TERMINAL and all the Periphery attached to it.

CHAPTER 3

This is the complete Assembler listing for the INTELLIGENT TERMINAL.

CHAPTER 4

This contains the schematic diagrams and component layout for the INTELLIGENT TERMINAL.

CHAPTER 5

This is the parts list for the INTELLIGENT TERMINAL.

CHAPTER 6

This is the bibliography for the complete project.

Since full production of the terminal started during the latter quarter of 1990, more than 1000 units have been installed at the various Post Offices country wide, as well as installations at major customer sites. The International market was also penetrated with the export of a number of terminals to Europe.

This must surely be an indication of the degree of professionalism in the design as well as marketing sections of Ankerdata.

Due to the constant undertaking of research programs, Ankerdata is kept at the forefront of technology, ensuring the local Point of Sale market can compete within world markets.

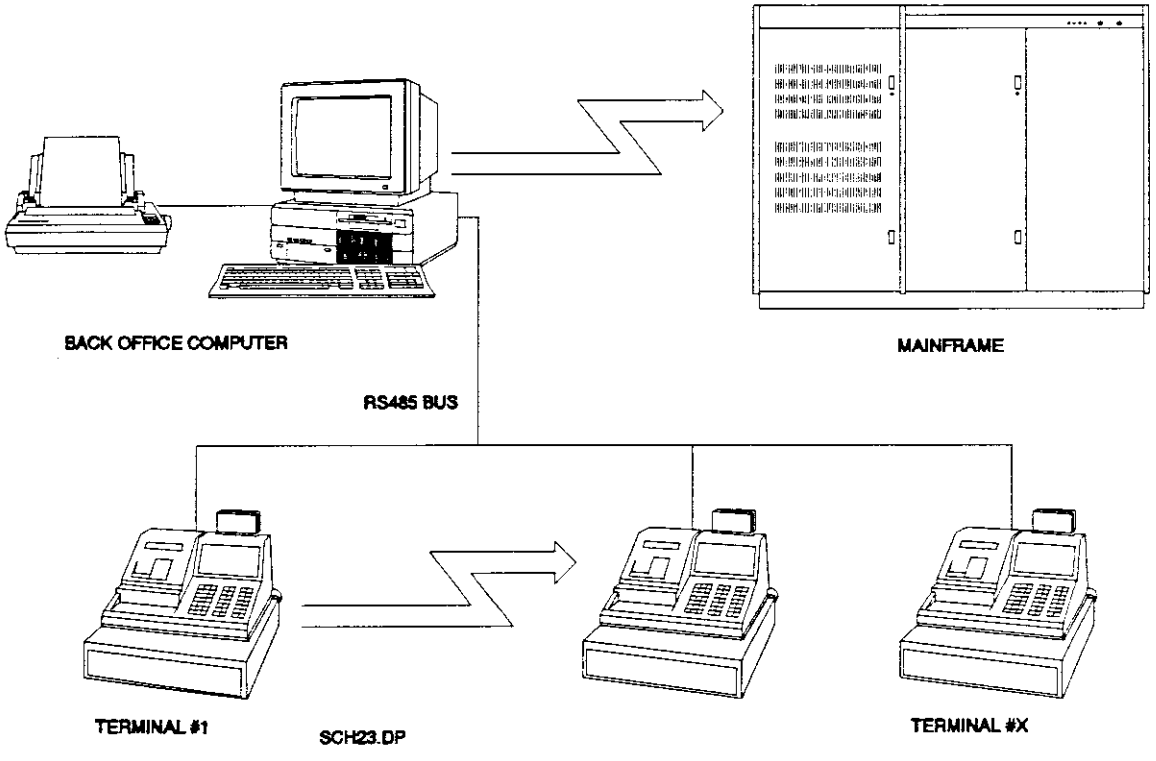
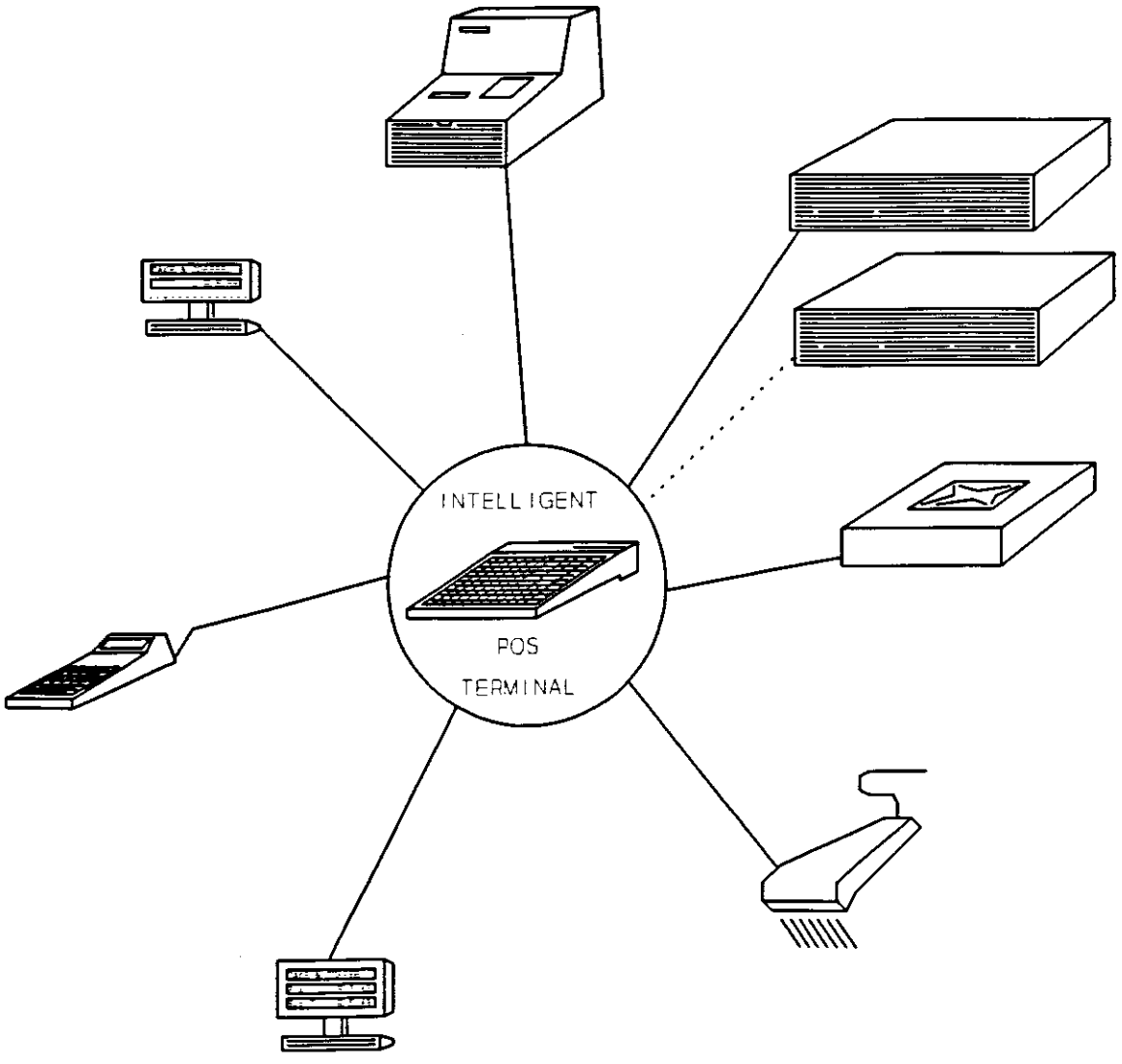


Figure 0-4. POS Terminals and Back Office Computer

INTELLIGENT POINT OF SALE TERMINAL

SOFTWARE DESCRIPTION



CHAPTER 1

1. COMMUNICATIONS INTRODUCTION

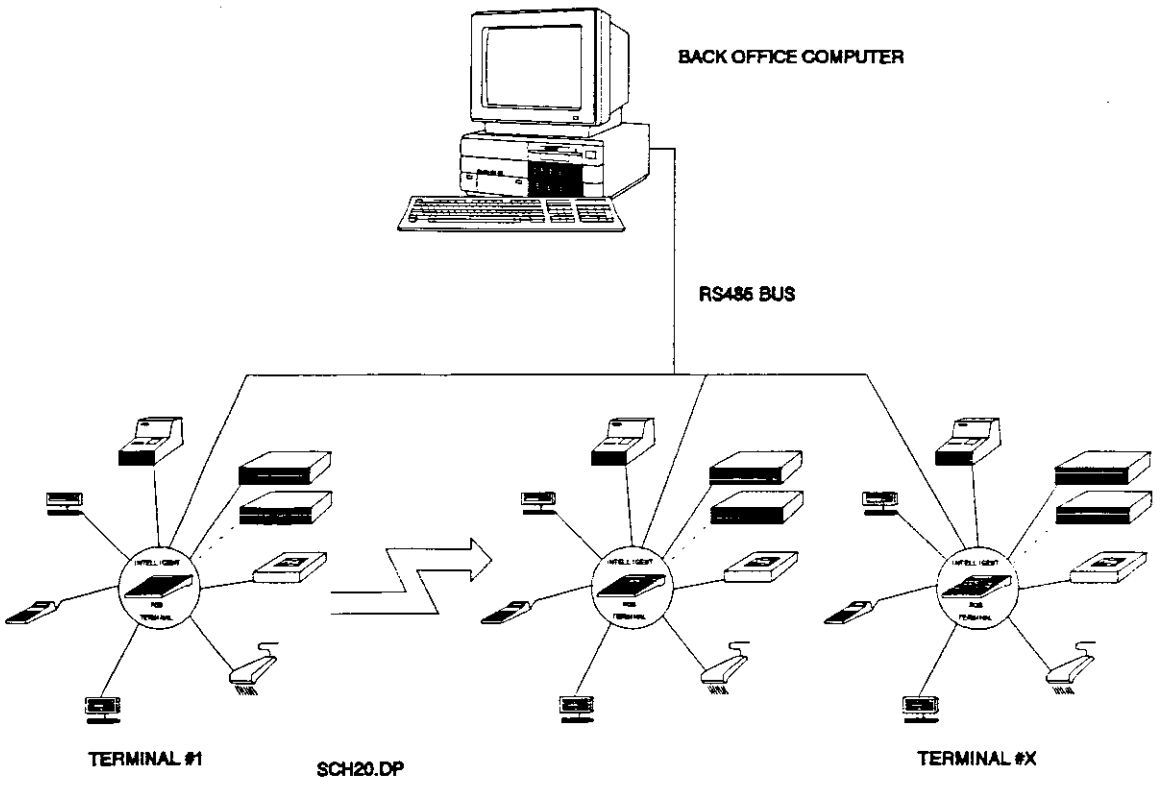


Figure 1-1. INTELLIGENT TERMINAL-Back Office Computer

This specification describes the protocol used on the Peripheral Network (PNET), which is the internal communications bus of each terminal.

Each terminal could consist of the following devices:

INTELLIGENT TERMINAL

SW1 Printer

2 X 16, 3 X 16, 2 X 20 Vacuum Fluorescent Displays

Handscanner and Desktop Scanner

Pin Pad

2 X Cash Drawers

Bi-Directional Magnetic Card Reader

The communications between the HOST and the various devices should be seen as two independent layers. The lower layer is the protocol layer, which takes care of the integrity of the data during data transfers. The upper layer is the command layer, which is used to control the various operations on the INTELLIGENT TERMINAL.

2. TRANSMISSION SPECIFICATION

1.	Communication System	Half Duplex
2.	Transmission Rate	19.2 KBaud
3.	Connection Control System	Poll/Select
4.	Response Method	ACK and NAK
5.	Error Control System	BCC Check: LRC Method Illegal Response Address Check
6.	Transmission Code	ASCII
7.	Transmission Mode	Transparent Mode
8.	Transmission Bit Order	LSB First
9.	Bit Configuration	DATA: 8 START: 1 STOP: 1 PARITY: NONE
10.	Electrical Connection	RS 485

Table 1-1. Transmission Specification

3. TRANSMISSION CONTROL CODES

SYMBOL	ASCII CODE	FUNCTION
DLE	10H	Data link escape code which becomes significant in combination with the following character.
ENQ	05H	Used for requesting line control (data link establish phase) or requesting response (text transfer phase).
EOT	04H	End of transmission character which makes all the stations on the line enter the control state.
ACK	06H	Acknowledge character.

SYMBOL	ASCII CODE	FUNCTION
NAK	15H	Negative acknowledge character which is used when the preceding block is not received correctly.
DLE.STX	10H-02H	Indicates the start of text or block, in transparent mode.
DLE.ETX	10H-03H	Indicates the end of text in transparent mode.

Table 1-2. Transmission Control Codes

4. TRANSMISSION FORMAT

Text is transmitted according to the following format.

DLE	STX	TEXT	DLE	ETX	LRC

1. Text length is 50 characters maximum. (This is due to the internal receive buffer of the INTELLIGENT TERMINAL).

2. The following processing is performed for specific characters appearing in **TEXT** transmission.

   (a)
   The first **DLE** code in the **DLE.DLE** sequence in the received text is deleted from the text and not included in the count of the text length.

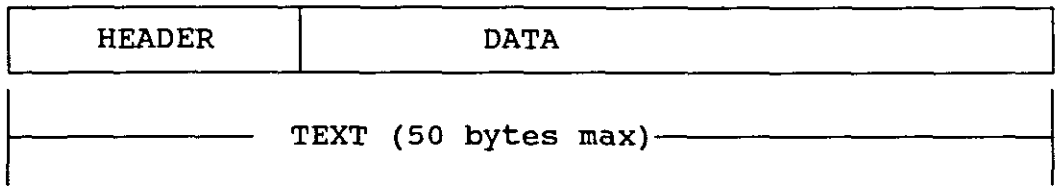
   (b)
   When a **DLE** is detected in the transmitted text, it is sent as a **DLE.DLE** sequence. But the first **DLE** is not included in the count of the text length.

   (c)
   The purpose of the **DLE.DLE** sequence is to indicate that the **DLE** is not a protocol control code, but is part of the data in the TEXT field.

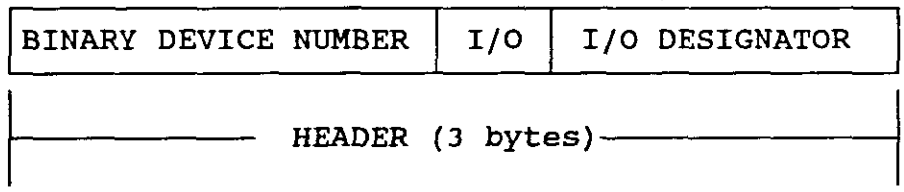
3. **BCC** (Block Check Character) uses LRC and is represented by one byte. (Refer to section 6 "Error Control").

4.1 TEXT FORMAT

Text is composed of the header field and data field.



The header field is composed of the control codes:



Binary Device Number: (1 Byte)

The device number refers to the physical device on the differential communications line. This could be an INTELLIGENT TERMINAL or an INTELLIGENT PRINTER. It is possible to have up to 16 devices on the line, numbered from 1 to 16.

I/O: (1 Byte)

As the INTELLIGENT TERMINAL is mainly an INPUT / OUTPUT device, all messages for output devices, such

as the LCD, LED's, drawer solenoids will be preceded by an 'O'. Therefore messages which are generally sent from the HOST to the INTELLIGENT TERMINAL will be preceded by an 'O'.

All input devices on the INTELLIGENT TERMINAL, such as the lock, keyboard, scanners will be preceded by an 'I'. Therefore messages which are sent from the INTELLIGENT TERMINAL to the HOST will be preceded by an 'I', to indicate that an input device has just been read.

I/O DESIGNATOR: (1 Byte)

This refers to a specific I/O device on the INTELLIGENT TERMINAL.

eg: 'L' lock
'K' keyboard
'R' drawers

DATA FIELD

This will be the data which is either to be sent to the INTELLIGENT TERMINAL or data which is received from the INTELLIGENT TERMINAL.

5. TRANSMISSION CONTROL PROCEDURES

The following are the five types of states in the transmission control procedures.

(1) Connection Phase

(2) Data Link Establish Phase

(3) Text Transfer Phase

(4) Data Link End Phase

(5) Disconnection Phase

5.1 CONNECTION PHASE (NO PROCESSING)

Transits to the data link establish phase.

5.2 DATA LINK ESTABLISH PHASE

A data link is established through transmission of the calling sequence.

5.2.1 DATA LINK ESTABLISH PHASE AT HOST

5.2.1.1 SELECTING (HOST HAS DATA TO SEND TO TERMINAL)

(1) Format

EOT	AD1	AD2	SEL	ENQ

AD1, AD2:

Represents the destination address in unpacked ASCII format .(Example terminal number 01H = ASCII 30H 31H)

SEL:

71H = 'q'

(2) Response Processing for Selecting Sequence at the HOST.

The following processing is performed for the response received in reply to the selecting sequence.

(a) Reception of ACK

A data link is established and processing transits to the transfer phase described in Section 5.3

(b) Reception of NAK

No data link is established and processing transits to the selecting sequence for the next terminal.

(c) Reception of Invalid Response

Response is ignored and processing maintains the response wait state.

5.2.1.2 POLLING (HOST REQUESTS TERMINAL FOR AVAILABLE DATA)

(1) Format

EOT	AD1	AD2	POL	ENQ

AD1, AD2:

Represents the destination address in unpacked ASCII format. (Example terminal number 01H = ASCII 30H 31H)

POL:

70H = 'p'

(2) Response Processing for Polling Sequence at the HOST.

The following processing is performed according to the response received in reply to the polling sequence.

(a) Reception of Text

A data link is established and processing transits to the text transfer phase described in Section 5.3

(b) Reception of EOT

No data link is established and processing transits to the polling sequence for the next station.

(c) Reception of Invalid Response

Response is ignored and processing maintains the response wait state.

5.2.2 DATA LINK ESTABLISH PHASE AT TERMINAL

The following processing is performed according to the response received while waiting for the polling sequence from the HOST.

(1) Reception of Selecting Sequence

(a) Transmission of ACK

When a request to receive is issued and the receive buffer is empty.

(b) Transmission of NAK

When no request to receive is issued or the receive buffer is full.

(c) No Response

When transmission is impossible or the addresses do not match.

(2) Reception of Polling Sequence

(a) Transition to Text Transfer Phase

When a valid polling sequence is received.

(b) Transmission of EOT

When a valid polling sequence is received but no data is available.

(c) No Response

When transmission is impossible or the addresses do not match.

(3) Reception of EOT

Maintains the polling sequence receive state

5.3 TEXT TRANSFER PHASE

When a data link is established according to the polling sequence, both source and destination stations enter the text transfer phase. Text is transferred in this phase.

5.3.1 TEXT TRANSFER PHASE AT THE HOST. (HOST TO TERMINAL)

(1) Processing for Response at Source Station

(a) Reception of Positive Acknowledge

For positive acknowledge, an ACK is returned to acknowledge receipt of the data string.

After an acknowledge in response to text an EOT is sent, the data link will be released and the request to send will terminate normally.

(b) Reception of NAK

When the retry counter (L) is L times or less, text is resent. In the case of L+1 times, an EOT is sent and the data link is released. The request to send terminates abnormally. (refer to

pg 22, RETRY COUNTER)

(c) Reception of Invalid Response

Any response other than the above is ignored and the response wait state is maintained.

(2) Response Transmission Processing at Destination Station

(a) Reception of Normal Text

Sends acknowledge ACK.

(b) Reception of ENQ

Immediately re-sends the same text as the preceding one.

(c) Reception of Error text

Immediately sends a NAK and waits for re-transmission. Cases where text are regarded as erroneous are:

(1) Occurrence of BCC error.

(2) Reception of DLE.STX after receiving DLE.STX.

(3) Reception of ACK after receiving DLE.STX.

(4) Text length exceeded 50 bytes.

(d) Reception of EOT

The data link is released, processing returns to the data link establish phase, and the request to receive terminates normally.

5.3.2 TEXT TRANSFER PHASE AT THE TERMINAL. (TERMINAL TO HOST)

(1) Processing for response at Source Station.

(a) Reception of ACK

For positive acknowledge, an ACK is returned to acknowledge receipt of the data string.

After an acknowledge in response to text an EOT is sent, the data link will be released and the request to send will terminate normally.

(b) Positive Acknowledge

When the retry counter (L") is L" times or less, a request ENQ is sent. In the case of L"+1 times, the data link is released and the request to send terminates abnormally. If, however, this response is received for the request ENQ send due to a reception time-out, text will be resent on condition that the retry counter (L") is L" times or less

(c) Reception of NAK

When the retry counter (L) is L times or less, text is resent. In the case of L+1 times, the data link is released and the request to send terminates abnormally

(d) Reception of invalid response

Any response other than the above is ignored and the response wait state is maintained.

(2) Response Transmission Processing at Destination Station

See as (2) in Para. 5.3.1.

5.4 DATA LINK END PHASE

The text transfer phase terminates upon the transmission or reception of the EOT or retry count overflows. (secondary station only). The transmission control procedures phase returns to the data link establish phase and the data link is released.

5.5 DISCONNECTION PHASE

No processing.

6. ERROR CONTROL

Error control for text is done in accordance with the LRC (Longitudinal Redundancy Check) method. The exclusive OR (XOR) of the characters following DLE.STX to the ETX is represented by one byte.

DLE	STX	DLE.DLE	DLE	ETX	LRC

The DLE for control (including the DLE preceding ETX) is excluded from the arithmetic operation of LRC.

Calculation of BCC (Block Check Character) 1 byte

Set initial value to 0, now perform the exclusive OR from DLE.STX to DLE.ETX commands to obtain the BCC. If a DLE.DLE sequence is present in the TEXT, only the 1 DLE should be included in the BCC calculation.

Example:

21h	01h	BCC

00H	XOR	21H	BCC = 21H

21H	XOR	01H	BCC = 20H

7. RETRY COUNTER FUNCTION

7.1 HOST RETRY COUNTER FUNCTION

(1) Text send retry count (L)

(2) Link establish ENQ (polling/selecting) send retry count (L')

(3) Response request ENQ send retry count (L")

7.2 TERMINAL RETRY COUNTER FUNCTION

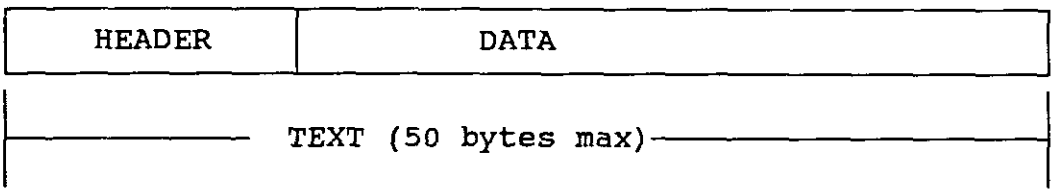
(1) Text send retry count (L)

(2) Response request ENQ send retry count (L")

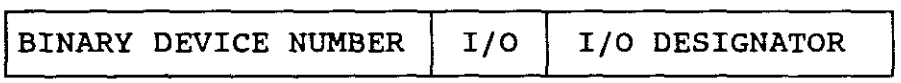
8. INTELLIGENT TERMINAL CONTROL CODES

This section deals with the actual control codes which are sent to the TERMINAL to control the various output functions, and to acquire the data from the various input devices.

The control sequence will contain the HEADER FIELD, as well as the DATA FIELD.



The 3 byte header field is composed of the following control codes:

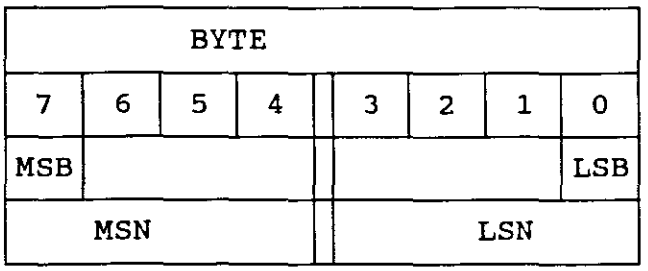


X = Binary Terminal Number

Helpful information:

(a)

A byte is divided into 2 nibbles, the LEAST SIGNIFICANT NIBBLE (LSN) and the MOST SIGNIFICANT NIBBLE (MSN). The byte could also be divided into 8 bits. BIT 0 is equal to the LEAST SIGNIFICANT BIT (LSB) and BIT 7 is equal to the MOST SIGNIFICANT BIT (MSB).



(b)

The following are the declarations which are used for the programming examples:

```
txmaster (txmsg, txtermnum, txlen)
          /* transmit routine for master      */
          /* returns: 0 if successful,        */
                      1 if comms error,       */
                      2 if invalid parameter */

char *txmsg;
          /* pointer to message               */

int txtermnum;
          /* terminal number */

int txlen;
          /* length of message to be transmitted*/


char msg[270];  /* mesage buffer */
```

8.1 OUTPUT DEVICE CONTROL CODES

8.1.1 16 X LED's

X	'O'	'L'	8 BYTE LED Control Field

8 Byte LED Control Field:

The 8 byte LED field is further divided into a 4 byte LED ON/OFF field and a 4 byte LED FLASH field. The LSN of each byte will control 4 LED's. Each of the 8 bytes must be OR'ed with 30H before transmitting this to the TERMINAL.

	LED ON/OFF				LED FLASH			
LED #	1-4	5-8	9-12	13-16	1-4	5-8	9-12	13-16
BYTE #	1	2	3	4	5	6	7	8

To turn on LED number 1, BIT0 in BYTE #1, must be set. Similarly to turn on LED number 4, BIT3 in BYTE #1 must be set. This will switch the LED on permanently. To flash LED #1, the associated BIT in the LED FLASH field must also be set. Therefore BIT0 in BYTE #5 must be set which will flash LED #1.

Programming example #1:

This will flash LED #1.

```
sprintf (msg,"OL10001000");
txmaster (msg, 1, 10);
```

Programming example #2:

This will flash LED #1 and switch on LED #8.

```
sprintf (msg,"OL18001000");
txmaster (msg, 1, 10);
```

8.1.2 TWO X LIQUID CRYSTAL DISPLAYS (LCD #1/#2)

X	'O'	'D'

0/1/2	32 BYTE DISPLAY STRING

Display options:

(a) '0' will display message on both displays.

(b) '1' will display message on LCD #1

(c) '2' will display message on LCD #2

(d) To clear the display the display transmit an e m p t y display sting.

(e) For space compression, place a 'HT' (horizontal tab = 0x09) followed by the number of spaces, which is OR'ed with 30H.

Programming example #1:

This will display a message on LCD #1.

```
sprintf(msg,"OD1**INTELLIGENT***POS     TERMINAL");
txmaster (msg, 1, 35);
```

Programming example #2

This will display a message on LCD #1 and LCD #2.

```
sprintf(msg,"OD0**INTELLIGENT***POS     TERMINAL");
```

```
txmaster (msg, 1, 35);
```

Programming example #3:

This will clear LCD #2.

```
sprintf(msg,"OD2");
txmaster (msg, 1, 3);
```

Programming example #4:

This is to illustrate how the string should be formatted for space compression. The 'HT' precedes the number of spaces to be inserted. In the following example 11 spaces had to be inserted, therefore 30H OR'ed with 11H = 41H.

```
sprintf(msg,"HELLO%c%cWORLD!",0x09,0x41);
txmaster (msg, 1, 3);
```

8.1.3 SOUND THE BUZZER

The buzzer has a fixed frequency and also a fixed duration. When the control code is sent to the TERMINAL it will therefore sound the buzzer for a fixed duration.

X	'O'	'B'

Programming example #1:

This will sound the buzzer.

```
sprintf(msg,"OB");
txmaster (msg, 1, 2);
```

8.1.4 OPEN CASH DRAWER #1 OR #2

X	'O'	'R'	1/2

Drawer options:

(a) '1' will open cash drawer #1.

(b) '2' will open cash drawer #2.

Programming example #1:

This will open cash drawer #2.

```
txmaster ("OR2", 1, 3);
```

8.1.5 ENABLE OR DISABLE MAGNETIC CARD READER

X	'O'	'C'	0/1

Magnetic Card Reader options:

(a) '0' will disable the magnetic card reader.

(b) '1' will enable the magnetic card reader.

Programming example #1:

This will enable the magnetic card reader.

```
sprintf(msg,"OC1");
txmaster (msg, 1, 3);
```

Programming example #2:

This will disable the magnetic card reader.

```
sprintf(msg,"OC0");
txmaster (msg, 1, 3);
```

8.1.6 HOST ENQUIRY FOR LOCK, DRAWER, RECEIPT & SWITCH STATUS

X	'O'	'E'

The HOST will send the above message to the TERMINAL to enquire as to the status of the input devices. This will normally be executed at power-up, when the application program will need to know what key is in the lock, as well as the status of the various input devices. Please refer to the section on INPUT DEVICES to see the response message sent from the TERMINAL to the HOST.

8.1.7 TWO RS232 SERIAL PORTS

X	'O'	'S'

1/2	#	C/D	40 BYTE STRING

The serial port facility which is provided on the TERMINAL is meant for very basic interfacing to serial devices, such as the 2x16 display. The main purpose is to transmit a string of data or receive a string of data with a certain amount of handshaking taking place via the RTS and CTS lines. This interface is not meant for any elaborate protocols such as that required by the SW1 serial printer. It is however completely adequate for serial printers that require the data to be transmitted with a minimum of handshaking. A typical printer would be the M-290 slip printer, which buffers all the incoming data.

The TERMINAL can receive either a DATA string or a COMMAND string. The COMMAND string is used to change the bit format setting of the serial port. The default bit format of both serial ports is the following:

BAUD RATE: 9600
PARITY: none
DATA BITS: 8
STOP BITS: 1

Serial port options:

(a) '1' transmit command or data to COM1.

(b) '2' transmit command or data to COM2.

(c) # is the number of bytes in the 'DATA STRING' + 2 bytes for '1/2' and 'C/D'. The number of bytes has to be represented in binary notation.

eg

DATA string = 20 bytes + 2 bytes, therefore # = 16H. (16H = 22 decimal)

(d) 'D' will inform the TERMINAL that the string is a DATA string and should be processed appropriately. This is used to transmit data to COM1 or COM2.

(e) 'C' will inform the TERMINAL that the string is a COMMAND string and should be processed appropriately. This is used to change the bit format of COM1 or COM2

The COMMAND string has to be in the following format in order to change the bit format for COM1 or COM2.

X	'O'	'S'

1/2	#	C/D	B	D	S	P

COMMAND string options:

(a) B is the baud rate, which have to be represented in binary notation. The baud rate options are the following:

B = 48H => 4800 Baud

B = 96H => 9600 Baud

B = 19H => 19.2 KBaud

B = 38H => 38.4 KBaud

(b) D is the number of data bits, which have to be represented in binary notation. The number of data bit options are the following:

D = 07H => 7 data bits.

D = 08H => 8 data bits.

(c) S is the number of stop bits which have to be represented in binary notation. The number of stop bit options are the following:

S = 01H => 1 stop bit.

S = 02H => 2 stop bits.

(d) P specifies whether parity will be ODD, EVEN or NO parity. The parity options are as follows:

P = 'N' => no parity.

P = 'O' => odd parity.

P = 'E' => even parity.

Programming example #1:

The following command will change the bit format to suit the M290 slip printer on COM1.

```
sprintf(msg,"OS1%cC%c%c%cE",6,0x96,0x07,0x01);
/* 9600, 7, 1, E */
txmaster (msg, 1, 9);
```

Programming example #2:

The following data string will be transmitted via COM1 to the M290 slip printer.

```
sprintf(msg,"OS1%cDTX FROM HOST TO TERM, TO  COM1%c%c%c
",34,0x0d,0x0a,0x11); /* last 3 bytes = CR, LF
                                pinch roller up       */
txmaster (msg, 1, 37);
```

Programming example #3:

The following data string will be transmitted via COM2 to the 2 x 16 character display.

```
sprintf(msg,"OS2%cD%c0REMOVE SLIP FROMFLAT_BED
PRINTER2%c",38,0x02,0x03);
/* 0x02=STX, 0x03=ETX required by the 2x16 display */
txmaster (msg, 1, 41);
```

This concludes the commands that can be transmitted from the HOST to the TERMINAL.

8.2 INPUT DEVICE CONTROL CODES

8.2.1 4 BIT CONTROL LOCK

X	'I'	'L'

1 BYTE LOCK STATUS

The following is a list of some of the binary values for some of the keys:

N key = 06H

M key = 07H

L key = 08H

K key = 09H

A key = 0EH

8.2.2 MAGNETIC CARD READER

X	'I'	'C'

01H/FFH	37 BYTE CARD DATA

If the card is read successfully an 01H will be returned followed by 37 bytes of card data. If an error is detected when the card was read, FFH will be returned without the 37 bytes of card data.

Magnetic card reader options:

(a) 01H signifies that the card was read successfully.

(b) FFH signifies that an error occurred when the card was read and that no card data will be returned.

8.2.3 DRAWER STATUS

X	'I'	'R'

1/2	00H/01H

Drawer status options:

(a) '1' specifies drawer #1.

(b) '2' specifies drawer #2.

(c) 00H specifies that the drawer is closed.

(d) 01H specifies that the drawer is open.

8.2.4 RECEIPT STATUS

Receipt status options:

(a) 00H specifies that the receipt switch is closed.

(b) 01H specifies that the receipt switch is open.

8.2.5 KEYBOARD OR SCANNER DATA

When entering data from the keyboard or from the scanners, the sequence in which the data was entered must be maintained. This is the reason that the keyboard data and scanner data were placed in the same buffer. The TERMINAL can buffer 25 key depressions or 2 scanned item numbers.

KEYBOARD ENTRY FORMAT

The key value will be the binary representation of the position of the key on the keyboard.

01	02	03	04	05	06	07	08	09
10	11	12	13	14	15	16	17	18
19	20	21	22	23	24	25	26	27
28	29	30	31	32	33	34	35	36
37	38	39	40	41	42	43	44	45
46	47	48	49	50	51	52	53	54
55	56	57	58	59	60	61	62	63
64	65	66	67	68	69	70	71	72

Table 1-3. Keyboard Matrix

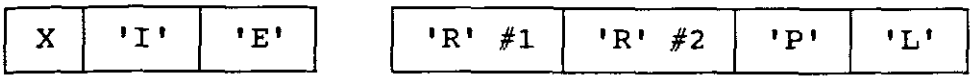
SCANNER ENTRY FORMAT

X	'I'	'K'

FFH	#	06H, 13 BYTE NUMBER

Scanner data is identified by FFH. # is the number of bytes in its binary representation of the number of bytes read from the scanned label A total of 14 bytes is therefore read from the label. The first byte is normally the EAN LABEL ID, and is normally 06H. This is followed by a 13 byte number

8.2.6 RESPONSE TO HOST ENQUIRY



The above string is the response which the TERMINAL will return when an ENQUIRY is sent from the HOST.

Enquiry options:

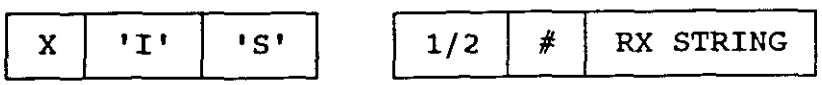
(a) 'R' #1 is the status of drawer number 1, and this could be 00H = closed, or 01H = open.

(b) 'R' #2 is the status of drawer number 2, and this could be 00H = closed, or 01H = open.

(c) 'P' is the receipt switch status, and this could be 00H = closed, or 01H = open.

(d) 'L' is the binary representation of the lock status.

8.2.7 TWO X RS232 SERIAL PORTS



The incoming serial data is treated on a byte basis, and

the TERMINAL does not distinguish between the start of a message and the end of a message, but treats each byte as a received character. This is then transmitted to the HOST computer. The application programmer should therefore ensure that complete messages are received, as the TERMINAL might transmit the message in sections.

Serial port options:

(a) '1' received data on COM1.

(b) '2' received data on COM2.

(c) # is the number of bytes in the 'RX STRING'. The number of bytes is represented in binary notation.

9. PROGRAMMING EXAMPLE

```
/* program to test intelligent terminal    05-90 */

#include <stdio.h>
#include <defn.h>


long msg_count;
char lock[20] = {
    '?','?', '?', '?', '?', '?', 'N', 'M', 'L', 'K', '?', '?',
    '?', '?', 'A', ' '
};

main()
{
    extern  long    msg_count;
    int     i, c, numch;
    char    msg[270];

    msg_count = 0;
    init_comx();        /* initialize com port variables  */
    scr_printf("(1)DRAW1  (2)DRAW2  (3)BUZZ  (4)LCD
          (5)CRD_ON  (6)CRD_OFF\n\n");
    scr_printf("(7)LED 8  (8)LED 1  (9)COM1  (0)COM2
          (A)INIT_COM1(96,7,E,1)\n");
    if (!initmcomm (1)) {
```

```
        scr_clear();
        scr_printf ("COMMS CHANNEL ERROR");
        exit(1);
    }

    while (TRUE) {
       intkbd();   /* read rx data from intelligent kyb  */
       if (scr_poll() != -1) {
            switch (toupper(scr_getc())) {
                case 0x1b:
                            scr_clear();
                            exit(0);        /* ESC */

                case '1':   txmaster ("OR1", 1, 3);
                            break;

                case '2':   txmaster ("OR2", 1, 3);
                            break;

                case '3':   sprintf(msg,"OB");
                            txmaster (msg, 1, 2);
                            break;

                case '4':   sprintf(msg,"OD1**INTELLIGENT***POS
    TERMINAL");
                            txmaster (msg, 1, 35);
                            msg_count = 0;
```

```
                    break;

            case '5':   sprintf(msg,"OC1");
                        txmaster (msg, 1, 3);
                        break;

            case '6':   sprintf(msg,"OC0");
                        txmaster (msg, 1, 3);
                        break;

            case '7':   sprintf(msg,"OL10001000");
                        txmaster (msg, 1, 10);
                        break;

            case '8':   sprintf(msg,"OL18001000");
                        txmaster (msg, 1, 10);
                        break;

            case '9':
sprintf(msg,"OS1%cDTX FROM HOST TO TERM, TO
COM1%c%c%c",34,0x0d,0x0a,0x11);
                        txmaster (msg, 1, 37);

sprintf(msg,"OS2%cD%c0REMOVE SLIP FROMFLAT_BED
PRINTER2%c",38,0x02,0x03);
                        txmaster (msg, 1, 41);
                        break;
```

```
                case '0':
sprintf(msg,"OS2%cD%c0   TX FROM HOST TO TERM, TO
COM2%c",37,0x02,0x03);
                                txmaster (msg, 1, 40);
                                break;

                case 'A':

sprintf(msg,"OS1%cC%c%c%cE",6,0x96,0x07,0x01);
     /* 9600, 7, 1, E  M290 slip printer */
                                txmaster (msg, 1, 9);
                                break;
                }
            }
        }
}
```

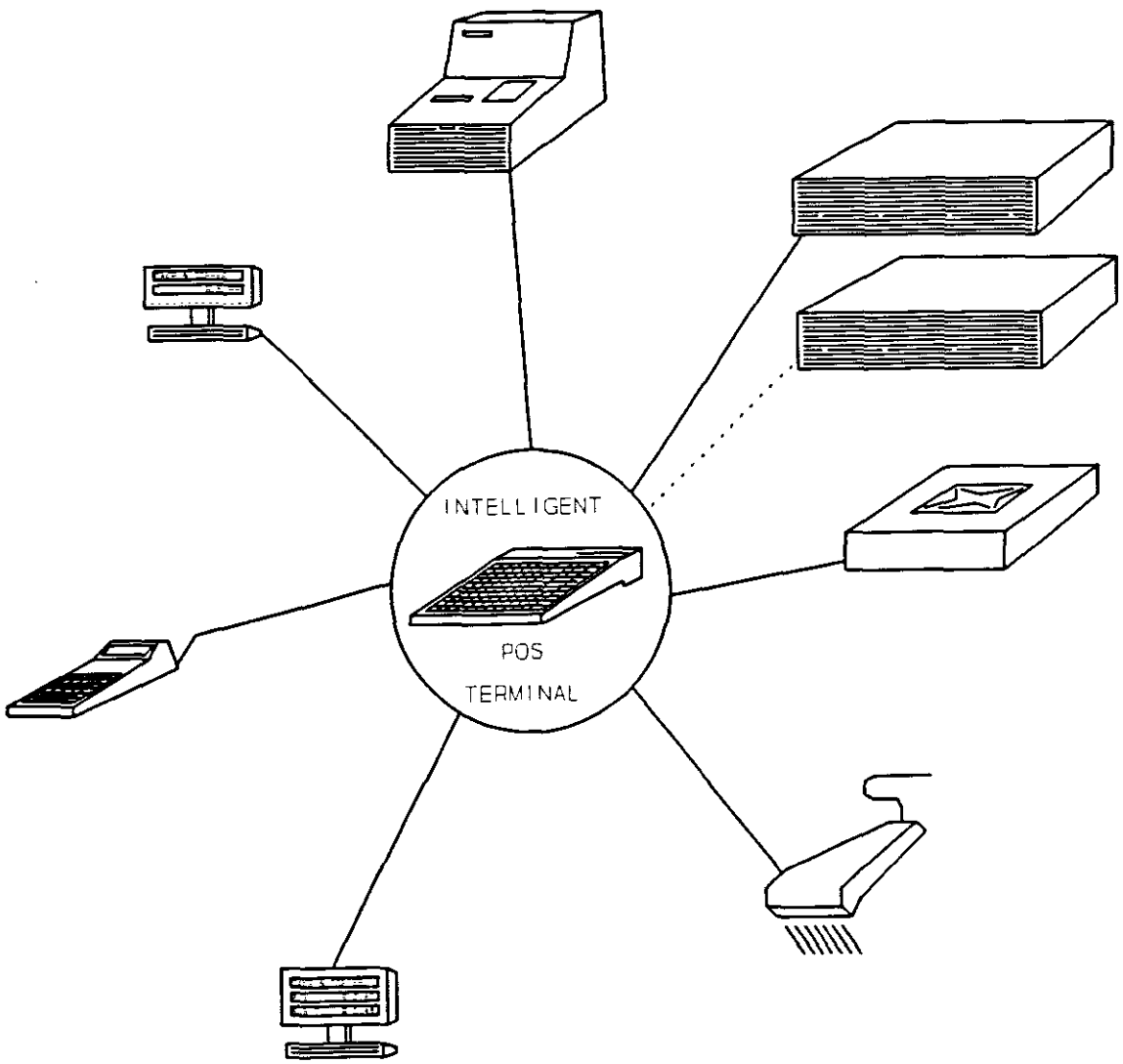
```
intkbd()    /* check intelligent  keyboard. return -1 if no key,
else raw value */
{
    int     i, c, numch;
    unsigned char   mod[270];
    char    msg[100];

   c = rxmaster (mod, 1, &numch);
   if (c == 0) {
        scr_curs (12,0);
        scr_puts ("                                              ");
        scr_curs (12,0);
        switch (mod[1]) {

            case 'C':
                if (mod[2] == 0xff){
                    sprintf(msg,"OD1PLEASE RE-SWIPE ***YOUR
                         CARD***");
                    txmaster (msg, 1, 35);
                }
                else if (mod[2] == 0x01){
                    printf ("CARD# ");
                    for (i = 3; i <= 20; i++)
                        printf ("%c", mod[i]);
                    sprintf(msg,"OD1YOUR CARD HAS      BEEN
                         READ");
                    txmaster (msg, 1, 35);
```

```
            sprintf(msg,"OB");
            txmaster (msg, 1, 2);
        }
        break;

    case 'P':
        printf ("RECEIPT ");
        if (mod[2] == 0x0)
            printf ("OFF");
        else if (mod[2] == 0x01)
            printf ("ON");
        break;

    case 'R':
        printf ("DRAW #%c ",mod[2]);
        if (mod[3] == 0x0)
            printf ("CLOSED");
        else if (mod[3] == 0x01)
            printf ("OPEN");
        break;

    case 'L':
        i = mod[2] >> 4;
        printf ("LOCK %c",lock[i]);
        break;
```

```
            case 'K':
                if (mod[2] == 0xff){
                    printf ("SCANNER DATA ");
                    for (i = 5; i <= mod[3]+3; i++)
                        printf ("%c", mod[i]);
                }
                else (printf ("KEY# %d", mod[2]));
                break;
            case 'S':
                printf ("COM%c ", mod[2]);
                for (i = 4; i <= mod[3]+3; i++)
                    printf ("%c", mod[i]);
                break;

            default:
                printf ("UNKNOWN RX STRING %s",mod);
                break;
        }
    }
    return (-1);
}
```

INTELLIGENT POINT OF SALE TERMINAL

HARDWARE DESCRIPTION



CHAPTER 2

1. SYSTEM DESCRIPTION

1.1 SYSTEM OVERVIEW

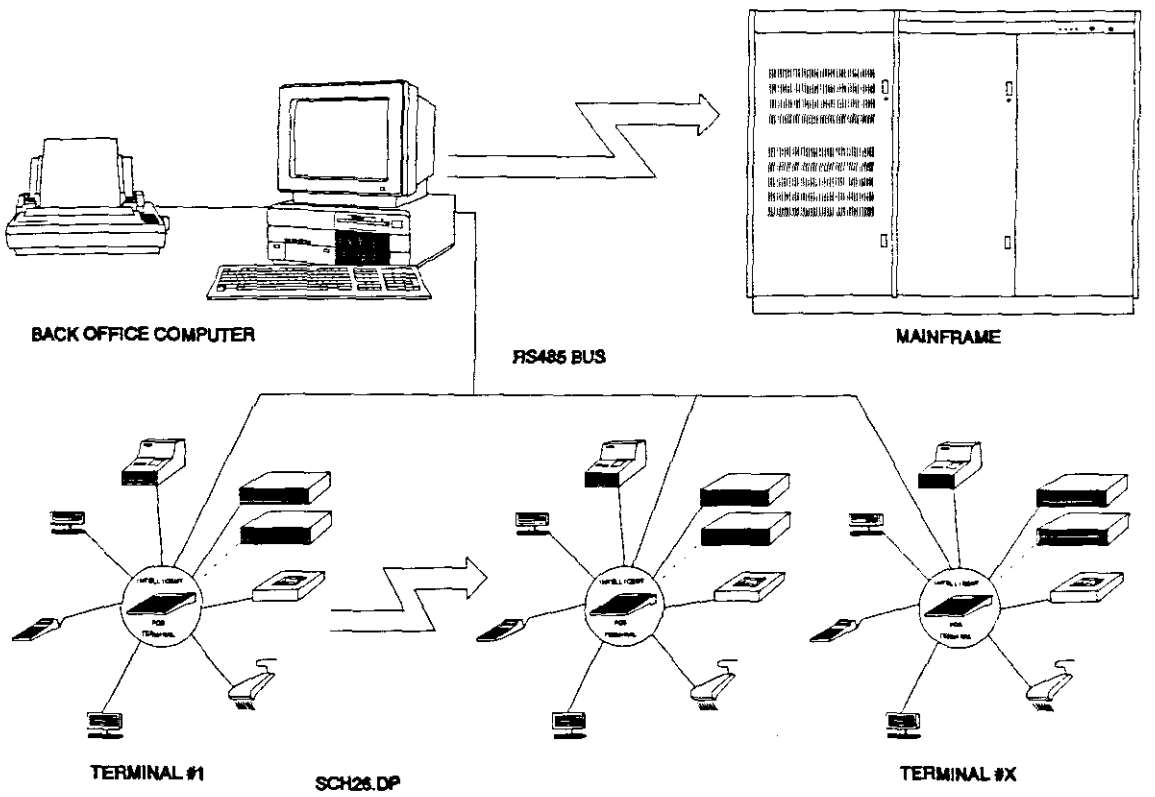


Figure 2-1. System Configuration

The system had to have the capability of communicating to a maximum of 255 terminals. Each terminal had to be individually addressable. The HOST computer, which would be situated in the back office, would have total control of the communications line. The protocol is based on the Bourroughs 7 bit ASCII protocol, which has a POLL - ACKNOWLEDGE structure. Each terminal has a

unique terminal number which prevents conflict on the communications bus, which might be caused by two terminals simultaneously transmitting data onto the bus.

The HOST will "POLL" a terminal, if it has any data available, the data will be transmitted to the HOST. If the HOST has any data for the terminal, the HOST will "SELECT" a particular terminal and the data will be transmitted to the terminal. A differential communications bus, utilising the RS485 electrical standard is used to inter-connect the terminals to the HOST. This produces a multi-drop connection, enabling a terminal to be easily introduce, or removed from the bus.

This external communications bus is called ANET, and has a data transfer rate of 38.4 KBuad. The communications bus is used to transmit the latest item data at "store open" to the terminals, and to consolidate the totals from the various terminals at "store close". The HOST can now generate all the necessary reports, and the consolidated data can now be transmitted to a remote mainframe via a modem.

1.2 TERMINAL OVERVIEW

Each terminal consists of the following hardware devices, which can be seen graphically by referring to Figure 2-2:

SW1 receipt/journal printer

Bi-directional magnetic card reader

2 X 16, 3 X 16 and/or 2 x 20 Vacuum Fluorescent Displays

2 X 16 Liquid Crystal Displays

2 x Cash Drawers

Handscanner and/or desktop scanner

INTELLIGENT TERMINAL, which forms the remainder of this hardware description

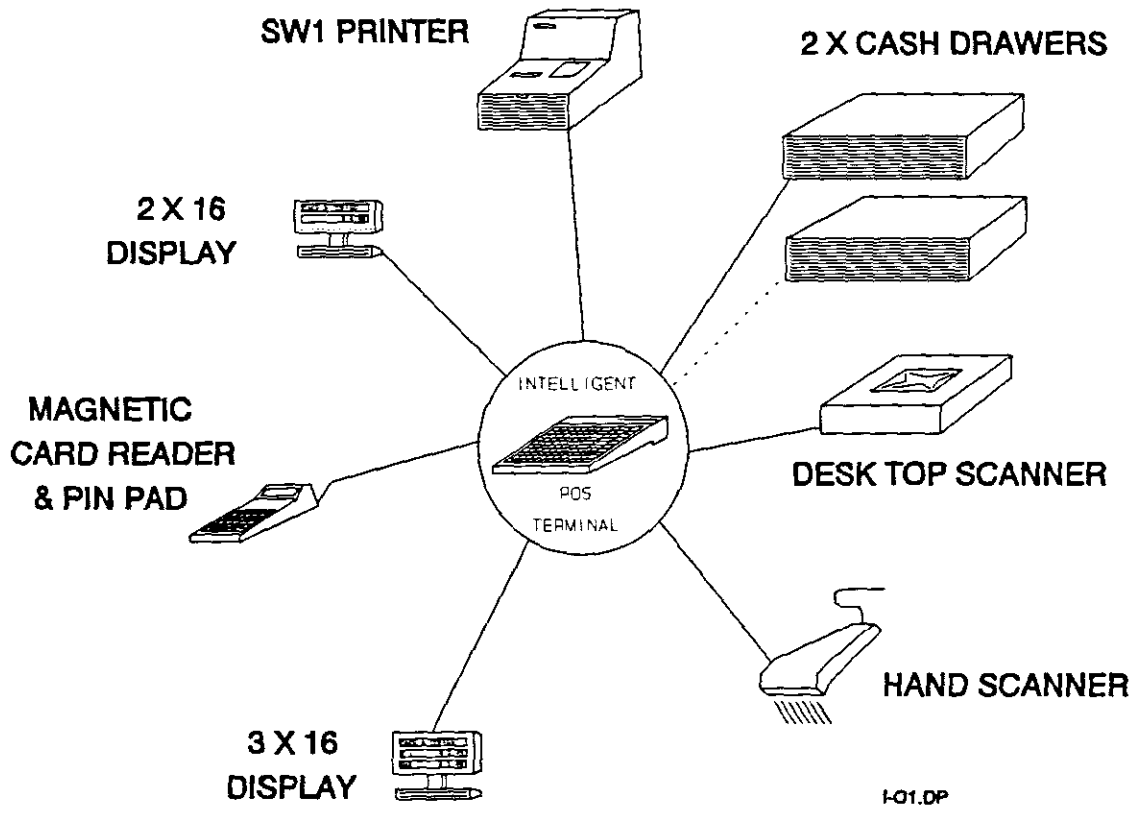


Figure 2-2. Terminal Description

1.3 DETAILED TERMINAL DESCRIPTION

1.3.1 PROCESSING SECTION

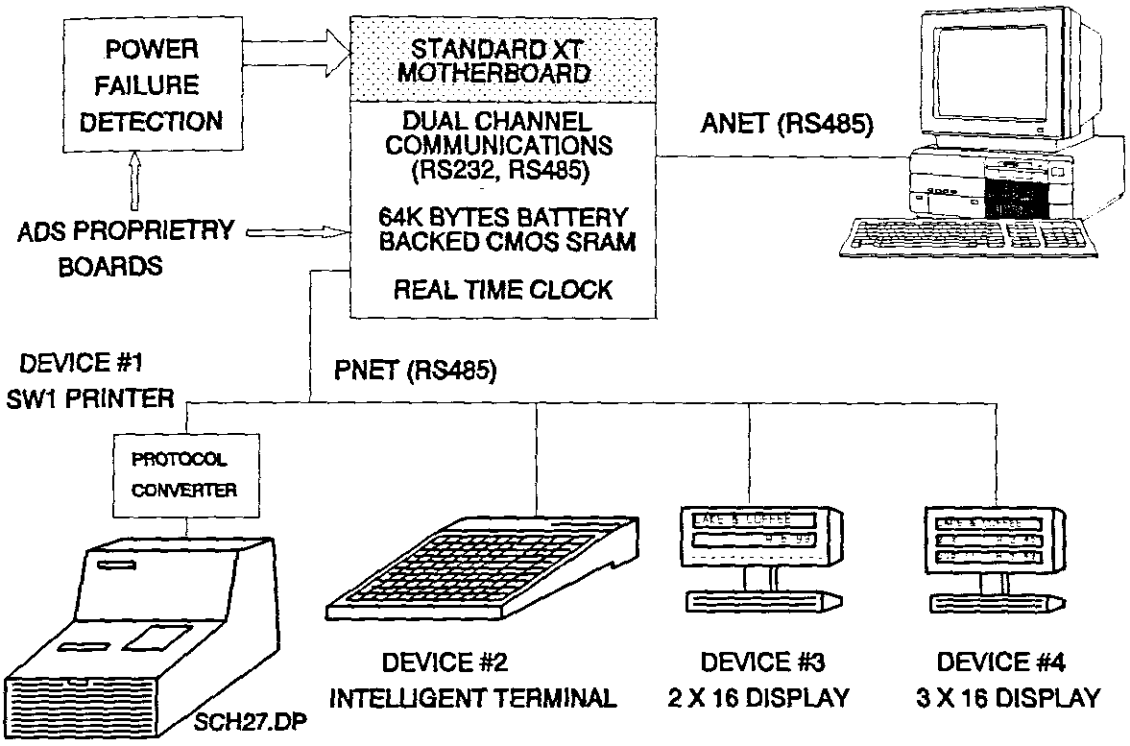


Figure 2-3. Detailed Terminal Description

Each terminal requires a main processing section which will control the various Input / Output devices, communications to the HOST and execute the required application program. This is achieved by using a standard XT motherboard in addition to two proprietary Ankerdata cards.

The power failure detection circuitry is situated in the power supply section. This card monitors the 220 VAC and +5V lines. If the mains drops below 180 VAC, or the +5V exceeds a certain predefined window, a Power Failure Interrupt is generated, and

the CPU will save the current variables to the battery backed CMOS. This ensures that all totals and current sale status is maintained when power is returned to the terminal. This feature is vital for the operation of the terminal.

The Ankerdata DMA card, is an 8 bit expansion card which plugs directly into any of the XT expansion slots. The card consists of the following:

- 64K Bytes battery backed CMOS Static RAM
- Dual Channel Communications (RS232/RS485)
- Real Time Clock
- Programmable Peripheral Interface (PPI)
- Selectable Terminal Address
- 4 Status LED's

One communications port is used for communicating to the HOST, which would be situated in the back office. This communications bus is called ANET. The second communications port is used for the internal operation of the terminal, and is called the Peripheral Network (PNET). This bus enables the various intelligent devices to communicate to the terminal.

1.3.2 COMMUNICATIONS SECTION

The communications bus, ANET, which was previously described was extended one level down, to be incorporated into the internal

operation of the terminal. This bus was called the Peripheral Network, or PNET, which has a data transfer rate of 19.2 KBaud. The protocol is a modified version of the Bourroughs 7 bit ASCII protocol. The main difference is that a more efficient binary format is used on PNET, but the basic POLL - ACKNOWLEDGE structure is still maintained. A more detailed description can be found in the "SOFTWARE DESCRIPTION" section.

1.3.3 DEVICE SECTION

The long term project goal was to design a completely modular system. The greatest advantage of a modular system is the ability to interchange the various modules, thus allowing a customer to "mix 'n match" to their requirements. The devices should be transparent to the terminal, which means that the application software does not have to change to accommodate every system configuration. Another advantage is that a new cabinet does not have to be designed to accommodate a new device, which is the case in an integrated system.

This meant that a distributed processing network had to be implemented, incorporating a certain amount of intelligence in each device. The great advantage of using a distributed processing network, is that the main processor is not bogged down with menial input / output tasks, and therefore has more time available for higher level processing. The XT motherboard, in

conjunction with the Ankerdata DMA card is therefore the HOST for all the Peripheral Devices. An important point to remember is that there are now two HOSTS controlling a single terminal / back office computer configuration. The main HOST controls ANET, which connects all the terminals to the back office. The secondary HOST controls PNET which is internal to the terminal, and connects all the devices to the terminal.

The vacuum fluorescent displays are controlled via an 8751. The INTELLIGENT TERMINAL and the protocol converter are controlled via an 8032.

The main advantage of the protocol converter is that any standard device can be used. The SW1 is an "off the shelf" printer with a standard RS232 interface. The converter intercepts the data from the differential bus, PNET, and then transmits the data via the protocol converters RS232 port to the printer. The same converter is used to connect an electronic checkout scale, thermal printer, kitchen printer and cheque reader to the system. Due to such a great deal of flexibility, any device with a standard RS232 serial port can be introduced into the system by merely changing the application program of the protocol converter.

2. INTELLIGENT TERMINAL SPECIFICATIONS

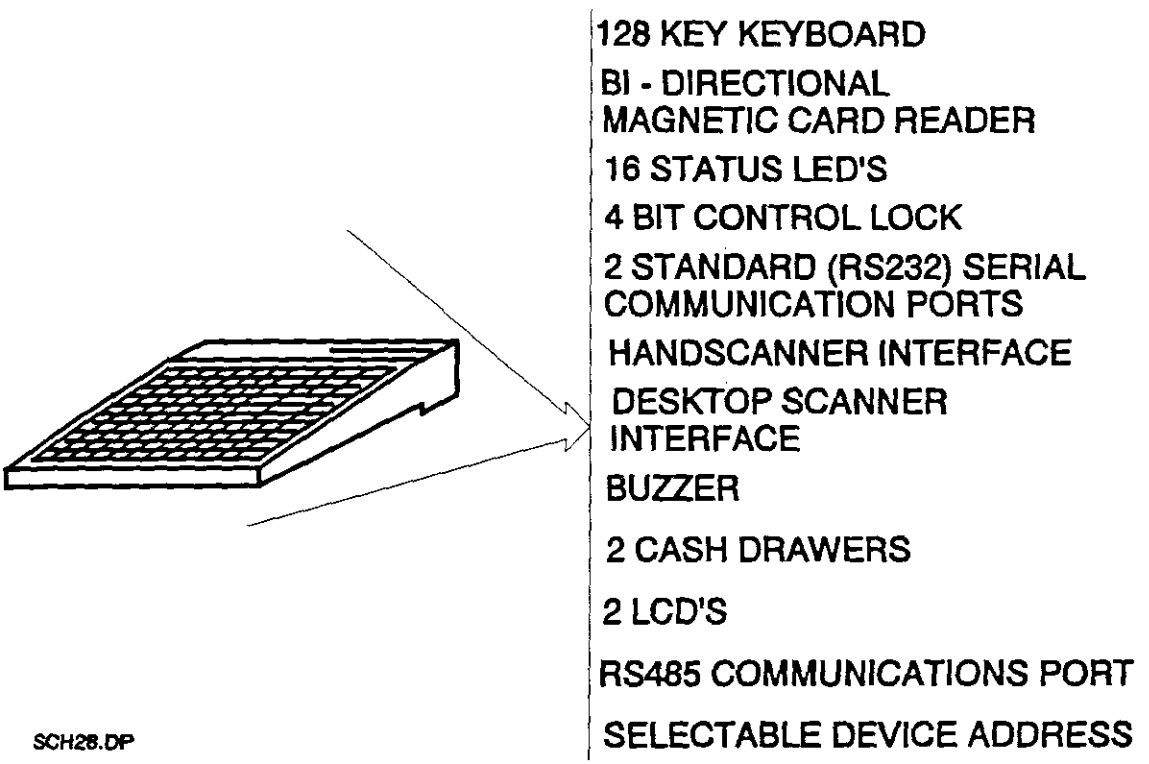


Figure 2-4. INTELLIGENT TERMINAL Specifications

The INTELLIGENT TERMINAL had to have the capability of interfacing to the following devices:

128 Key keyboard

Two, 2 line by 16 character Liquid Crystal Displays with LED backlighting

4 Bit control lock

Optically Coupled Serial Interface (OCIA) for a desktop scanner

Optically Coupled Serial Interface (OCIA) for a handscanner

Two standard serial ports (RS232)

One bi-directional magnetic card reader interface

16 Status LED's

Selectable terminal address

Two cash drawer interfaces

Buzzer

Receipt status switch

Communications via a multidrop type system (RS485), or a point to point system (RS232). This meant that a number of the INTELLIGENT TERMINALS could be connected to a single HOST, or that other devices could share the same communications bus.

3. MICROCONTROLLER DESCRIPTION

3.1 DATA AND PROGRAM MEMORY ORGANISATION

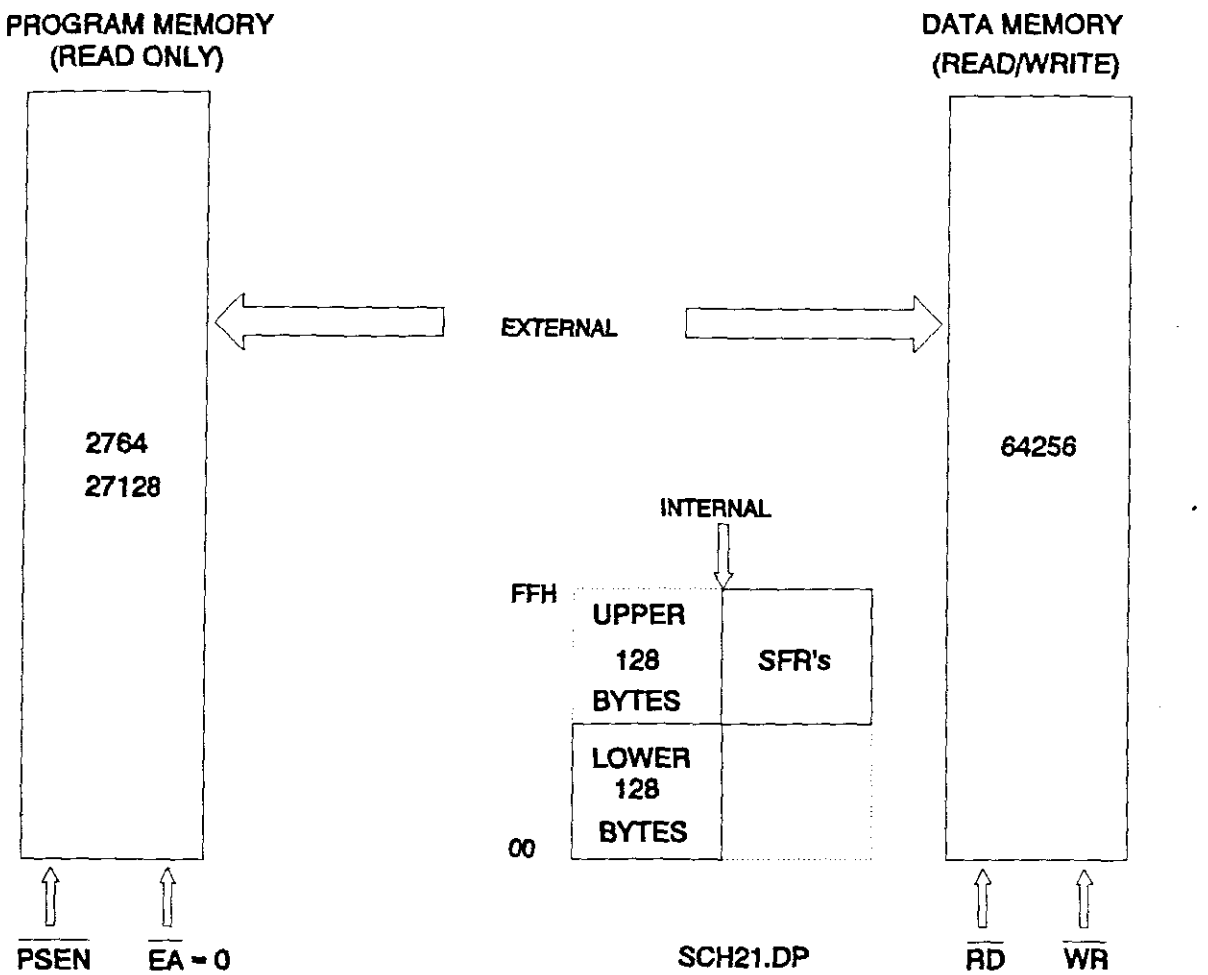


Figure 2-5. Memory Organisation

All MCS-51 devices have separate address spaces for Program and Data Memory, as shown in Figure 2-5. The logical separation of Program and Data Memory allows the Data Memory to be accessed by 8-bit addresses, which can be more quickly stored and manipulated by an 8-bit CPU.

A major limitation of the 8051, is the limited amount of internal data memory. The problem is magnified by the rather limited number of instructions available for external data memory accesses. Due to this limitation all single byte variables are placed in the internal data memory section, and all the buffers, such as the display buffer, is placed in external data memory.

The lower 128 bytes of internal data memory can be directly accessed, but the upper 128 bytes has to be indirectly accessed. The upper 128 bytes are insufficient for the INTELLIGENT TERMINAL requirements, and all the buffers larger than 3 bytes are placed in external data memory. The problem of external data memory accesses arises typically when data has to be read from external data memory, then written to external data memory. This implies that the Data Pointer (DPTR), which is the only 16 bit register available, has to be saved each time during the read, write routine. This makes the routine very stack intensive.

The problem is greatly compounded when accessing the I/O devices, which are all memory mapped, and then accessing the external data memory, where the various buffers are maintained. There is however a simple method of addressing the first 256 bytes of external memory, using an 8 bit register, either R0 or R1. To utilise this method, one has to ensure that PORT 2, which forms the high order address bus is set to zero, and that the required low order address is set via R0 or R1. The ~RD and ~WR signals are used to address the external data memory.

Program Memory can only be read, not written to. There can be up to 64K bytes of program memory. The read strobe for external Program Memory is the ~PSEN (Program Store Enable). The External Access Enable (~EA) pin is strapped to Vss, so that all program fetches are directed to external ROM. The ROMless parts must have this pin externally strapped to Vss to enable them to execute properly.

Data Memory occupies a separate address space from Program Memory. Up to 64K bytes of external RAM can be addressed in external Data Memory space. The CPU generates read and write signals, ~RD and ~WR, as needed during external Data Memory accesses.

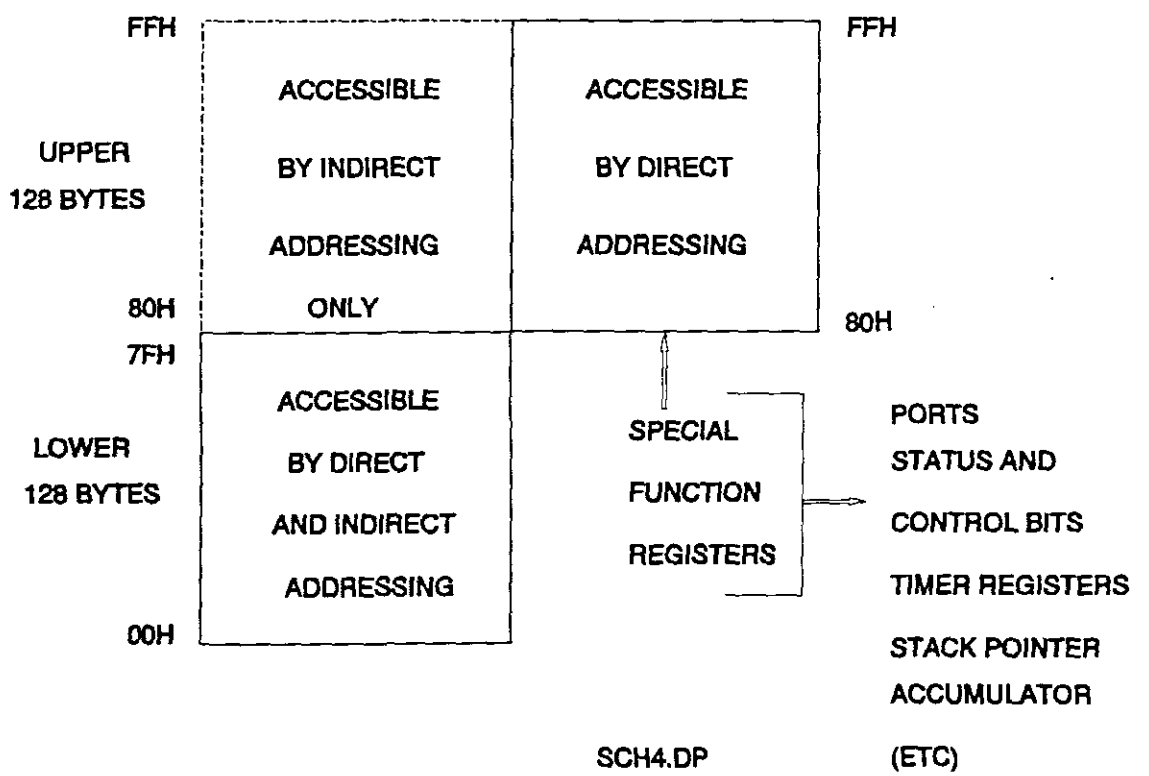


Figure 2-6. Internal Data Memory

3.2 INTERNAL DATA MEMORY ORGANISATION

The lowest 32 bytes are grouped into 4 banks of 8 registers. After a reset the stack pointer is initialized to point above the first register bank. This is changed via the initialization code to point above the second register bank. The two register banks are maintained due to the ease with which the banks can be switched. A particular bank is selected by two bits in the Program Status Word (PSW). The register banks are extremely useful when used in conjunction with interrupt routines. Instead of saving all the registers, a simple bank switch instruction can be used.

The next 16 bytes above the register banks form a block of bit-addressable memory space. The 128 bits in this area can be directly addressed by specific bit instructions. These instructions are extremely useful, as there is direct control over a hardware (port pin) or software (flag) bit. It is not necessary first to mask a particular bit, then to perform a test on the bit, as the instructions are geared specifically for bit manipulation. A total of 28 bits is defined, which are mainly used for flags. As mentioned earlier, the stack pointer is initialized to 0FH. When the stack is used, the pointer is firstly incremented, which implies that the stack starts at 10H, and then grows upwards. A total of 60 bytes has been allocated to the stack. The remainder of internal memory is allocated to single byte variables such as counters.

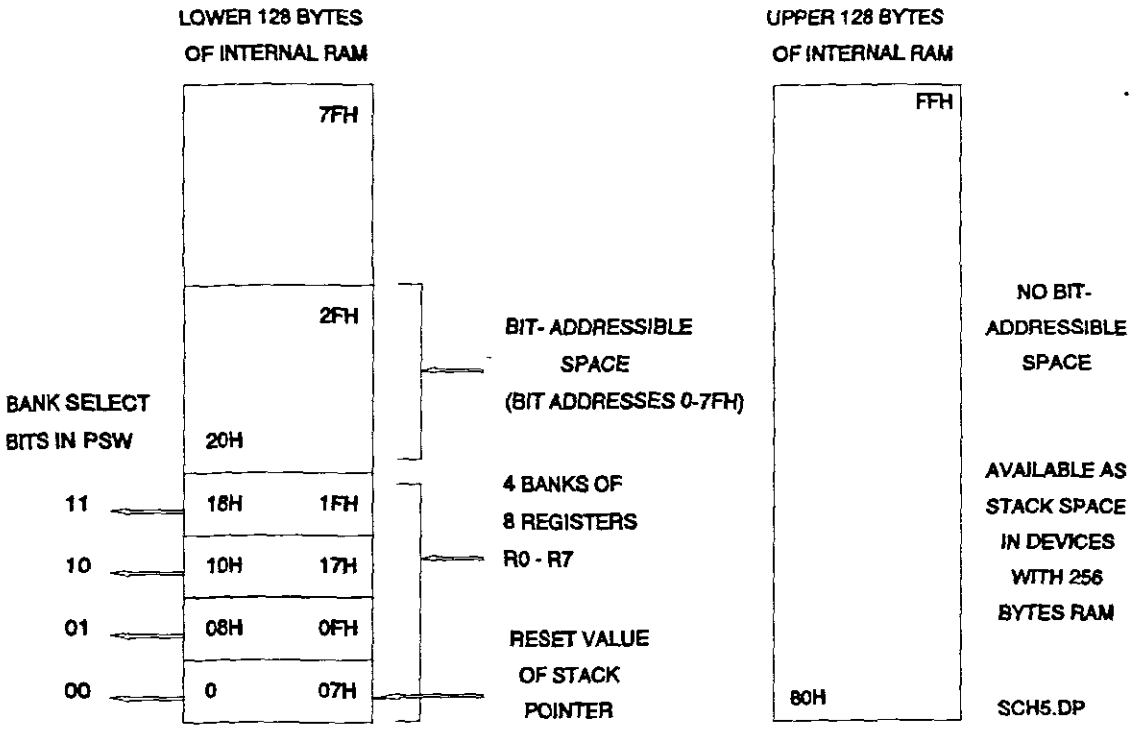


Figure 2-7. Internal RAM Usage

4. DESIGN IMPLEMENTATION

4.1 PROCESSING SECTION

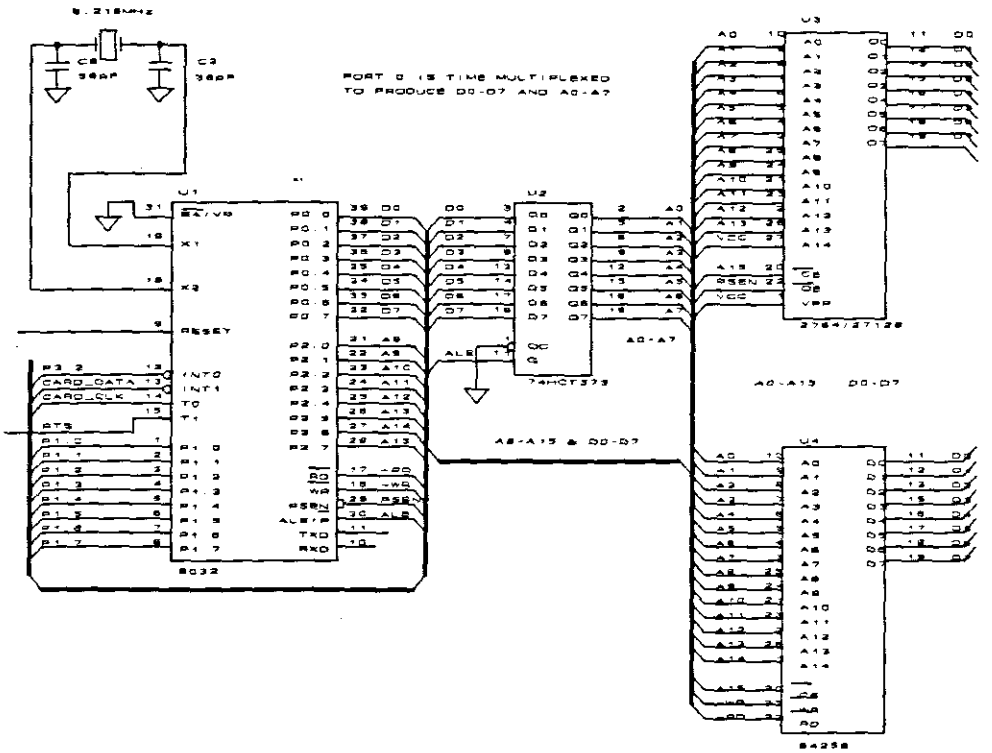


Figure 2-8. Single Board Computer

It was decided to use the 12MHz version of the 8032 microcontroller, because of the following features:

- ROMless
- 256 Bytes Internal RAM
- 4 I/O Ports
- Three 16-bit timers
- UART
- 8 Interrupt sources / 6 Interrupt vectors

From the specification it is quite obvious that the controller would be quite busy communicating with all the devices. It was also vitally important that there be a minimal delay from the time that an event, such as the start of a keystroke, till the time that the HOST processed the keystroke. Due to this fact, as well as not knowing whether the controller would cope with the work load, the processing section was sandwiched to the rest of the I/O circuitry. This meant that if the capacity of the controller was reached it would just mean plugging on a more powerful processing sub-section.

All the code was written in assembler, using Intel's ASM51 and RL51. After the first prototype was built, it was found that the controller was more than adequate, and coped with the tasks it was allotted.

It was decided to keep the Program Memory and Data Memory apart, which meant that a maximum of 64K bytes of Program and Data Memory space was available. The requirements of the INTELLIGENT TERMINAL meant that an 8K bytes Program space would be sufficient, however allowance was made for a 16K bytes EPROM just as a precaution. A 32K byte static RAM was used for the Data space, where the remaining upper 32K bytes of Data space would be allocated to I/O. This meant that the I/O would be Memory Mapped.

As the 8032 has a multiplexed bus, it was necessary to

demultiplex the bus. For this purpose the '373 is included. Port 0 of the 8032 emits the lower eight address lines, as well as the eight data lines. During an external memory access the lower address is first put out, and latched to the outputs of the '373 with the Address Latch Enable (ALE) signal. On the next cycle the data is placed on the data bus. Port 2 emits the upper eight address lines, thus forming a 16 bit address.

4.2 COMMUNICATIONS INTERFACE SECTION

As mentioned in the specification the terminal had to be capable of a multidrop type communications system. It was decided to use the RS485 standard. The DS3695 line driver is the ideal chip for this purpose. The Output Enable (OE) of the chip is driven from a port pin. A problem however became apparent during power-up of the controller. During the period that the controller is reset, the port pins are all high, this meant that the output would be enabled and that there was a possibility of contention on the communications line. It was therefore necessary to include an inverter, in the form of a VN10KM FET, between the port pin and the DS3695.

It was also necessary to include the standard RS232 line drivers, in the event that the terminal had to be connected to the serial port of a standard Personal Computer (PC). The MAX232 essentially replaced two chips, namely DS1488 and DS1489 line drivers. The

beauty of the MAX232 is that it also operates off a +5V supply. The RS232 standard specifies that the signal swings between +12V and -12V, this meant that a +12V and -12V supply has to be generated. The MAX232 takes care of this with its internal charge pump. JP1 selects between RS232 and RS485.

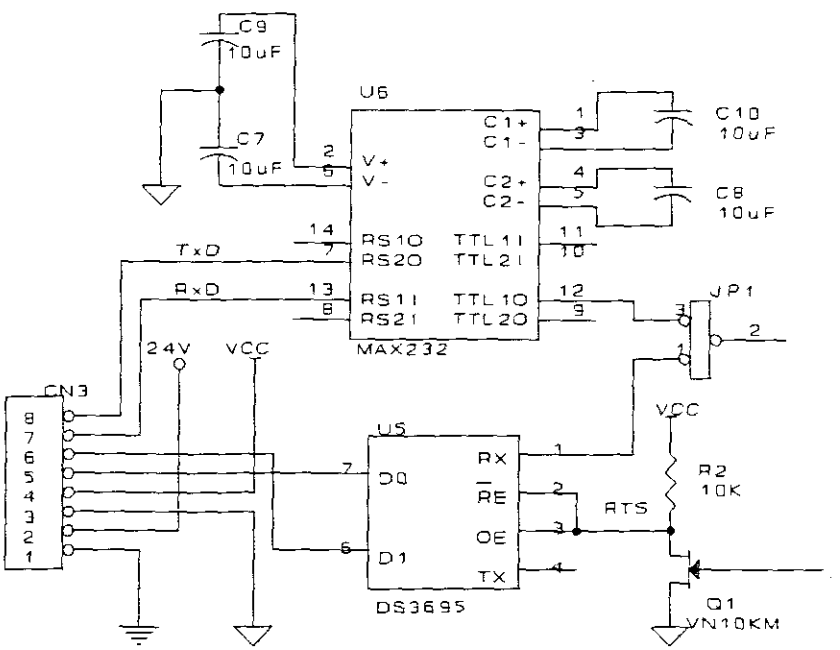


Figure 2-9. Communication Interfaces

4.3 PORT PIN ASSIGNMENT

PORT	DESCRIPTION
P0	LOWER ADDRESS BUS AND DATA BUS
P1	GENERAL PURPOSE I/O

PORT	DESCRIPTION
P1.0	SCANNER DATA
P1.1	SCANNER CLOCK
P1.2	SCANNER RESET
P1.3	DRAWER #1 OPEN
P1.4	DRAWER #1 OPEN / CLOSE SENSE
P1.5	DRAWER #2 OPEN
P1.6	DRAWER #2 OPEN / CLOSE SENSE
P1.7	RECEIPT SWITCH SENSE
P2	UPPER ADDRESS BUS
P3	SPECIFIC AND GENERAL PURPOSE
P3.0	RX DATA
P3.1	TX DATA

PORT	DESCRIPTION
P3.2	IRQ OF KEYBOARD CONTROLLER
P3.3	CARD DATA
P3.4	CARD CLOCK
P3.5	ENABLE FOR DS3695
P3.6	~WR
P3.7	~RD

Table 2-1. Port Pin Assignment

5. INPUT / OUTPUT DECODE CIRCUITRY

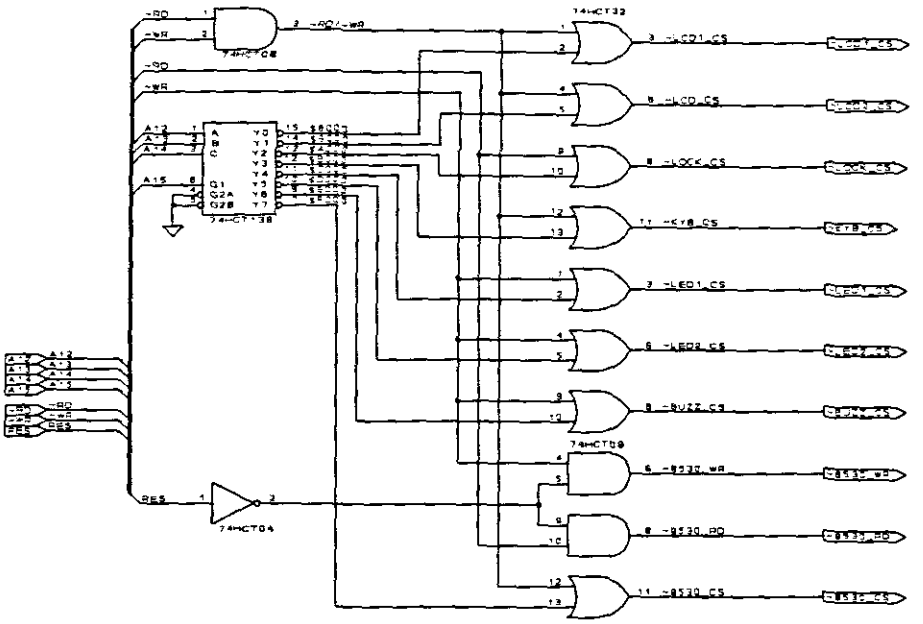


Figure 2-10. Equivalent CCT for 20L10 PAL

The Upper 32K bytes of Data Memory has been allocated to I/O devices. This meant that 32K bytes was available for Memory Mapped I/O. The 32K byte block is further decode into eight, 4K byte blocks. This means that each device has a 4K block of address space assigned to it. Address line A15 is used to select the decoder, '138. Address lines A14, A13, A12 are used to select 1 of the eight possible devices.

As can be seen from Figure 2-10, the ~RD and ~WR lines are logically AND'ed, which means that if either one of these lines are active, the output will be active 'L'. This signal is used to ensure that a device will only be selected upon the generation

of a valid ~RD or ~WR.

~Chip Select 0 (~CS0) is generated at address 8000H and is used to select Liquid Crystal Display #1 (LCD #1). The ~RD / ~WR signal is further OR'ed with ~CS0 to ensure that LCD #1 is only selected during a valid RD/WR as well as ~CS0 being valid. Both the ~RD / ~WR signal and ~CS0 has to be active 'L' before the display is selected via ~LCD1_CS. The same applies for ~LCD2_CS and for ~KYB_CS. All these devices can be read as well as written, therefore the need for the ~RD and ~WR signals.

The LED's and the buzzer are only written to, therefore the ~WR signal is logically OR'ed with their respective chip selects to ensure that these devices are only selected during a valid write and when their respective chip selects are valid.

The 82530 communications chip does not have a reset pin available on the device, due to this a special function has to be implemented on the ~RD and ~WR pins of the 82530. To ensure a valid reset during power-up, the ~RD and ~WR pins have to be held low, this is the only way to ensure a valid hardware reset. The chip can also be reset via software. The ~RD and ~WR signals for the 82530 communications chip are therefore logically AND'ed with the reset signal. This forces the ~82530_RD and ~82530_WR signals both 'L' at power-up, thus ensuring a valid hardware reset. The ~82530_CS is generated at address 0F000H and is logically OR'ed with the ~RD / ~WR signals.

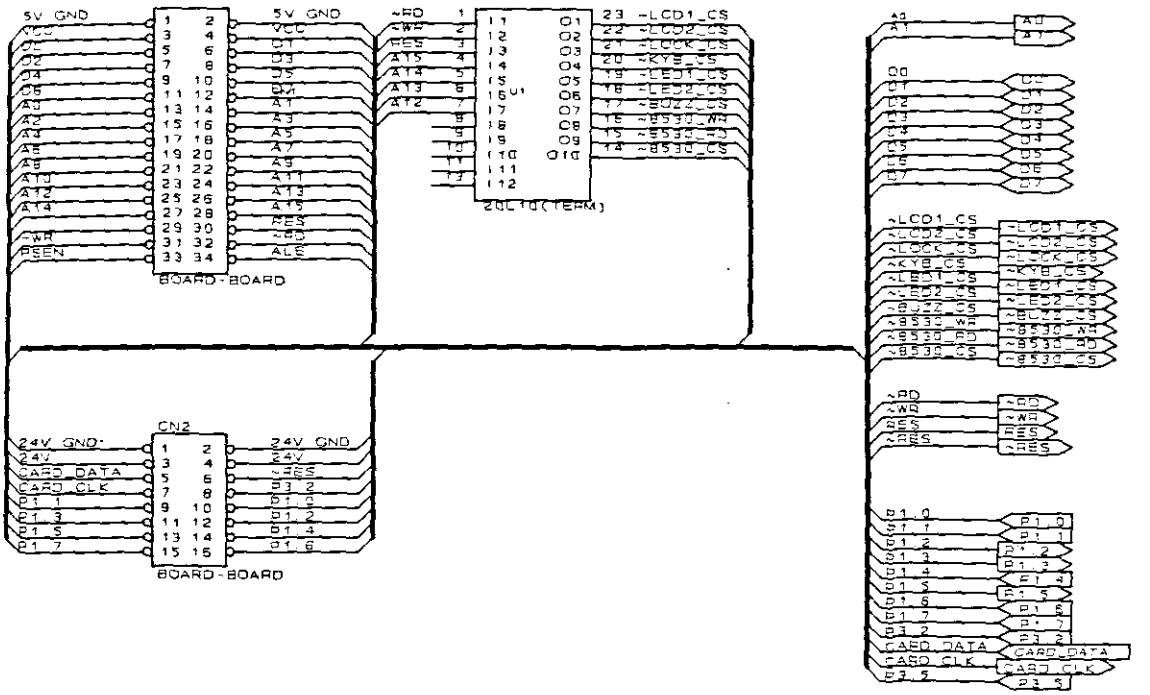


Figure 2-11. I/O Decoder 20L10

It was decided to use a 20L10 PAL to replace the 4 chips that would have been used to do the decoding. A great deal a space was also saved by utilising the PAL. Figure 2-11 shows the PAL and the various chip selects.

6. ADDRESS MAP AND I/O MAP

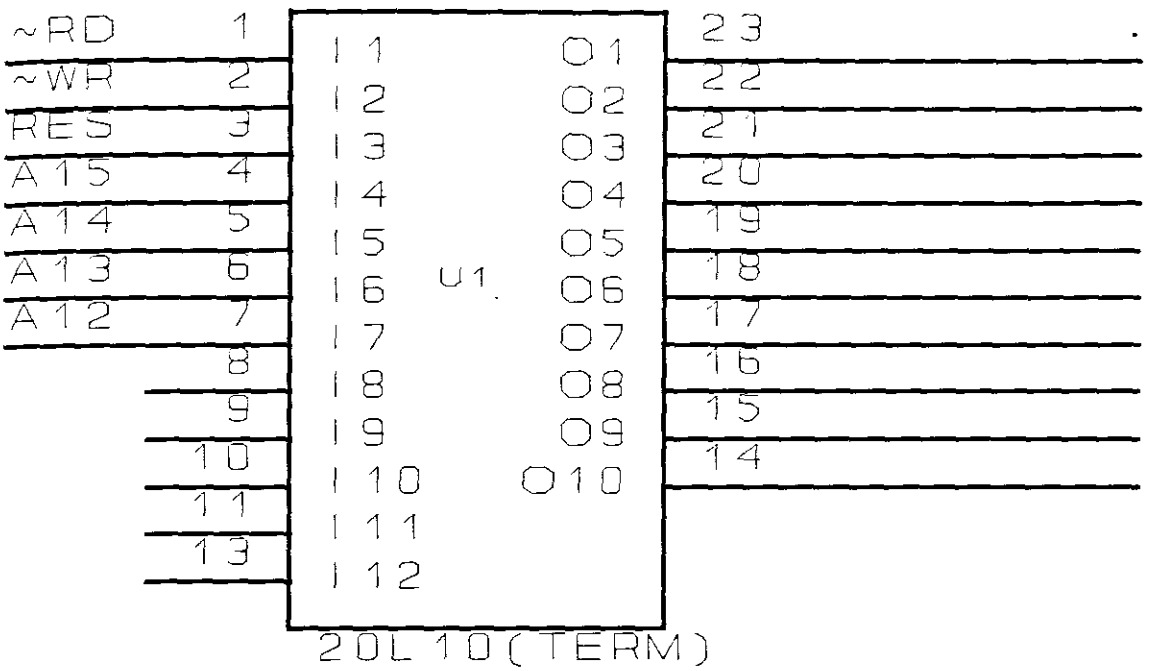


Figure 2-12. 20L10 PAL

6.1 MEMORY I/O EQUATES

```
LCD1_CMD_WR     EQU   8000H     ;INSTRUCTION WRITE
LCD1_STAT       EQU   8001H     ;READ LCD STATUS BIT, D7
LCD1_DAT_WR     EQU   8002H     ;DATA BUFFER WRITE
LCD1_DAT_RD     EQU   8003H     ;DATA BUFFER READ

LCD2_CMD_WR     EQU   9000H     ;INSTRUCTION WRITE
LCD2_STAT       EQU   9001H     ;READ LCD STATUS BIT, D7
LCD2_DAT_WR     EQU   9002H     ;DATA BUFFER WRITE
LCD2_DAT_RD     EQU   9003H     ;DATA BUFFER READ

LOCK_JMP        EQU   0A000H    ;LOCK AND JUMPERS
```

```
KYB_DAT         EQU     0B000H
KYB_CMD         EQU     0B001H

LED_1           EQU     0C000H

LED_2           EQU     0D000H

BUZZER          EQU     0E000H

COMSCB          EQU     0F000H    ;COMMS STATUS/CONTROL REG. CH B
COMDATB         EQU     0F001H    ;DATA REG. CH B
COMSCA          EQU     0F002H    ;COMMS STATUS/CONTROL REG. CH A
COMDATA         EQU     0F003H    ;DATA REG. CH A
```

Table 2-2. Memory I/O Equates

6.2 MEMORY MAP

ADDRESS	CONTROL	DESCRIPTION
0F003H	~82530_CS	DATA REG. FOR CH A
0F002H	~82530_CS	COMMAND REG. FOR CH A
0F001H	~82530_CS	DATA REG. CH B

ADDRESS	CONTROL	DESCRIPTION
0F000H	~82530_CS	COMMAND REG. CH B
0E000H	~BUZZ_CS	BUZZER
0D000H	~LED2_CS	SECOND BANK OF LED's
0C000H	~LED1_CS	FIRST BANK OF LED's
0B000H	~KYB_CS	8279 DATA REGISTER
0B001H	~KYB_CS	8279 COMMAND REGISTER
0A000H	~LOCK_CS	CONTROL LOCK
9003H	~LCD2_CS	READ LCD DATA BUFFER
9002H	~LCD2_CS	WRITE LCD DATA BUFFER
9001H	~LCD2_CS	LCD STATUS REGISTER
9000H	~LCD2_CS	LCD INSTRUCTION REGISTER

ADDRESS	CONTROL	DESCRIPTION
8003H	~LCD1_CS	READ LCD DATA BUFFER
8002H	~LCD1_CS	WRITE LCD DATA BUFFER
8001H	~LCD1_CS	LCD STATUS REGISTER
8000H	~LCD1_CS	LCD INSTRUCTION REGISTER
7FFFH 0000H	~A15	32K BYTES STATIC RAM 64256 SRAM
0FFH 00H		256 BYTES OF INTERNAL RAM
1FFFH 0000H	~A15	EXTERNAL ROM (8K X 8) 2764 EPROM

Table 2-3. Memory Map

7. BUZZER DESCRIPTION

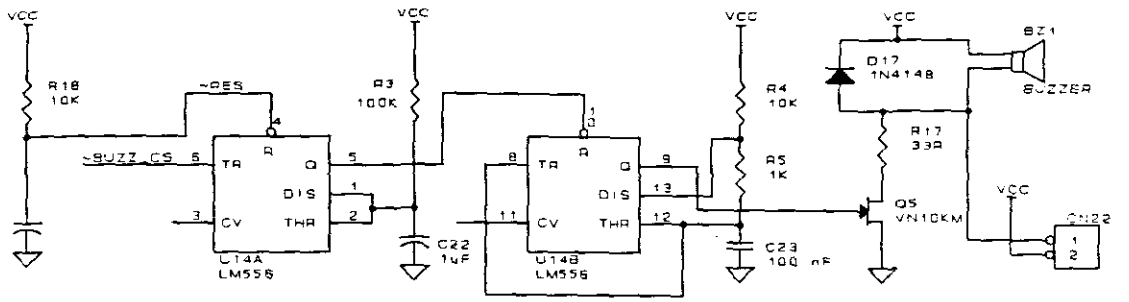
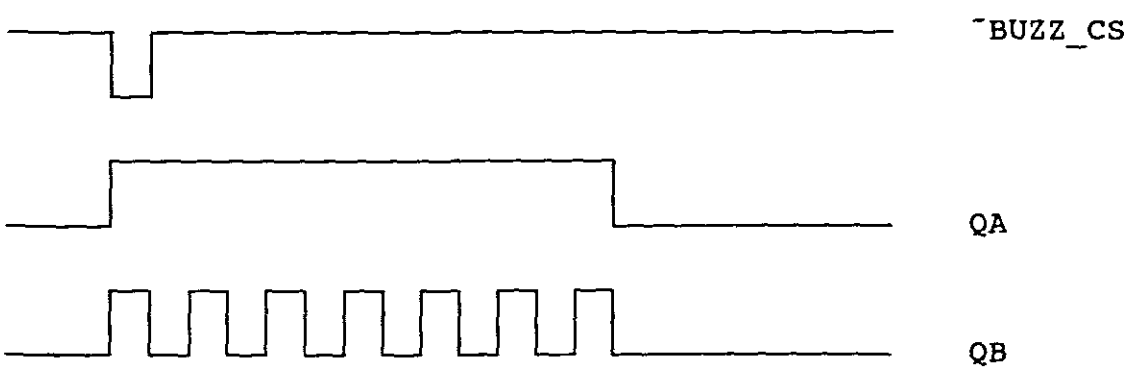


Figure 2-13. Buzzer Circuit

The LM556 is a dual version of the ever popular LM555 timer. The timer can be used in three modes, Astable, Bistable and Monostable modes. The one half of the LM556 is configured in Astable or free-running mode. The frequency is set so as to generate a pleasing beep each time a key is depressed. The second half of the LM556 is used in Monostable or one-shot mode. It produces a fixed duration pulse each time it is triggered. This pulse is used to enable the first half of the LM556. There is a facility to either drive an on-board buzzer or to connect a speaker which could be off-board.



It was decided to use the LM556 so as to reduce the processor overhead. The processor merely has to generate a ~BUZZ_CS, as opposed to having to continuously toggle a port bit to generate the buzzing. The one drawback of using the above approach is that the frequency cannot be changed, which could be useful for indicating error conditions.

8. CONTROL LOCK AND TERMINAL NUMBER INTERFACE

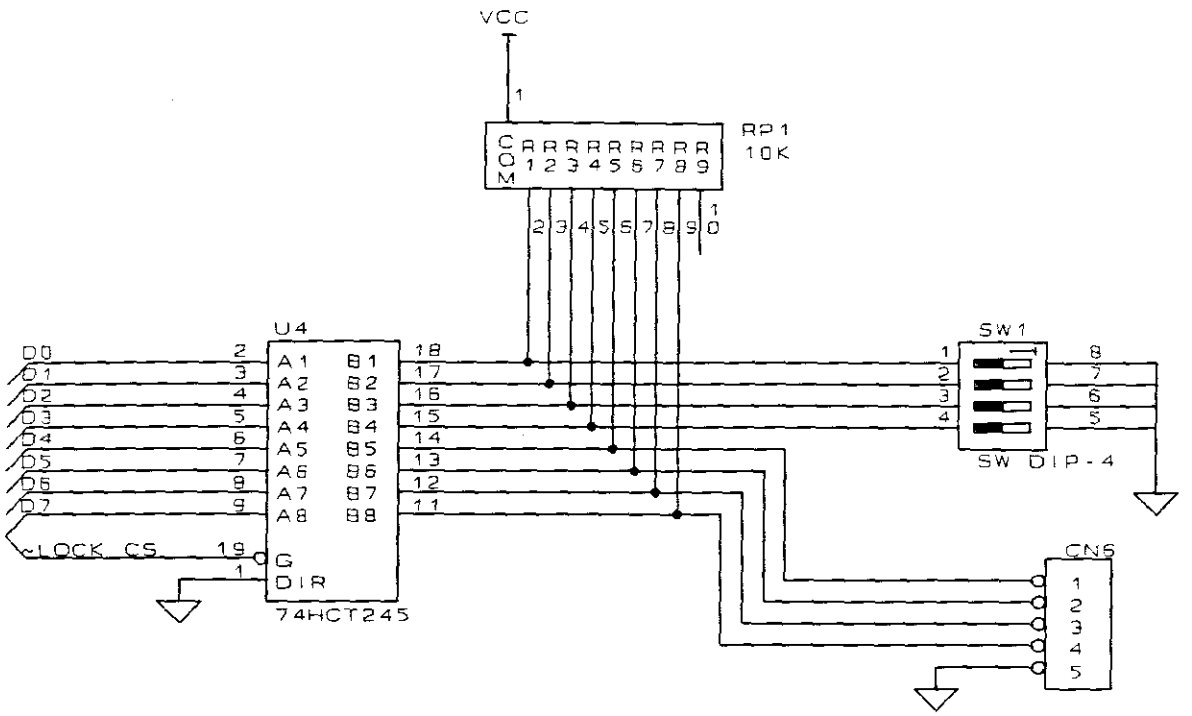


Figure 2-14. Lock & Terminal No. Interface

The '245 is a bi-directional buffer, but in this configuration the buffer direction is fixed to read data. The inputs to the '245 are pulled up via a 10KΩ resistor pack. The lower nibble of the '245 is allocated to the 4 bit dip switch, which is used to set the terminal number. The upper nibble is allocated to the 4 bit control lock. Whenever a ~LOCK_CS is generated, the status of the lock and terminal number are read.

9. LED INTERFACE

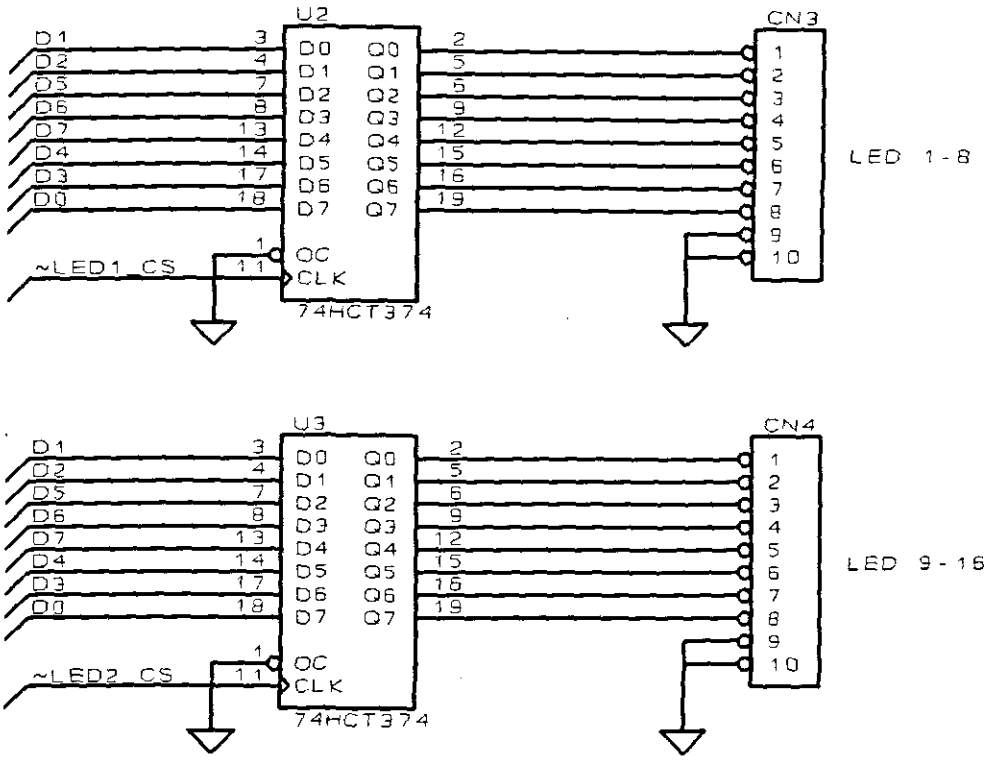


Figure 2-15. LED Interface

The '374 is an octal latch. In this configuration the output is permanently enabled. Whenever an LED_CS is generated the data present on the data bus is latched via the '374 and this is reflected on the LED's. The LED's are used as status indicators eg paper empty, offline, validation.

10. LIQUID CRYSTAL DISPLAY INTERFACE (LCD)

10.1 GENERAL DESCRIPTION

DMC series is the name given to the dot matrix LCD display modules that have recently been developed by OPTREX CORPORATION. The modules consist of high contrast-large viewing angle TN type LC panel, with a CMOS driver and controller, which have a low power consumption. The controller is equipped with an internal character generator ROM, RAM and RAM for display data. All display functions are controllable by instructions making the interfacing with the CPU easy.

Both display data RAM and character generator RAM can be read, making it possible to use any part not used for display as general data RAM.

10.2 LCD SPECIFICATIONS

1. 5 X 7 + Cursor. 5 X 8 dots or 5 X 11 dots. Dot matrix LCD

2. 4 bit or 8 bit interface with CPU is possible

3. Display data RAM 80 x 8 bit (Max 80 characters)

4. Character generator ROM Character font 5 X 7 dots 160
   Character font 5 X 10 dots 32

5. Character generator RAM Program write (64 X 8 bit)
   Character font 5 X 7 dots
   Character font 5 X 10 dots

6. Both display data RAM and Character generator RAM can be read by the CPU

7. Duty ratio 1 Line Display:
   1/8 duty 5 X 7 dots + Cursor, 5 X 8 dots
   1/11 duty 5 X 11 dots
   1/16 duty 5 X 7 dots + Cursor, 5 X 8 dots
   2 Line Display:
   1/16 duty 5 X 7 dots + Cursor, 5 X 8 dots
   4 Line Display:
   1/16 duty 5 X 8 dots

8. Wide variety of operating instructions: Display clear, Cursor home, Display ON/OFF, Display character blink, Cursor shift, Display shift

9. Internal automatic reset circuit when supplied with power

10. Internal oscillating circuit

11. CMOS process used

10.3 CONNECTION POSITIONS

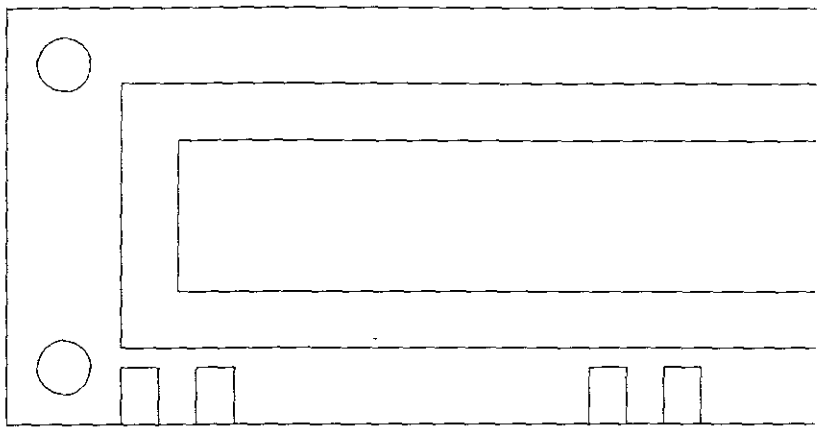


Figure 2-16. LCD Interface

PIN #	SYMBOL
1	Vss
2	Vcc
3	Vee
4	RS
5	R/W
6	E
7	DB0
8	DB1
9	DB2
10	DB3
11	DB4
12	DB5
13	DB6
14	DB7

Table 2-4. Pin Connections

10.4 PIN DESCRIPTION

SIGNAL NAME	DESCRIPTION
DB4 - DB7	4 LINES OF HIGH ORDER DATA BUS. TRI - STATE, BI-DIRECTIONAL. TRANSFER OF DATA BETWEEN CPU AND MODULE IS PROCESSED VIA THESE LINES. ALSO DB7 CAN BE USED AS A BUSY FLAG.
DB0 - DB3	4 LINES OF LOW ORDER DATA BUS. TRI-STATE, BI-DIRECTIONAL. TRANSFER OF DATA BETWEEN CPU AND MODULE IS PROCESSED VIA THESE LINES. HOWEVER IN THE CASE OF 4 BIT OPERATION, THEY ARE NOT USED.
E	OPERATION START SIGNAL FOR DATA READ / WRITE.
R/$\overline{W}$	SIGNAL TO SELECT READ (R) OR WRITE (W). '0' : WRITE '1' : READ

SIGNAL NAME	DESCRIPTION
$\overline{RS}$	SIGNAL TO SELECT REGISTER. '0' : INSTRUCTION REGISTER (WRITE) '0' : BUSY FLAG; ADDRESS COUNTER (READ) '1' : DATA REGISTER (WRITE READ)
Vee	TERMINAL FOR LCD INTENSITY
Vcc	+5V
Vss	0V (GND)

Table 2-5. Signal Description

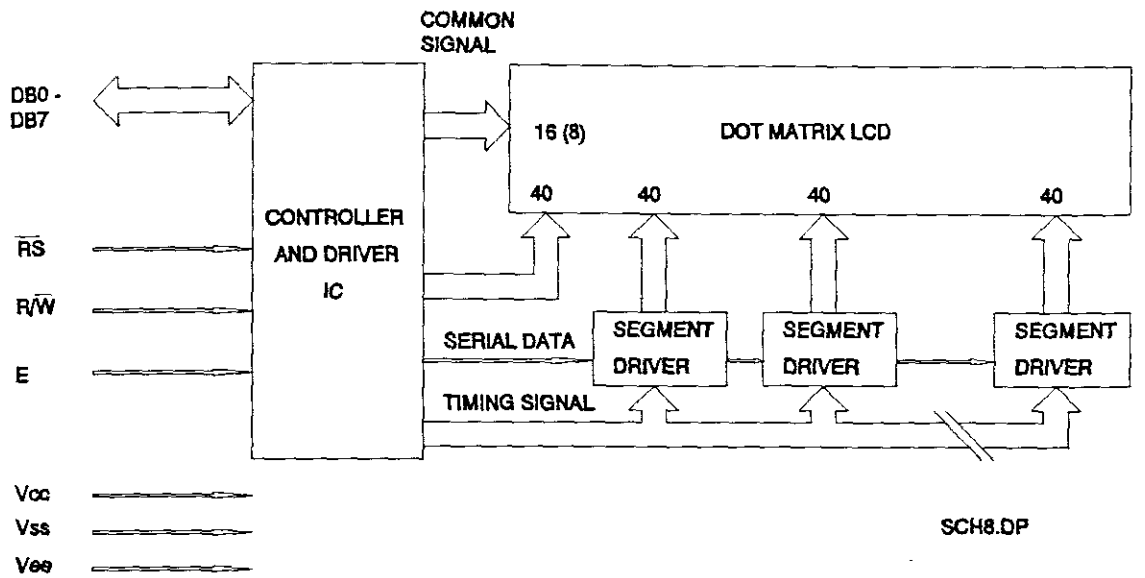


Figure 2-17. LCD Block Diagram

10.5 EXPLANATION OF INTERNAL OPERATION

10.5.1 REGISTER

The Controller has 2 kinds of 8 bit registers, they are the Instruction register (IR) and the data register (DR).

IR is a register to store instruction codes like Display Clear or Cursor Shift as well as address information for display data RAM (DD RAM) or character generator RAM (CG RAM). The IR can be written from CPU but cannot be read by the CPU. DR is a register used for storing temporary data to be written into DD RAM or CG RAM, and data to be read out from DD RAM or CG RAM. Data written into DR from CPU is automatically written into DD RAM or CG RAM

by internal operation. Also DR is used to store data when reading out data from DD RAM or CG RAM. When address information is written into IR, data is read out from DD RAM or CG RAM to DR by internal operation. Data transfer to the CPU is then completed by the CPU reading the DR. After CPU reads DR, data in the DD RAM or CG RAM at the next address is sent to the DR for the next read from the CPU. Register select (RS) signals make their selection from these two registers.

$\overline{RS}$	R/$\overline{W}$	OPERATION
0	0	IR WRITE, INTERNAL OPERATION (DISPLAY CLEAR etc)
0	1	BUSY FLAG (DB7) AND ADDRESS COUNTER. (DB0 - DB6) READ
1	0	DR WRITE, INTERNAL OPERATION (DR > DD RAM OR CG RAM)
1	1	DR READ, INTERNAL OPERATION (DD RAM OR CG RAM)

Table 2-6. Register Selection

10.5.2 BUSY FLAG (BF)

When the Busy Flag is '1', the module is busy with an internal operation and the next instruction will not be accepted. As shown in Table 2-6, the busy flag outputs to DB7 when RS = 0, R/W = 1. The next instruction must be written after ensuring that the busy flag is '0'.

10.5.3 ADDRESS COUNTER (AC)

The address counter (AC) assigns addresses to DD RAM and CG RAM. When the instruction for address is written in IR, the address information is sent from IR to AC. The selection of either DD or CG RAM is also determined concurrently by the instruction. After writing into (reading from) DD RAM or CG RAM display data, address counter (AC) is automatically incremented by +1 (or decremented by -1). AC contents are outputted to DB0 - DB7 when $\overline{RS}$ = 0 and R/$\overline{W}$ = 1 as shown in Table 2-6.

10.5.4 DISPLAY DATA RAM (DD RAM)

The display data RAM stores display data represented in 8-bit character code. Its capacity is 80 X 8 bits or 80 characters. The display data RAM that is not used for display can be used as a general data RAM.

10.5.5 CHARACTER GENERATOR ROM (CG ROM)

The character generator ROM (CG ROM) is a ROM capable of generating from an 8-bit character code, 5 X 7 dots or 5 X 10 dots of character patterns. The said ROM can generate 160 kinds of 5 X 7 dots or 32 kinds of 5 X 10 dots of character patterns. However caution is required when operating this ROM, as those character fonts of 5 X 7 dots of the module will not be displayed after the 8th row of 5 X 10 dots character pattern. Also this ROM can be modified to generate character patterns generated by the user.

10.5.6 CHARACTER GENERATOR RAM (CG RAM)

The character generator RAM is the RAM with which the user can rewrite character patterns via software. With 5 X 7 dots, 8 types of character patterns can be written. The area not used for the display can be used as general purpose data RAM.

10.6 DETAILED EXPLANATION OF INSTRUCTIONS

10.6.1 CLEAR DISPLAY

$\overline{RS}$	R/$\overline{W}$	D7	D6	D5	D4	D3	D2	D1	D0
0	0	0	0	0	0	0	0	0	1

Writes the space code "20" (hexadecimal) into all address of DD RAM. Returns display to its original position if it was shifted. In other words the display disappears and the cursor or blink moves to the left edge of the display (the first line if 2 lines are displayed). The execution of the clear display instruction, sets entry mode to increment mode.

10.6.2 RETURN HOME

$\overline{RS}$	R/$\overline{W}$	D7	D6	D5	D4	D3	D2	D1	D0
0	0	0	0	0	0	0	0	1	X

X = don't care

Sets the DD RAM address to "0" in the address counter. Returns the display to its original position if it was shifted. DD RAM contents are not change. The cursor or the blink moves to the left edge of the display (the first line if 2 lines are displayed)

10.6.3 ENTRY MODE SET

$\overline{RS}$	R/$\overline{W}$	D7	D6	D5	D4	D3	D2	D1	D0
0	0	0	0	0	0	0	1	I/D	S

I/D:

Increments (I/D = 1) or decrements (ID = 0) the DD RAM

address by 1 when a character code is written into or read from the DD RAM. The cursor or blink moves to the right when incremented by 1. The same applies to writing and reading of CG RAM.

S:

Shifts the entire display either to the right or to the left when S = 1, shift to the left when I/D = 1 and to the right when I/D = 0. Thus it looks as if the cursor stands still and only the display seems to move. The display does not shift when reading from DD RAM nor when S = 0.

10.6.4 DISPLAY ON/OFF CONTROL

$\overline{RS}$	R/$\overline{W}$	D7	D6	D5	D4	D3	D2	D1	D0
0	0	0	0	0	0	1	D	C	B

D:

The display is ON when D = 1 and OFF when D = 0. When OFF due to D = 0, display data remains in the DD RAM. It can be displayed immediately by setting D = 1.

C:

The cursor displays when C = 1 and does not display when C = 0. Even if the cursor disappears, the function of I/D etc, does not change during data write. The cursor is

displayed on the 8th line when a 5 X 7 dot character font has been selected.

B:

The character indicated by the cursor, blinks when B = 1. The blinking effect is achieved switching between all blank characters and the display characters at a 0.4 sec interval. The cursor and the blink can be set to display simultaneously.

10.6.5 CURSOR OR DISPLAY SHIFT

$\overline{RS}$	R/$\overline{W}$	D7	D6	D5	D4	D3	D2	D1	D0
0	0	0	0	0	1	S/C	R/L	X	X

X = don't care

Shifts cursor position or display to the right or left without writing or reading display data. This function is used to correct or search the display. In a 2-line display the cursor moves to the 2nd line when it passes the 40th digit of the 1st line. Notice that the 1st and 2nd line display will shift at the same time. When the displayed data is shifted repeatedly each line only moves horizontally, but the 2nd display line does not shift into the 1st line position. The contents of Address Counter (AC) do not change if the only action performed is shifting the display.

S/$\overline{C}$	R/$\overline{L}$	
0	0	SHIFTS CURSOR POSITION LEFT (AC DEC. BY 1)
0	1	SHIFTS CURSOR POSITION RIGHT (AC INC. BY 1)
1	0	SHIFTS THE ENTIRE DISPLAY TO THE LEFT.
1	1	SHIFTS THE ENTIRE DISPLAY TO THE RIGHT

X = don't care

10.6.6 FUNCTION SET

1-LINE DISPLAY

$\overline{RS}$	R/$\overline{W}$	D7	D6	D5	D4	D3	D2	D1	D0
0	0	0	0	1	DL	0	F	X	X

X = don't care

2-LINE DISPLAY

$\overline{RS}$	R/$\overline{W}$	D7	D6	D5	D4	D3	D2	D1	D0
0	0	0	0	1	DL	1	X	X	X

DL:

Sets interface data length. Data is sent or received in 8

bit lengths. (D7 - D0) when DL = 1, and in bit lengths (D7 - D4) when DL = 0. When the 4 bit length is selected, data must be sent or received twice.

F:

Sets character font.

F = 1 : 5 X 10 dots

F = 0 : 5 X 7 dots

10.6.7 INTENSITY CONTROL

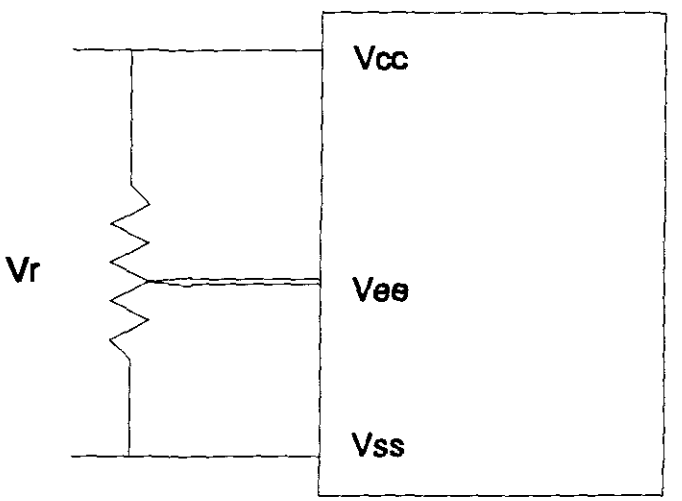


Figure 2-18. Intensity Adjust

The above diagram is the suggested intensity control circuit. The circuit configuration is identical for the LED backlighting

10.7 DESIGN IMPLEMENTATION

The external inverters were necessary, as there were no gates available in the 20L10 PAL. Whenever a valid LCD_CS is generated data is either read or written to LCD number one or two. The backlighting and intensity controls are mounted off-board, making these adjustments available to the operator.

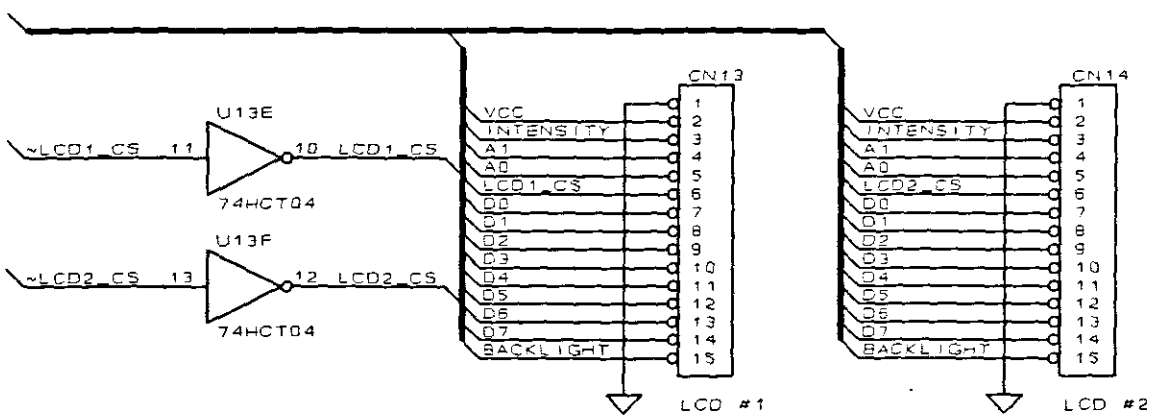


Figure 2-19. LCD Interface

11. MAGNETIC CARD READER INTERFACE

11.1 DATA FORMAT

There are two formats presently available in which the data is stored on the magnetic card, they are the International Air Transport Association (IATA) format and in the American Banking Association (ABA) format.

11.1.1 ABA-CODED DATA FORMAT

11.1.1.1 CODED CHARACTER SET

BITS					ROW	CHARCACTER
P	B4	B3	B2	B1		
1	0	0	0	0	0	0
0	0	0	0	1	1	1
0	0	0	1	0	2	2
1	0	0	1	1	3	3
0	0	1	0	0	4	4
1	0	1	0	1	5	5
1	0	1	1	0	6	6
0	0	1	1	1	7	7
0	1	0	0	0	8	8
1	1	0	0	1	9	9

BITS					ROW	CHARCACTER
P	B4	B3	B2	B1		
1	1	0	1	0	10	:
0	1	0	1	1	11	; *
1	1	1	0	0	12	<
0	1	1	0	1	13	= *
0	1	1	1	0	14	>
1	1	1	1	1	15	? *

Table 2-7. ABA-Coded Character Set

A binary-coded-decimal 4-bit subset with odd parity is used to encode data on the magnetic stripe of ABA-formatted cards. This character code is numeric. Refer to Table 2-7 for the coded character set.

The characters in table 2-7 which are marked with a '*' have specific meanings when used for this application:

Row 11 ';' represents "*start sentinel*"

Row 13 '=' represents "*separator*"

Row 15 '?' represents "*end sentinel*"

P = Odd Parity

11.1.1.2 INFORMATION FORMAT

The format of the information encoded on the magnetic stripe of an ABA-formatted card is as follows:

Start sentinel	1 character
Account number	Up to 19 characters
Separator	1 character
Discretionary data	The balance up to the maximum record length (40 characters)
Stop sentinel	1 character
Longitudinal redundancy check	1 character
Total	40 characters maximum

All the characters are displayable (including the start- and stop-sentinel characters) and are part of the maximum 40-character total.

11.1.1.3 LONGITUDINAL REDUNDANCY CHECK (LRC)

The magnetic stripe reader runs an LRC test on the ABA 4-bit code before it is converted to the 7-bit ASCII code. The LRC test contains the following steps.

1. The parity bit of each character code is stripped off.

2. All characters from the start sentinel to and including the stop sentinel are combined by an exclusive OR function.

3. The result of step 2 is compared to the LRC character (minus the parity bit) that was received from the magnetic stripe of the card read.

11.1.2 IATA-CODED DATA FORMAT

11.1.2.1 CODED CHARACTER SET

				B6	0	0	1	1
				B5	0	1	0	1
				COL				
B4	B3	B2	B1	ROW	0	1	2	3
0	0	0	0	0	SP	0	@	P
0	0	0	1	1	!	1	A	Q
0	0	1	0	2	"	2	B	R
0	0	1	1	3	#	3	C	S
0	1	0	0	4	$	4	D	T
0	1	0	1	5	%	5	E	U
0	1	1	0	6	&	6	F	V
0	1	1	1	7	'	7	G	W
1	0	0	0	8	(	8	H	X
1	0	0	1	9	)	9	I	Y
1	0	1	0	10	*	:	J	Z
1	0	1	1	11	+	;	K	[
1	1	0	0	12	,	<	L	\
1	1	0	1	13	-	=	M	]
1	1	0	0	14	.	>	N	^
1	1	1	1	15	/	?	O	_

Table 2-8. IATA-Coded Character Set

A 6-bit-plus-odd-parity character code is used to encode data on the magnetic stripe of IATA-formatted cards. This character code is alphanumeric. Refer to Table 2-8 for the coded character set.

11.1.2.2 INFORMATION FORMAT

The information encoded on the magnetic stripe of an IATA-formatted card can be in either of two formats, format A or B. The content of each format is as follows:

FORMAT A	FORMAT B
Start sentinel 1 character	Start sentinel 1 character
Format code "A" 1 character	Format code "B" 1 character
Surname	Account number Up to 19 chars
Surname separator "/"	Separator 1 character
Initials or first name	Surname
Separators (when required) = "space"	Surname seperator "/" Initials or first name

FORMAT A	FORMAT B
Title (when used) Separator (when required) = "space" Separator 1 character Discretionary data The balance up to the maximum record length (79 characters) Stop sentinel 1 character Longitudinal redundancy check, 1 character Total 79 characters (max)	Separator (when required) = "space" Title (when used) Separator (when required) = "space" Separator 1 character Discretionary data The balance up to the maximum record length (79 characters) Stop sentinel 1 character Longitudinal redundancy check 1 character Total 79 characters (max)

Table 2-9. IATA Information Format

11.1.2.3 LONGITUDINAL REDUNDANCY CHECK (LRC)

The magnetic stripe reader runs an LRC test on the IATA 6-bit code before it is converted to the 7-bit ASCII code. The LRC test contains the following steps.

1. The parity bit of each character code is stripped off.

2. All characters from the start sentinel to and including the stop sentinel are combined by an Exclusive OR function.

3. The result of step 2 is compared to the LRC character (minus the parity bit) that was received from the magnetic stripe of the card read.

11.2 READER DESCRIPTION

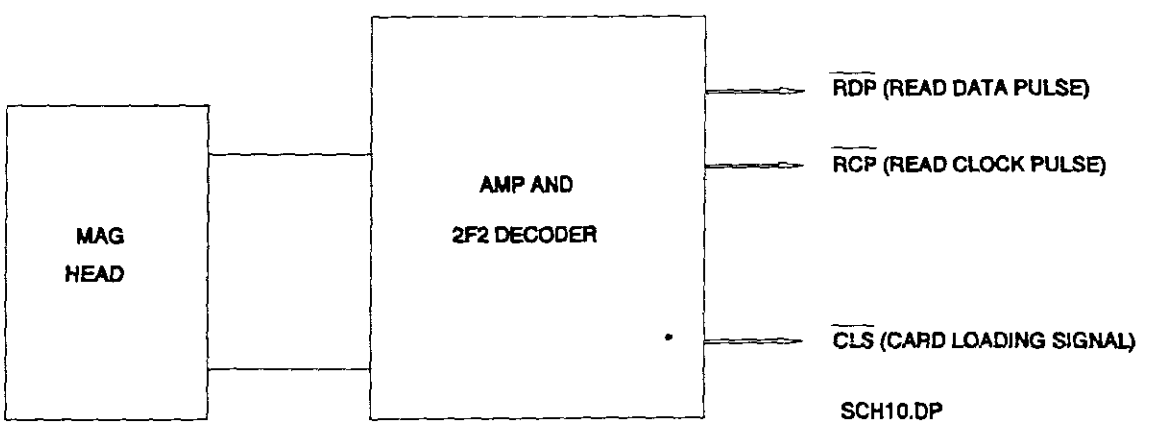


Figure 2-20. Reader Block Diagram

READABLE TRACK	1	2	3
RECORDING DENSITY	210 BPI	75 BPI	210 BPI
RECORDING OUT (@ 10cm/s) BIT INTERVAL (@ 150 cm/s)	1.21 ms 80.6 µsec	3.39 ms 225.8 µs	1.21 ms 80.6 µs
TRACK WIDTH	1.5mm	1.5mm	1.5mm
CENTRE POSITION	7.0±.2mm	10.3±.2mm	13.6±2.mm

Table 2-10. Reader Specification

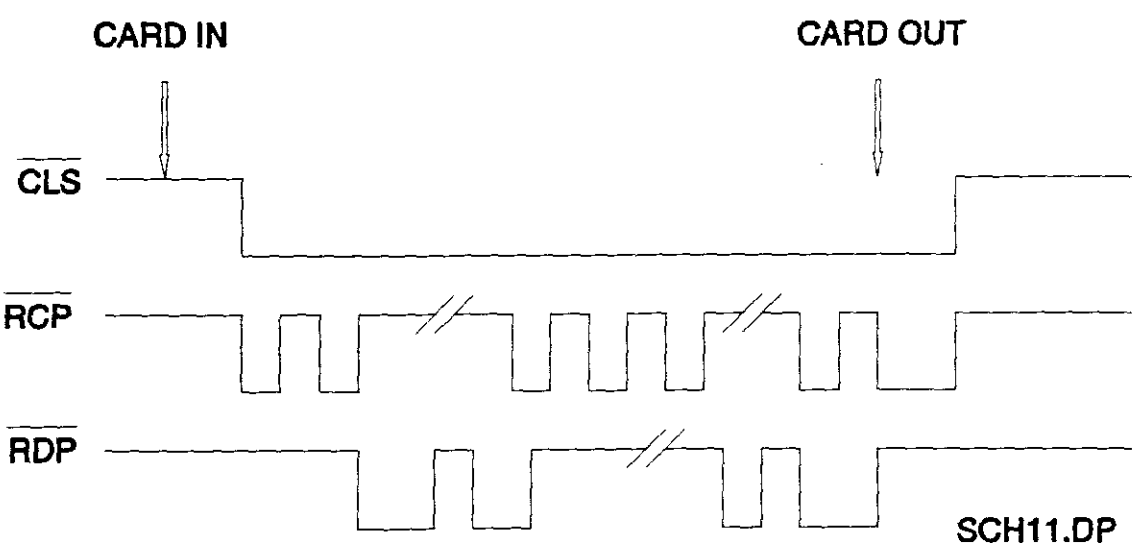


Figure 2-21. Timing Diagram

PIN #	SIGNAL	REMARKS
1	GND - GROUND	GROUND FOR +5V AND SIGNAL GROUND
2	¯RDP - READ DATA PULSE	OUTPUT READ DATA. (NEGATIVE LOGIC) THIS SIGNAL IS DEMODULATED BY 2F2 DECODER, AND SAMPLING OF THIS SIGNAL IS PERFORMED BY THE TRAILING EDGE OF SIGNAL RCP. LEVEL 'H' INDICATES DATA AS '0', AND LEVEL 'L' INDICATES DATA AS '1'
3	¯RCP - READ CLOCK PULSE	THIS SIGNAL IS USED FOR READ DATA SAMPLING. (NEGATIVE LOGIC) THIS SIGNAL PERFORMS SAMPLING FOR SIGNAL RDP AT THE TRAILING EDGE.

PIN #	SIGNAL	REMARKS
4	¯CLS - CARD LOADING	THIS SIGNAL INDICATES THAT A CARD IS RUNNING ON MAG. HEAD. LEVEL IS 'H' WHILE A CARD IS RUNNING ON MAG. HEAD, AND LEVEL BECOMES 'L' WHEN A CARD STOPS OR IS NOT ON MAG. HEAD.
5	+5V	POWER SUPPLY

Table 2-11. Reader Control Signals

11.3 DESIGN IMPLEMENTATION

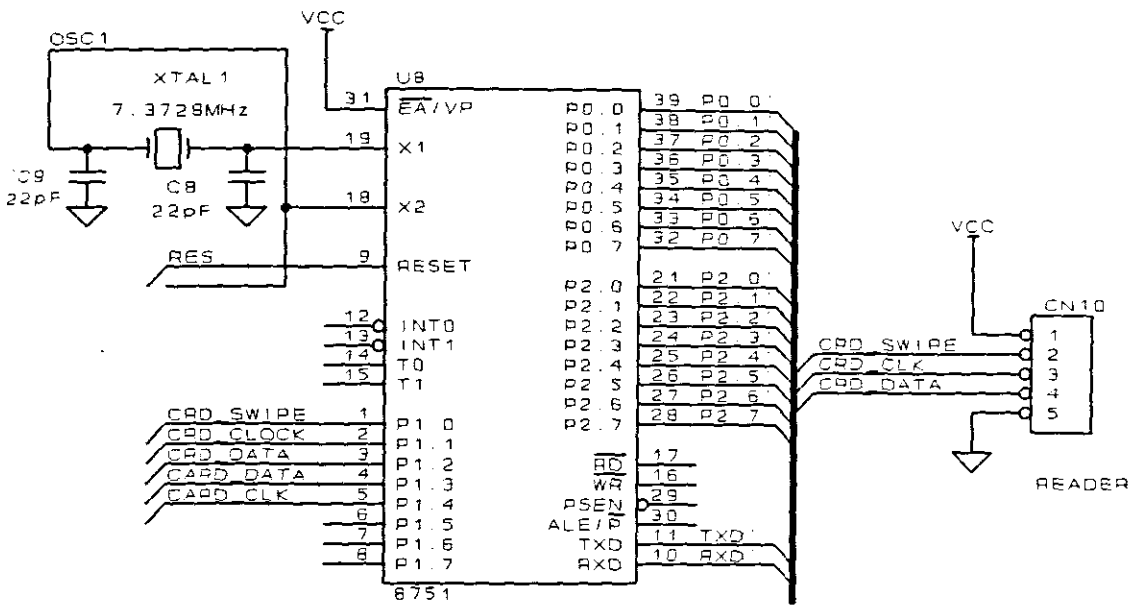


Figure 2-22. Card Data Transfer

It was decided to dedicate the 8751 to the magnetic card reader due to the nature of information being read. If the reader was connected to the main controller, there is always the possibility that a communications interrupt would occur while a magnetic card was being read. The communications interrupt has to have the highest priority, which prevented the reader from being assigned the highest priority. The 8751 was chosen to reduce board space, and the relevant card reader software could fit into the 4K on-chip EPROM.

The 82530 communications chip required an external clock. The 8751 was chosen so that it would be a multiple of the

communications frequency required. The clock output of the 8751 is fed into an inverter, whose output is then fed into a '393, where the frequency is further divided. The communications clock for the 82530 and the keyboard clock for the 8279 is generated via the '393.

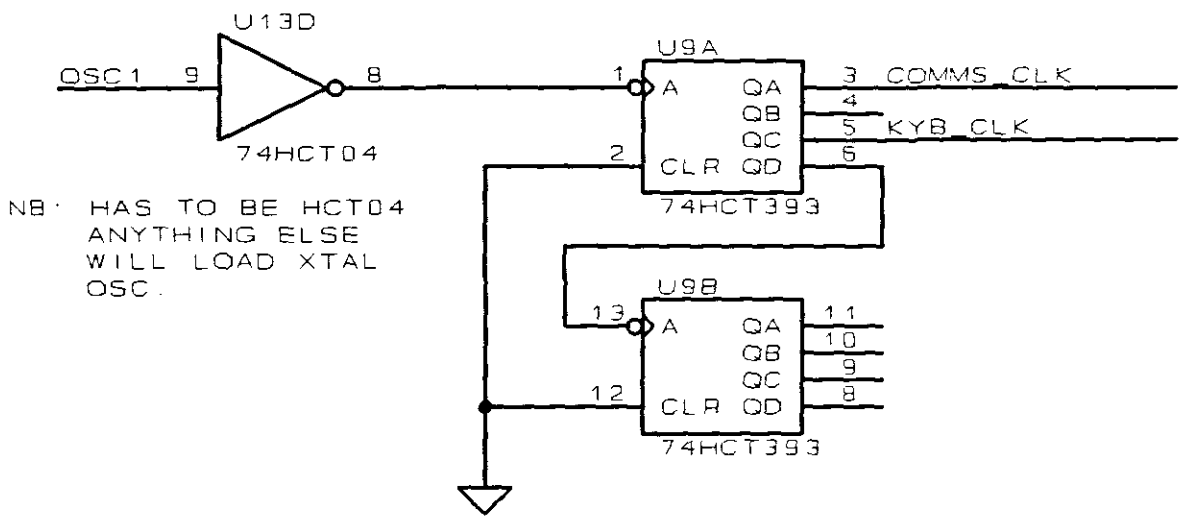


Figure 2-23. I/O Clock Generation

The remaining port pins of the 8751 were routed to an expansion connector for future use. If another device has to be placed onto the terminal in the future, only the code in the 8751 has to be changed.

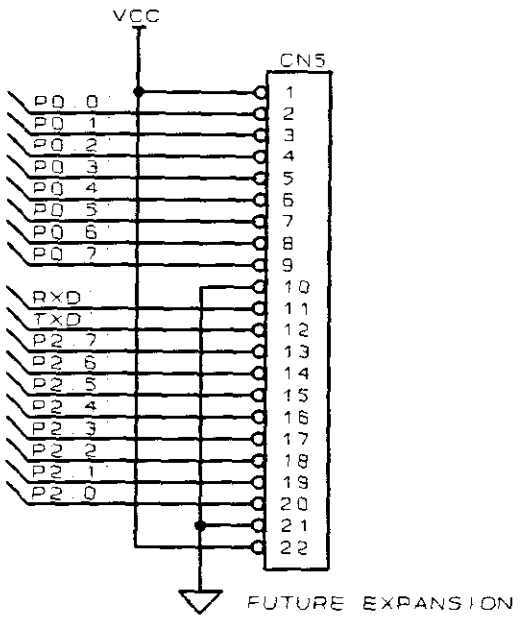


Figure 2-24. Expansion Bus

The 8751 has been configured so that the various port pins are used for I/O. This means that there is no data bus available to transfer data between the 8751 and the main controller, the 8032. This meant that a system had to be devised to facilitate the transfer of data between the 8751 and the 8032. A serial synchronous link was established between the two devices.

Two port pins on the 8032 are connected to two port pins on the 8751. These signals are the CRD_DATA and CRD_CLK signals. If card data is available, the 8751 takes the CRD_DATA line 'L', informing the 8032 that card information is available. The 8032 now responds by supplying the 8751 with the CRD_CLK signal. The 8751 in turn responds by placing the serial data on the CRD_DATA line. The 8032 reads the data on the falling edge of the CRD_CLK

signal. A status byte is also returned to the 8032, if the status byte is OFFH, then an invalid swipe has taken place, and the card has to be re-read.

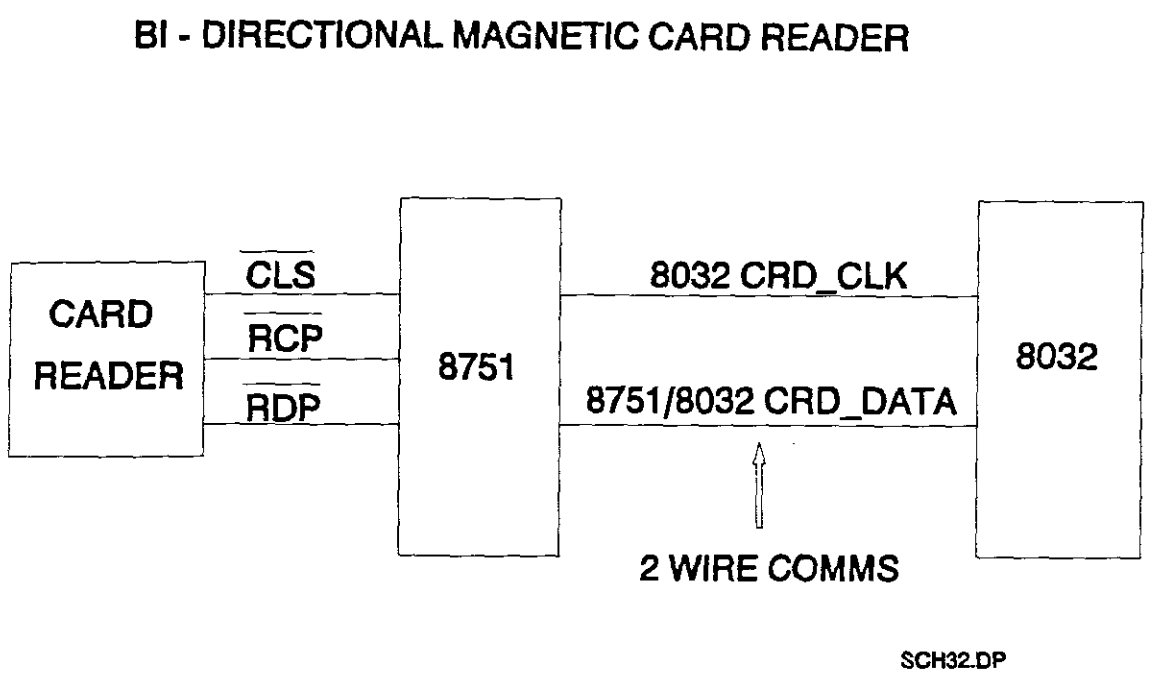


Figure 2-25. Interconnection between 8032 and 8751

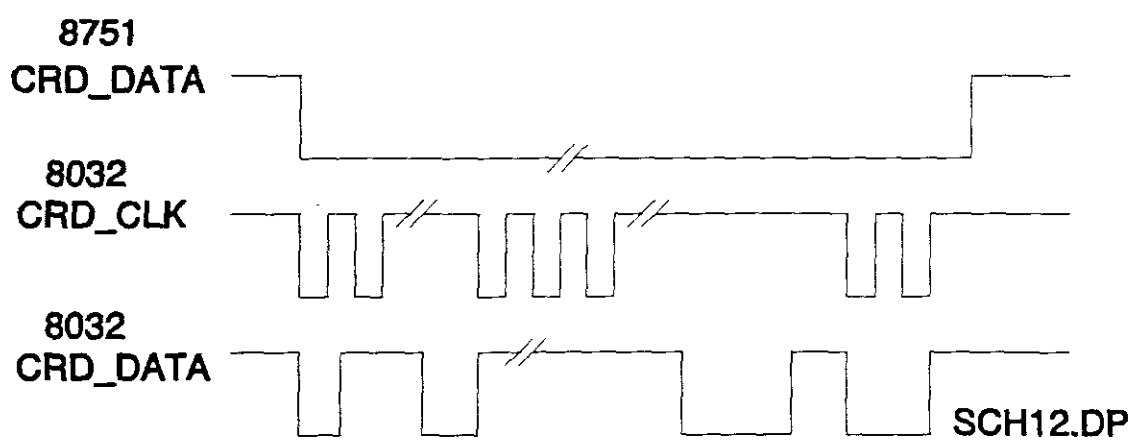


Figure 2-26. Data Transfer Timing Diagram

12. KEYBOARD INTERFACE

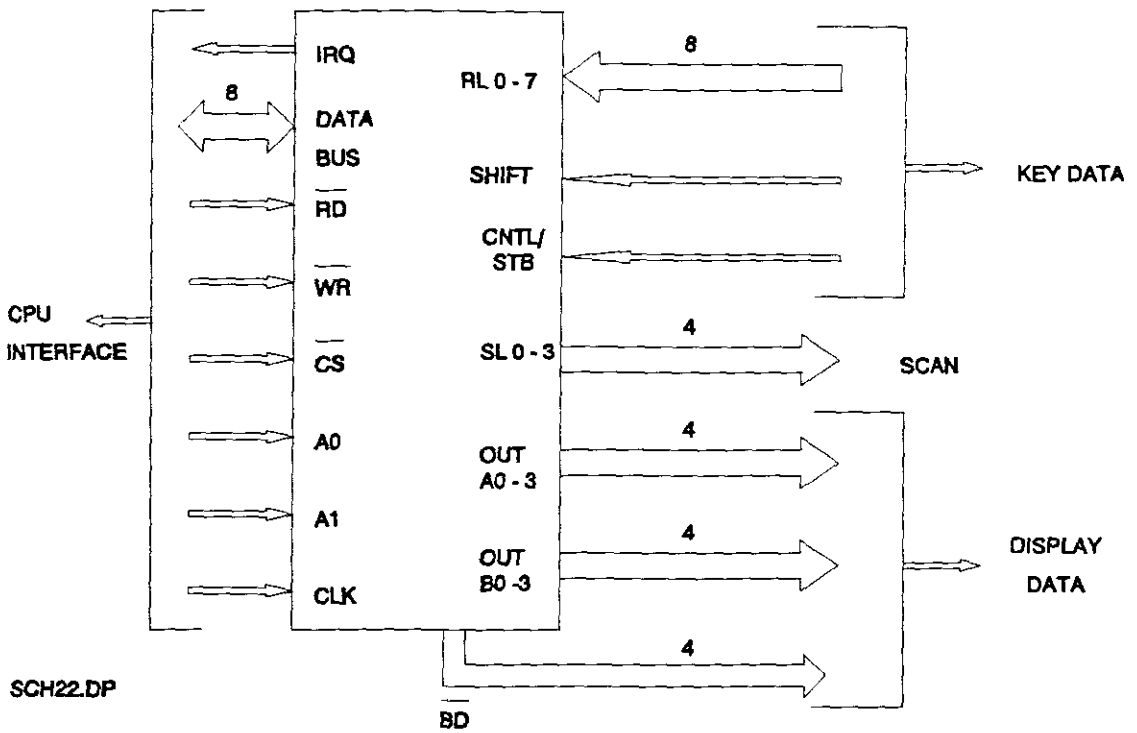


Figure 2-27. 8279 Block Diagram

12.1 FEATURES OF THE 8279

The Intel 8279 is a general purpose programmable keyboard and display I/O interface device capable of simultaneous keyboard / display operations. The 8279 has the following features:

- Scanned Keyboard Mode
- Scanned Sensor Mode
- Strobed Input Entry Mode
- 8-Character Keyboard FIFO
- 2-Key Lockout or N-Key Rollover with Contact Debounce
- Dual 8- or 16-Numerical Display

Single 16-Character Display

Right or Left Entry 16-Byte Display RAM

Mode Programmable CPU

Programmable Scan Timing

Interrupt Output on Key Entry

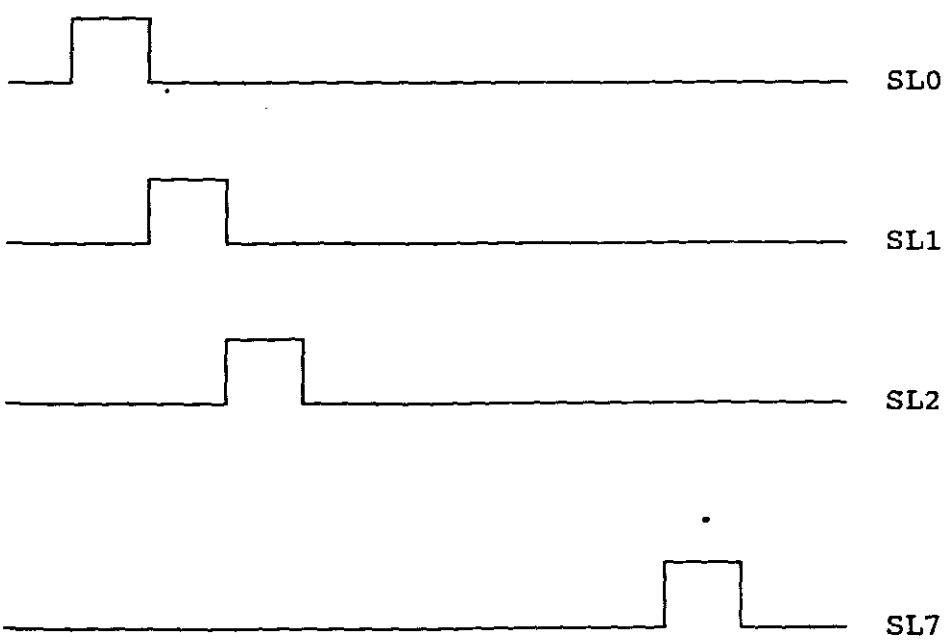
12.2 HARDWARE DESCRIPTION

It was decided to use the 8279 to alleviate the microcontroller of the menial task of scanning a keyboard and performing the necessary debouncing, which is required when reading mechanical switches. The microcontroller merely has to read a status register of the 8729, and if any key depression has been registered, read the FIFO of the 8279. The display portion provides a scanned display interface for LED, incandescent, and other popular display technologies. Due to the fact that Liquid Crystal Displays were used, the display portion of the 8279 was not utilised. Although only 50% of the 8279's features are being utilised, the overhead that will be placed on the microcontroller to scan a keyboard justifies the use of the 8279.

The keyboard portion can provide a scanned interface to a 64-contact key matrix. The specification for the INTELLIGENT TERMINAL required a maximum of 128 keys. The 8279 has two extra inputs, a control and a shift input. A method was devised to include the shift input in the keyboard interface, so as to

increase the amount of keys that could be read. The inclusion of 74HCT30 increases the number of return lines, thereby extending the matrix to 128 keys. If the control input were used, the matrix could be increased to 192 keys.

When a key on the extended key matrix is depressed, the shift input is activated and the relevant return line registers the key depression. When the key value is read, the shift status will be set, indicating that a key on the extended keyboard has been depressed. The 10KΩ resistor pack is included to pull the inputs to the 74HCT30 'H', when no keys are depressed. Diode D9 - D16 are included to prevent shorting two outputs of the 74HCT139 if two keys on different scan lines are depressed simultaneously.



12.3 SOFTWARE DESCRIPTION

When a key is depressed, the debounce logic is set. Other depressed keys are looked for during the next two scans. If none are encountered, it is a single key depression and the key depression is entered into the FIFO along with the status of the CNTL and SHIFT lines. Key entries set the interrupt output line to the CPU. In the scanned keyboard mode, the character entered into the FIFO corresponds to the position of the switch in the keyboard plus the status of the CNTL and SHIFT lines. CNTL is the MSB of the character and SHIFT is the next Most Significant Bit. The next three bits are from the scan counter and indicate the row the key was found in. The last three bits are from the column counter and indicate to which return line the key was connected.

CNTL	SHIFT	S2	S1	S0	R2	R1	R0

12.4 DESIGN IMPLEMENTATION

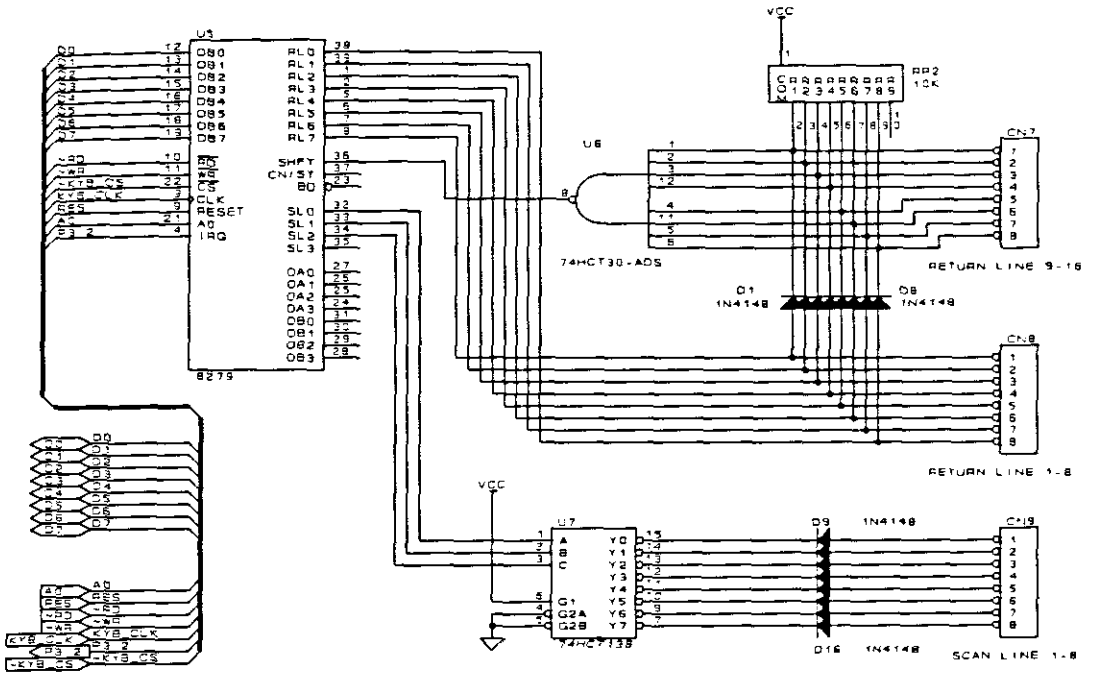


Figure 2-28. Keyboard Interface

13. SERIAL COMMUNICATIONS INTERFACE (ZILOG 8530)

13.1 8530 FUNCTIONAL DESCRIPTION

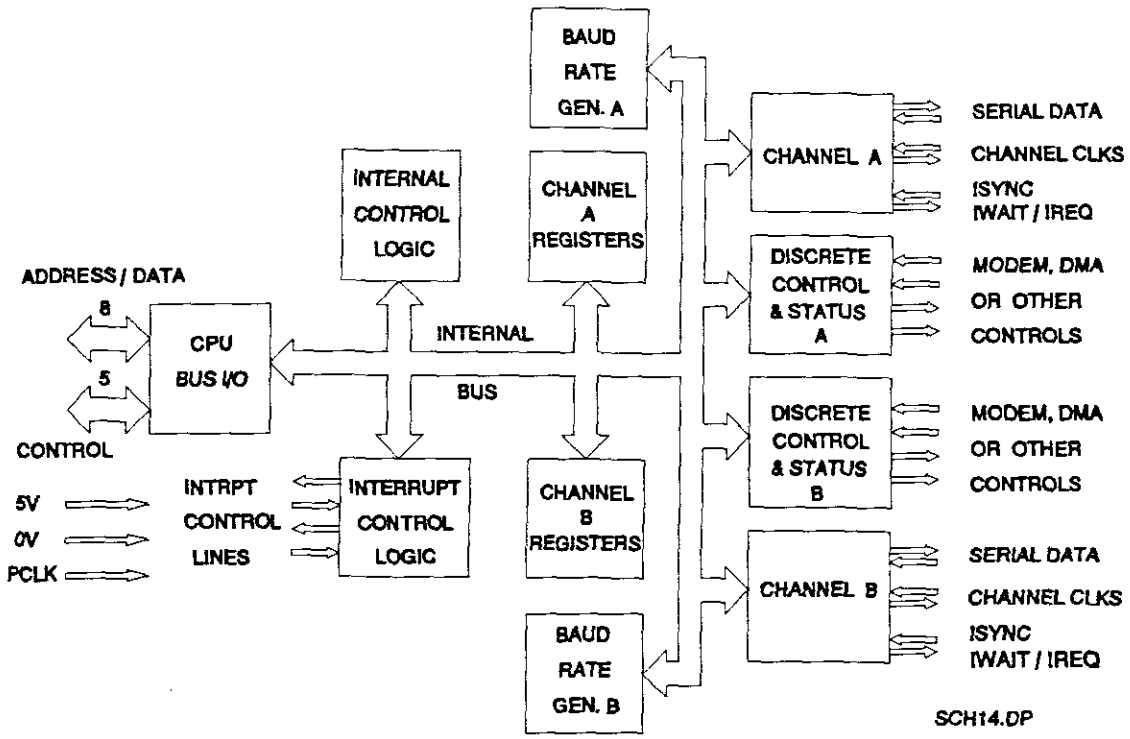


Figure 2-29. 8530 Block Diagram

The Z8530 Serial Communications Controller (SCC) is a Zilog Z8000 peripheral component, designed to provide multifunction support for handling the large variety of serial communication protocols available. The Z-SCC internal structure provides all the interrupt and control logic necessary to interface with nonmultiplexed buses. Interface logic is also provided to monitor modem or peripheral control inputs and outputs. All of the control signals are general purpose and can be applied to various peripheral devices as well as used for modem control.

The centre for data activity revolves around the internal read and write registers. The programming of these registers provides the Z-SCC with a functional "personality", i.e., register values can be assigned before or during program sequencing to determine how the Z-SCC will establish a given communication protocol.

13.2 HARDWARE DESCRIPTION

The 8530 in conjunction with the two MAX232 line drivers, constitutes the basis of the dual channel RS232 serial ports. The 8530 does not have a RESET pin available, and the only manner whereby a hardware reset can be ensured is by simultaneously pulling the ~RD and ~WR lines 'L'. The ~RD and ~WR lines for the 8530 are therefore AND'ed with the RESET line, to ensure a hardware reset at power-up.

The clock for the 8530 baud rate generator is derived from the clock output of the 8751, which is used for the magnetic card reader. The clock signal is further divided by the '393, and then applied to pin# 20 of the 8530. Address lines A0 and A1 select the command / data registers, and selects between channel A or channel B. The addresses are as follows:

```
COMSCB          EQU   0F000H   ;COMMS STATUS/CONTROL REG. CH B
COMDATB         EQU   0F001H   ;DATA REG. CH B
COMSCA          EQU   0F002H   ;COMMS STATUS/CONTROL REG. CH A
COMDATA         EQU   0F003H   ;DATA REG. CH A
```

The serial ports had to have the ability to communicate with any standard RS232 device. Due to this fact the control lines, Request to Send ($\overline{RTS}$) and Clear to Send ($\overline{CTS}$) had to be provided. This would facilitate the necessary handshaking which might be required by certain devices. When these handshaking signals are not in use, internal resistors present in the MAX232 device pull the signals so as to hold the outputs in an inactive state.

13.3 SOFTWARE DESCRIPTION

The 8530 is initialised with the following default values:

Baud Rate:	9600
Data Bits:	8
Parity:	none
Stop Bits:	1

The software for the 8530 allows a great deal of flexibility, in that the baud rate, bit length, parity, start and stop bit formats can be changed via the HOST. This is achieved by transmitting the required initialization to the INTELLIGENT TERMINAL, which will in turn re-initialise the 8530 with the new settings. The various options are as follows:

Baud Rates:	4800, 9600, 19200, 38400
Data Bits:	7, 8
Parity:	none, even and odd
Stop Bits:	1, 2

The 8530 has the capability of being interrupt driven, but this option was not utilised. It was decided to operate the 8530 in a polled mode. This means that the command registers for channel A and channel B have to be continuously interrogated to determine if any characters have been received. If a character is available, the data register has to be read to retrieve the received character. If a character has to be transmitted, the command register must first be read to ensure that the transmitter is empty, before the character is written to the register. If this is not done, the previous character might be overwritten.

13.4 DESIGN IMPLEMENTATION

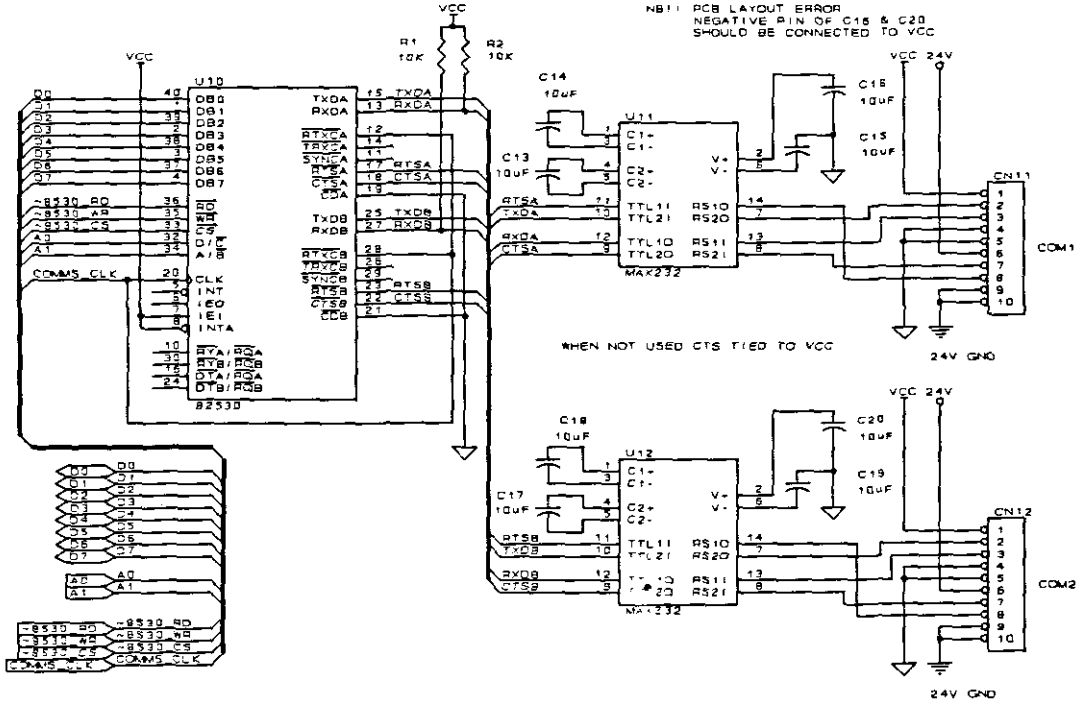


Figure 2-30. Communications Interface

The 8530 has the capability of being interrupt driven, but this option was not utilised. It was decided to operate the 8530 in a polled mode. This means that the command registers for channel A and channel B have to be continuously interrogated to determine if any characters have been received. If a character is available, the data register has to be read to retrieve the received character. If a character has to be transmitted, the command register must first be read to ensure that the transmitter is empty, before the character is written to the register. If this is not done, the previous character might be overwritten.

13.4 DESIGN IMPLEMENTATION

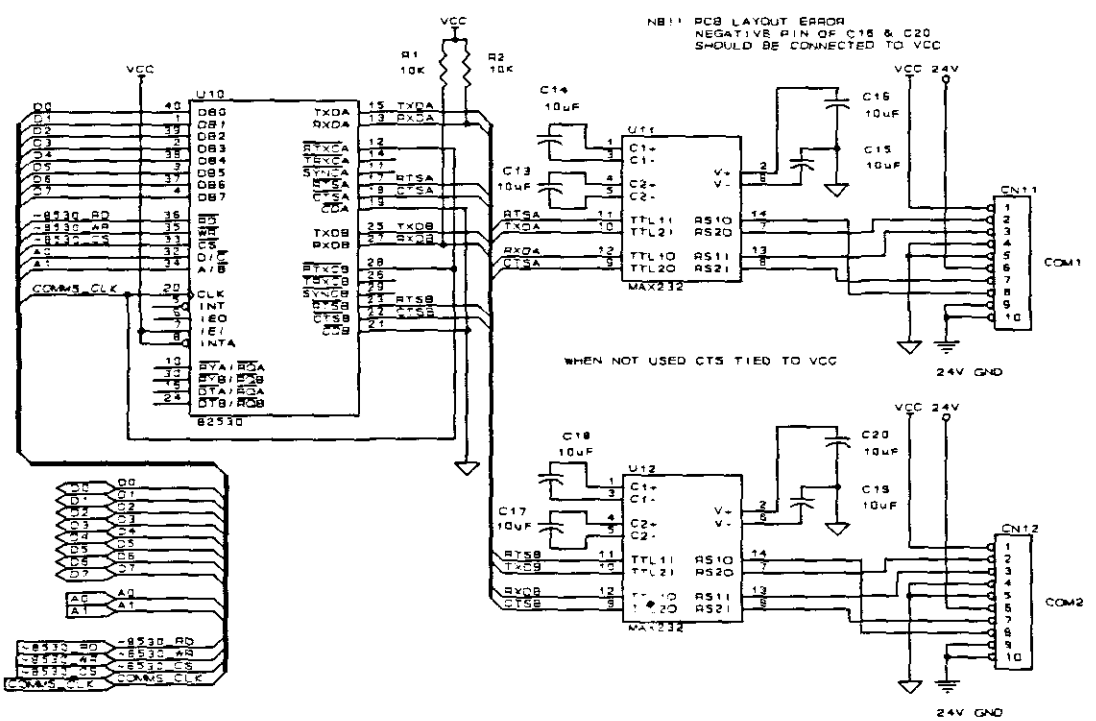


Figure 2-30. Communications Interface

14. SCANNER INTERFACE

14.1 OPTICALLY COUPLED SERIAL INTERFACE (OCIA)

The interface of the HOST system consists of four signals carried by a four-pair shielded cable, supplied by the customer, terminated at the scanner in a circular connector. Two of the signals provide bit-serial information to and from the scanner. The HOST interface controls the speeds of the received and transmitted serial data by providing clocking pulses on the remaining two signals. The signals on the interface are optically isolated in the scanner on the received end. Electrical isolation of the interface can be obtained by providing one optically isolated receiver in the HOST system.

14.2 INTERFACE SIGNALS

Three signals are inputs to the scanner and one is an output from the scanner. The signals pass information by controlling current flow through the pair of wires for each signal. A logical '1' is indicated when current flows through the pair.

14.2.1 RECEIVE DATA SIGNAL (RDATA)

The RDATA signal is used to serially transmit a byte (8 bits of

information) from the scanner to the HOST system. The RDATA RTN line provides a current source at 5 volts with respect to the scanner ground. The RDATA line normally stays at 5 volts, not sinking any current (logical '0'). A logical '1' is achieved by the RDATA line going low, thus sinking current through the signal pair. When the scanner is ready to transmit a byte (See Figure 2-32 for timing), the RDATA line will be brought low to a logical '1'. The HOST system must then provide 9 clock pulses on the CLKIN signal to receive the data bits. On the leading edge of the first clock pulse, the scanner will set RDATA to the negative logic value of the least significant data bit (D0). Bits D1 through D7 are provided after each of the successive seven clock pulses. The ninth clock pulse will cause RDATA to go back to the logic '0' state (no data ready). When the next byte is ready for transmission, the RDATA signal will be brought low again.

14.2.2 RECEIVE DATA CLOCK (CLKIN)

The CLKIN signal is used to serially clock each bit of information out the scanner as described in 12.2.1. The CLKIN signal normally is at logic '0' with no current flowing in the pair. In normal usage, CLKIN RTN is connected to +5 volts and CLKIN is driven by an open collector TTL gate capable of sinking at least 48mA. A logic '1' is the active state of the driver gate.

14.2.3 RECEIVE RESET SIGNAL (RESETIN)

The reset signal is used to re-initialize the scanner as on power-up. The Resetin signal normally is at logic '0' with no current flowing in the pair. In normal usage, Reset RTN is connected to +5 Volts and Resetin is driven by an open collector TTL gate capable of sinking at least 48mA. A logic '1' is the active state of the driver gate.

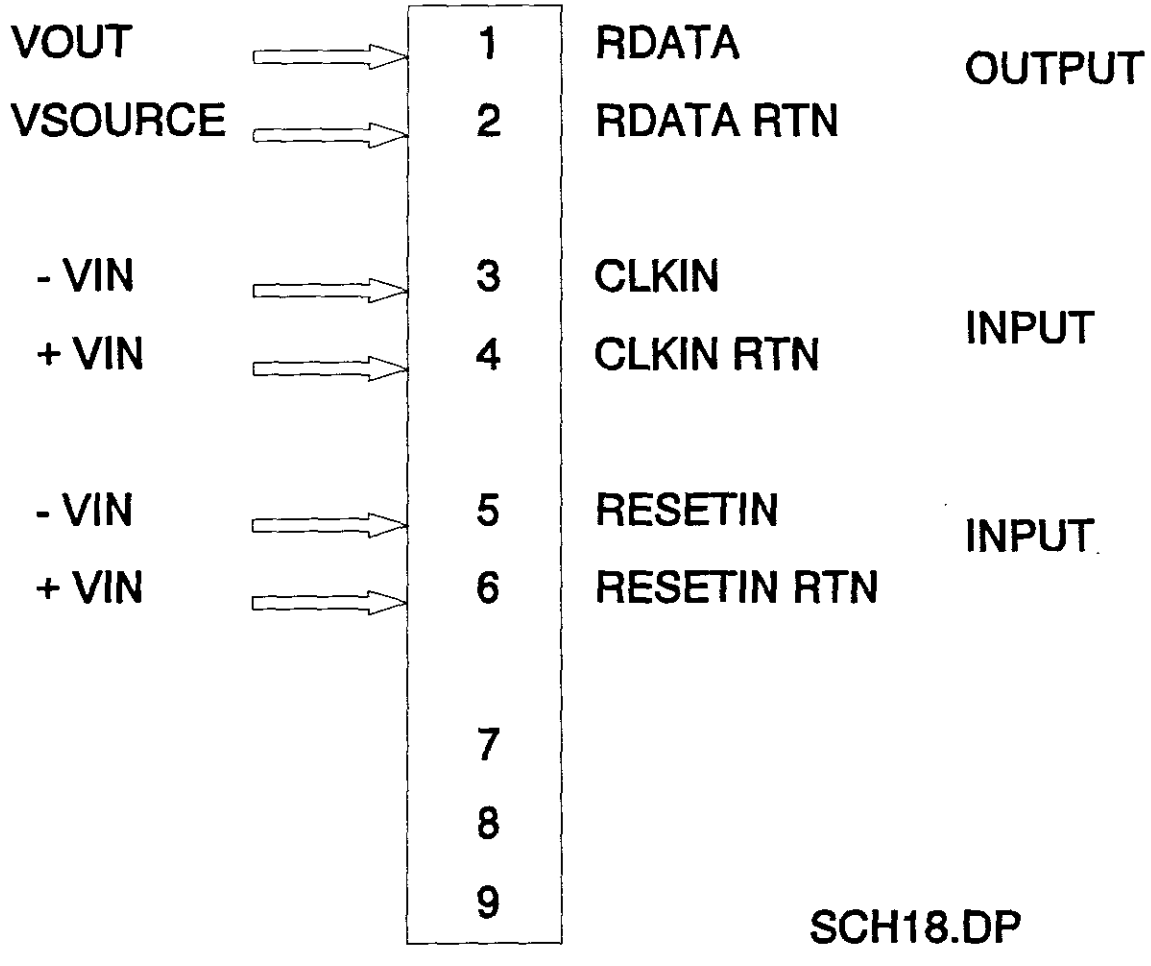


Figure 2-31. Scanner Connection

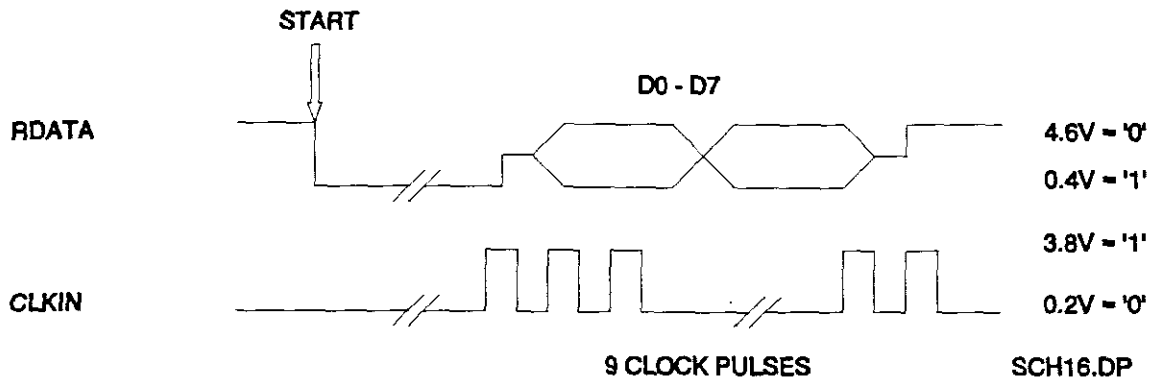


Figure 2-32. Byte Transfer From Scanner

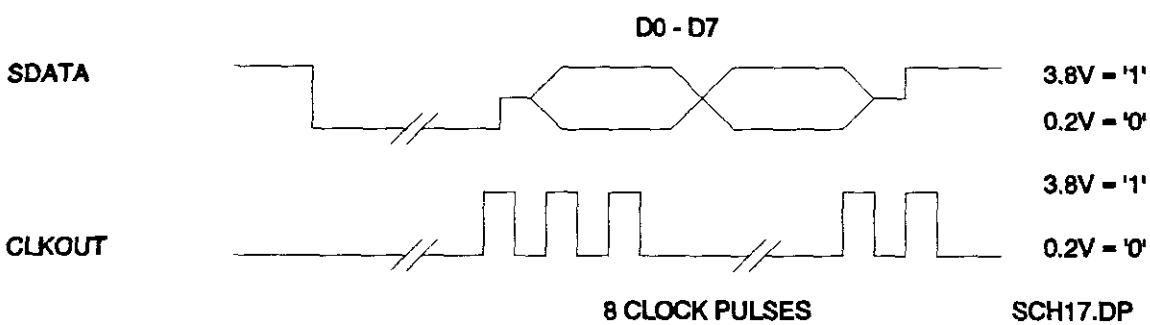


Figure 2-33. Byte Transfer To Scanner

14.3 MESSAGE FORMATS

The output of the scanner consists of numeric sequences read from a coded symbol. These are transmitted over the RDATA signal as a sequence of bytes. The general format is a version identifier character followed by the numeric characters from the symbol.

14.4 BYTE FORMATS

Each character in the label is transmitted in one byte (8 bits). The byte consists of a 6 bit data field, a bit to indicate the last byte of message, and a parity bit to achieve odd parity over the 8 bits. Table 2-12 provides the binary codes for the possible characters transmitted by the scanner.

D7	D6	D5	D4	D3	D2	D1	D0	INFORMATION
P	L	1	1	0	0	0	0	0
P	L	1	1	0	0	0	1	0
P	L	1	1	0	0	1	0	2
P	L	1	1	0	0	1	1	3
P	L	1	1	0	1	0	0	4
P	L	1	1	0	1	0	1	5
P	L	1	1	0	1	1	0	6
P	L	1	1	0	1	1	1	7
P	L	1	1	1	0	0	0	8
P	L	1	1	1	0	0	1	9
P	L	0	0	0	0	0	1	A (NORMAL UPC ID)
P	L	0	0	0	0	1	0	B (RESERVED)
P	L	0	0	0	0	1	1	C (RESERVED)
P	L	0	0	0	1	0	0	D (RESERVED)
P	L	0	0	0	1	0	1	E (0 SUPPRESS ID)
P	L	0	0	0	1	1	0	F (EAN LABEL ID)

L:

D7	D6	D5	D4	D3	D2	D1	D0	INFORMATION
P	0	X	X	X	X	X	X	NOT LAST BYTE
P	1	X	X	X	X	X	X	LAST BYTE

P:

D7	D6	D5	D4	D3	D2	D1	D0	INFORMATION
0	L	X	X	X	X	X	X	ODD PARITY
1	L	X	X	X	X	X	X	EVEN PARITY

X = don't care

Table 2-12. Byte Format

14.5 SYMBOL MESSAGE FORMATS

These are four different formats for the string sent for each symbol. These formats correspond to the regular UPC symbols (Version A) the zero-suppressed UPC symbol (Version E) and the long and short forms of the EAN symbol.

14.6 DESIGN IMPLEMENTATION

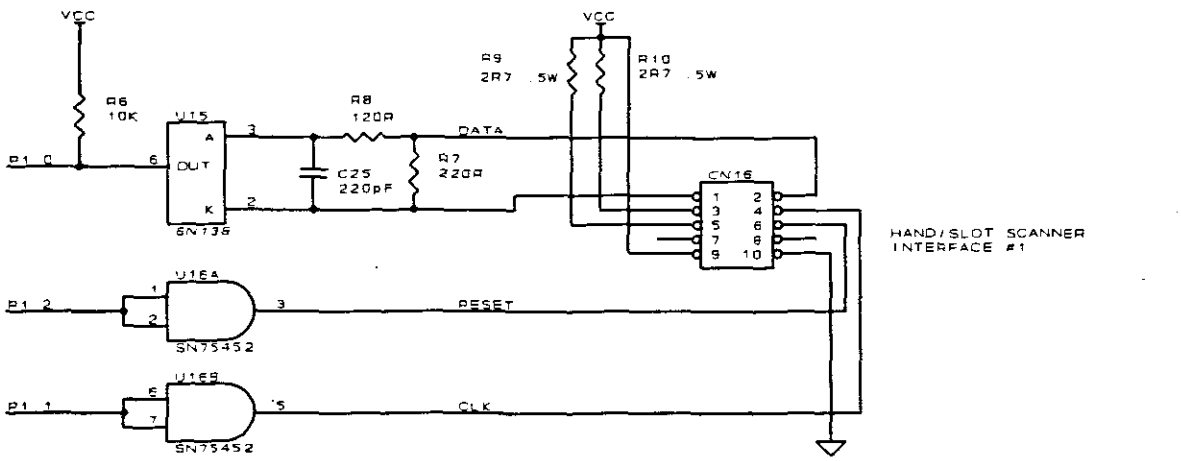


Figure 2-34. Scanner Interface

The 6N136 was found to be a suitable opto-coupler for interface purposes of receiving information from the scanner. The SN75452 is an open-collector device, which is used to drive the signals to the scanner. As the output of the 6N136 is an open-collector output, it was possible to logically 'OR' the two 6N136's, thereby using only one port pin. The same applied to the RESET and CLK signals.

15. DRAWER INTERFACE

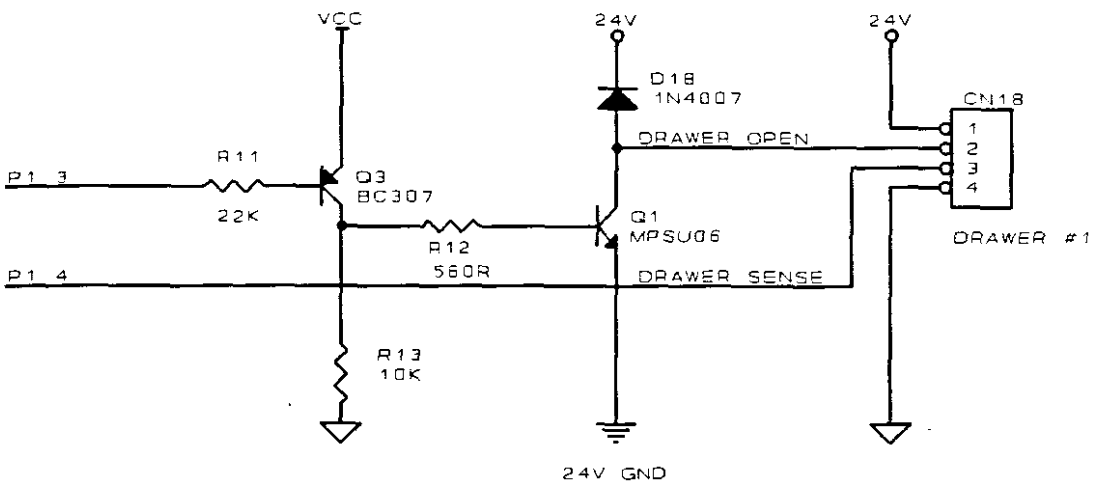


Figure 2-35. Drawer Interface

The drawer open signal is generated from a port pin. The signal is first buffered by a small signal transistor, which in turn drives the power transistor. The diode is included to prevent any damaged which might be caused to the power transistor by the back EMF generated by the solenoid. The solenoid has a DC resistance of 20Ω. The drawer sense is directly connected to a port pin. This signal passes through a micro switch, which is located in the drawer. If the status of the drawer changes, this change in status is immediately communicated to the HOST. As can be seen from the above figure, the circuit is identical for both drawers.

16. *RECEIPT SENSE*

The receipt sense is directly connected to port pin P1.7. If the status of the receipt switch changes, this change in status is immediately communicated to the HOST. This switch is used to either enable or disable the receipt printing on the printer.

17. PRESENT AND FUTURE DEVELOPMENT

Due to the constant undertaking of research programs, Ankerdata is kept at the forefront of technology, the following is a list of a few of the present projects:

1. INTELLIGENT TERMINAL re-design

   Flash Memory - in a PLCC Package

   16MHz 8032 - in a PLCC Package

   Reduced Board Space

   Reduced Connector Count

2. ANKERdata Motherboard

   Integrate ANKERdata specific I/O functions with a SCAT motherboard

3. Memory Expansion Card

   16 Bit Memory Expansion Card with 4 MBytes of Battery Backed Static RAM, to be used as Expanded memory

4. SDLC Communications

5. Third generation Vacuum Fluorescent Displays

6. Improved power failure detection circuitry

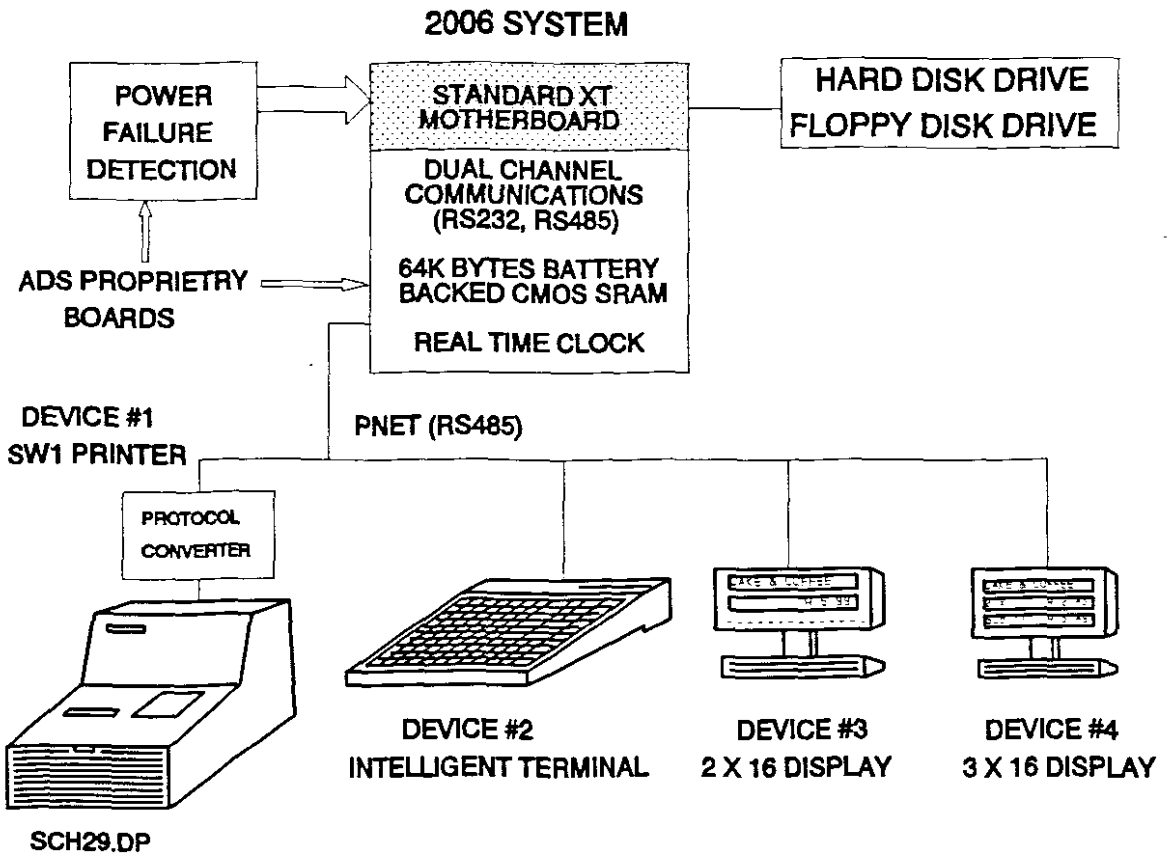


Figure 2-36. 2006 System (Metro)

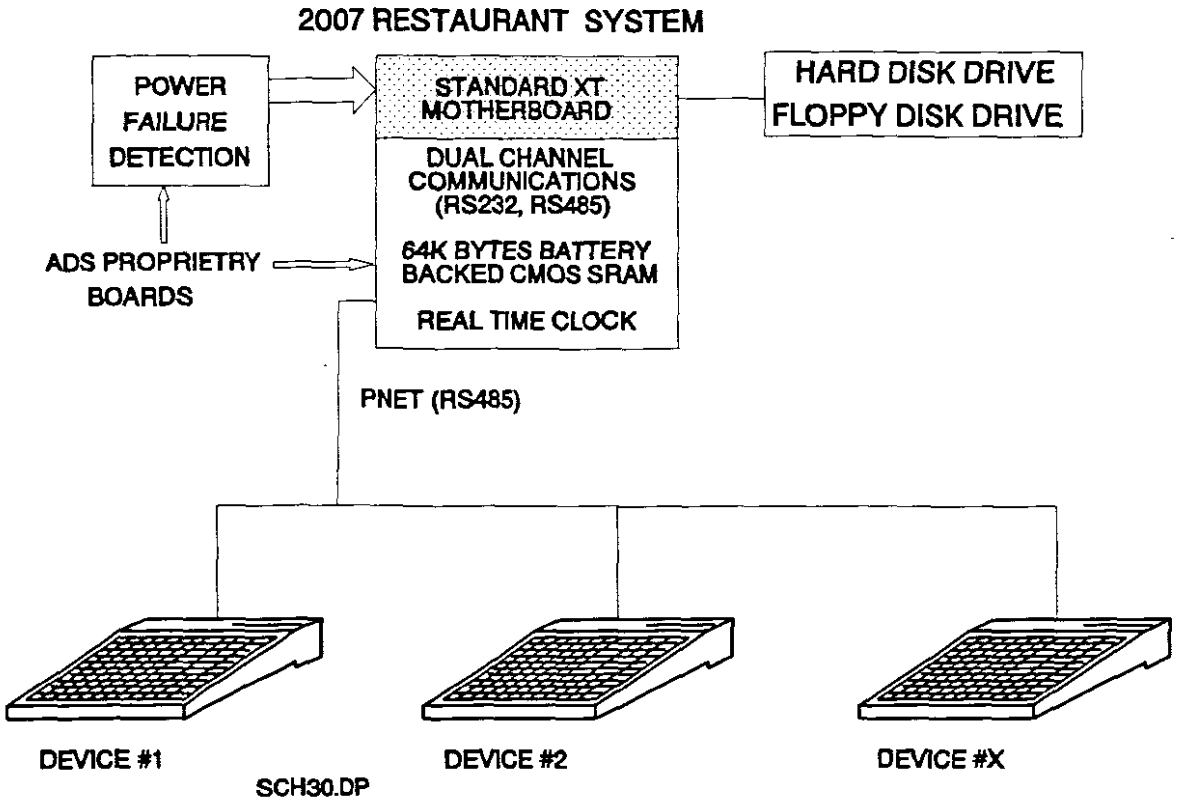
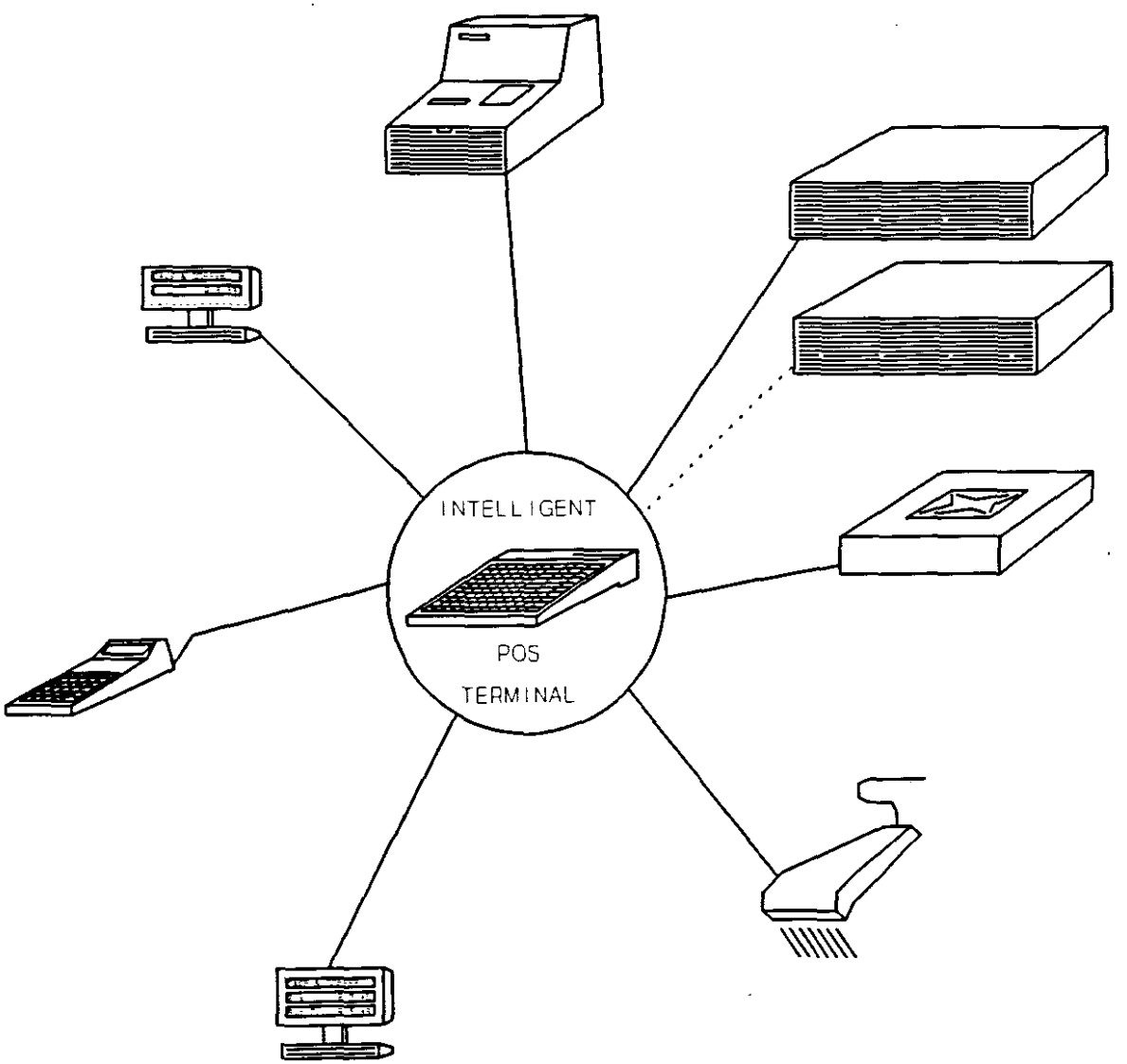


Figure 2-37. 2007 System (Restaurant)

INTELLIGENT POINT OF SALE TERMINAL

ASSEMBLER CODE LISTING



ER 3

BLER CODE LISTING

B.ASM

MAIN_MODULE

```
**************************************************************************
C  COUNT, TEMP_BUF, KEY_FLAG, KEY_BUF, LED_BUF, LED_FLAG, LED_COUNT1
C  LED_COUNT2, LOCK_STAT, LOCK_FLAG, DRAW1_FLAG, DRAW1_STAT, DRAW2_FLAG
C  DRAW2_STAT, KEY_COUNT, TX_BUF, RX_BUF, TX_BUF_FULL, RX_BUF_FULL,
UF
C  TX_COUNT, RX_COUNT, LCD_FLAG, ADDR1, RX_MODE, LCD_NUM, LCD_COUNT
C  ADDR2, TX_DATA, RX_BCC, TX_BCC, BCC_FLAG, IS_DATA, RE_TX_FLAG
C  RX_DLE, BIN_ADDR, TMOUT, CRD_FLAG, CRD_BUF, CRD_COUNT, SCAN_BUF
C  SCAN_FLAG, SCAN_PARITY, SCAN_COUNT, QUE_COUNT, QUE_FLAG, KEY_QUE
C  QUE_FULL, CRD_ON_OFF, REC_FLAG, REC_STAT

   CODE (INIT_HARDWARE, LCD_WR, CODE_TO_LCD, BEEP, TEST, KYB_READ)
   CODE (IDATA_TO_TEMP, HEX_TO_ASCII, HEX_TO_DEC, LED_ON_OFF, LED_FLASH)
   CODE (LOCK_READ, DRAW1_OPEN, DRAW1_READ, TEST_COMMS, CHK_TX_BUFFS)
   CODE (CLS1, CLS2, CHK_RX_BUFFS, TX_STRING, CRD_READ, SCAN_READ)
   CODE (ADD_QUE, CHK_QUE, REC_READ, DRAW2_OPEN, DRAW2_READ, ERR_BEEP)

****    82530 COMMS CHIP
IC  TX82_STAT, COUNT82, TXBUF1_FULL, TXBUF2_FULL, RXBUF1_FULL,
```

```
2_FULL
;  COM1_TXBUF, COM1_RXBUF, COM2_TXBUF, COM2_RXBUF, RX1_COUNT, RX2_COUNT
;  PARITY, ST_BITS, TX_DATA_BITS, RX_DATA_BITS, BAUD
;  RTSA,RTSB

   CODE (INIT_82530, A82530, B82530, TX_COMX, CODE_TO_XDATA)
   CODE (INIT_82530VARS, SETUP_TX, SETUP_RX)

***    HOST ENQUIRY OF TERMINAL STATUS, LOCK, DRAW, RECEIPT
   CODE (ENQUIRE)

C  ENQ_FLAG, ENQ_BUF, ENQ_FULL

;*************************************************************************

;*************************************************************************
DATA SEGMENT                                                            *
;*************************************************************************
_SEG    SEGMENT     DATA
        RSEG        DATA_SEG        ;RELOCATABLE INTERNAL DATA

K:      DS          60              ;60 BYTES FOR THE STACK
K_END:                              ;USED TO CHECK FOR STACK OVERFLOW
_BUF:   DS          01
T:      DS          01              ;BYTE COUNTER
COUNT1: DS          01              ;INDICATES WHEN LED_FLASH SHOULD
                                    ;BE CALLED, TO FLASH LEDS
```

```
OUNT2:  DS      01              ;REQUIRE A 16 BIT COUNTER
UM:     DS      01              ;0 = BOTH LCD'S, 1 = LCD_1, 2 = LCD_2
OUNT:   DS      01              ;COUNT # OF BYTES WRITTEN TO LCD_BUF
OUNT:   DS      01              ;COUNT # OF BYTES IN KEY_BUF
UNT:    DS      01              ;NUMBER OF BYTES IN TX_BUFF
UNT:    DS      01              ;NUMBER OF BYTES IN RX_BUFF
DE:     DS      01              ;COMMS MODE
AR:     DS      01              ;RECEIVED CHAR
ADDR:   DS      01              ;BINARY VALUE OF TERMINAL ADDR
1:      DS      01              ;DEVICE ADDR IS A 2 BYTE ASCII VALUE
2:      DS      01
C:      DS      01              ;RX_BUF BCC COUNTER
C:      DS      01              ;TX_BUF BCC COUNTER
OUNT:   DS      01              ;USED TO COUNT BYTES IN CHK_TX_BUFFS
_COUNT: DS      01              ;COUNT # OF BYTES SCANNED
COUNT:  DS      01              ;COUNT # OF BYTES IN KEYQUE

IABLES RELATING TO 82530 COMMS CHIP
:       DS      01              ;TO TOGGLE RTS LINE, HAVE TO WR TO

:       DS      01              ;BUT THIS REG CAN'T BE READ,
EFORE
                                ;KEEP IMAGE OF THE VALUE FOR THIS REG
_STAT:  DS      01              ;TRANSMISSION SUCCESSFUL
K,0FFH=BAD
T82:    DS      01              ;# OF BYTES TO WRITE TO 82530 COULD
COUNT:  DS      01              ;# OF BYTES IN COM1_RXBUF
```

```
UNT:    DS          01              ;# OF BYTES IN COM2_RXBUF
:       DS          01              ;FOLLOWING VARIABLES USED WHEN
S:      DS          01              ;82530 SETUP FROM HOST
        DS          01
'A_BITS:    DS      01
'A_BITS:    DS      01

_CNT:       DS      01              ;COMMS RETRY COUNTER

;*****************************************************************************

;*****************************************************************************
IT SEGMENT                                                                  *
;*****************************************************************************
EG      SEGMENT     BIT
        RSEG        BIT_SEG         ;RELOCATABLE BIT SPACE, INTERNAL

LAG:    DBIT        01              ;HOST HAS DONE ENQUIRY TO TERMINAL
ULL:    DBIT        01              ;ENQ_BUF IS FULL, TX THIS TO HOST
LAG:    DBIT        01              ;INDICATES KET DATA IN KEY_BUF
LAG:    DBIT        01              ;INDICATES LED DATA AVAILABLE
_FLAG:  DBIT        01              ;LOCK STATUS HAS CHANGED
_FLAG:  DBIT        01              ;THIS INDICATES THAT DRAW1 STATUS
                                    ;HAS CHANGED
_FLAG:  DBIT        01              ;THIS INDICATES THAT DRAW2 STATUS
                                    ;HAS CHANGED
LAG:    DBIT        01              ;THIS INDICATES THAT RECEIPT ON/OFF
```

```
                                        ;STATUS HAS CHANGED
 FLAG:  DBIT        01                  ;TX_BUF HAS TO BE RE-TRANSMITTED
 AG:    DBIT        01                  ;DATA IN LCD_BUFFER
 AG:    DBIT        01                  ;DLE FOUND
 L:     DBIT        01                  ;POL = 0, SEL = 1
 :      DBIT        01                  ;0 = ACK0, 1 = ACK1
 A:     DBIT        01                  ;DATA IN INPUT BUFFERS
 :      DBIT        01                  ;INDICATES THAT A DLE WAS RECEIVED
        DBIT        01                  ;TIMER 0, TIMESOUT BEFORE LCD IS


 AG:    DBIT        01                  ;1 = CARD DATA AVAILABLE
 LAG:   DBIT        01                  ;INFORMS MAIN LOOP OF SCANNER DATA
 AG:    DBIT        01                  ;DATA IN KEY_QUE TO BE TRANSMITTED
 LL:    DBIT        01                  ;KEY_QUE IS FULL, PREVENT KEY & SCAN
 _OFF:  DBIT        01                  ;0=DON'T TX CRD DATA. 1 = TX CRD DATA
 ARITY:     DBIT    01                  ;USED FOR PARITY CHECK
 _FULL:     DBIT    01                  ;TX BUFFER IS FULL
 _FULL:     DBIT    01                  ;RX BUFFER IS FULL

ABLES RELATING TO 82530 COMMS CHIP
1_FULL:     DBIT    01                  ;DATA FOR COM1 OF 82530
2_FULL:     DBIT    01                  ;DATA FOR COM2 OF 82530
1_FULL:     DBIT    01                  ;DATA FROM COM1 OF 82530
2_FULL:     DBIT    01                  ;DATA FROM COM2 OF 82530
ITS
*****************************************************************************
```

```
*************************************************************************
TERNAL DATA SEGMENT                                                     *
*************************************************************************
:G      SEGMENT     XDATA
        RSEG        EXT_SEG         ;RELOCATABLE XTERNAL DATA SEGMENT
        ORG         0

:F:     DS          04              ;LED BUFFER
:TAT:   DS          01              ;LOCK STATUS
_STAT:  DS          01              ;DRAW1 STATUS, OPEN = 1, CLOSED = 0
_STAT:  DS          01              ;DRAW2 STATUS, OPEN = 1, CLOSED = 0
:AT:    DS          01              ;REC STATUS, OPEN = 1, CLOSED = 0
JF:     DS          08              ;8279 INTERNAL BUFFER = 8 BYTES
:       DS          50              ;COMMS BUFFER
:A:     DS          50              ;TRANSMIT DATA BUFFER
JF:     DS          40              ;40 CHARS FOR LCD
JF:     DS          41              ;STATUS+40 DATA BYTES. 01=OK,
3AD
BUF:    DS          24              ;18 BYTES OF DATA + 1 BYTE STATUS
UE:     DS          25              ;ADD KEY & SCAN DATA INTO THIS BUFFER
UF:     DS          04              ;USED FOR STATUS BYTES FOR HOST
ST
 BYTES

*******     R0/1 CAN ADDRESS MAX OF 255 BYTES EXTERNAL DATA MEMORY

F:      DS          255             ;MAX RX BUFFER SIZE
```

```
ABLES RELATING TO 82530 COMMS CHIP

TXBUF: DS          50              ;ALLOW FOR 2X20 DISPLAY STRING
RXBUF: DS          40              ;DON'T FORGET STX, ETX, DEV #
CXBUF: DS          50
RXBUF: DS          40

*****************************************************************************

*****************************************************************************
       CODE SEGMENT                                                        *
*****************************************************************************
       CSEG        AT 0            ;ABSOLUTE SEGMENT FOR MAIN MODULE

       ORG         0
       USING       1               ;USING RB 0,1
       LJMP        MAIN

UDE (EQUATES.ASM)

*****************************************************************************
INTERRUPT VECTORS                                                          *
*****************************************************************************
       ORG         03H             ;VECTOR ADDR OF EXTERNAL INT0
       SJMP        EXT0_INT

       ORG         0BH             ;TIMER_0 VECTOR ADDRESS
```

```
        SJMP    TIMER0_INT

        ORG     13H             ;VECTOR ADDR OF EXTERNAL INT1
        SJMP    EXT1_INT

        ORG     1BH             ;TIMER_1 VECTOR ADDRESS
        SJMP    TIMER1_INT

        ORG     0023H           ;VECTOR ADDR OF SERIAL INT
        SJMP    COMMS_INT

        ORG     2BH             ;TIMER_2 VECTOR ADDRESS
        SJMP    TIMER2_INT

*************************************************************************
NTERRUPT ROUTINES                                                       *
IMER INTERRUPT VECTOR ADDR                                              *
*************************************************************************
0_INT:                          ;TF0 IS RESET BY H/W WHEN VECTORING
        CLR     TR0             ;STOP TIMER_0
        SETB    TMOUT           ;TIMER HAS OVERFLOWED

        RETI
*************************************************************************
```

```
;*************************************************************************
;ONG INTERRUPT VECTOR ADDR                                               *
;*************************************************************************
_INT:
!_INT:
:NT:
:NT:
        LCALL       CLS1
        MOV         DPTR,#MSG6
        LCALL       CODE_TO_LCD
        MOV         DPTR,#LCD1_DAT_WR
        LCALL       LCD_WR
        MOV         COUNT,#15
        LCALL       ERR_BEEP            ;SIT IN PERMANENT LOOP

        RETI
*************************************************************************


;*************************************************************************
TART COMMS INTERUPT ROUTINE. COMMS INTERRUPT VECTOR ADDR                 *
                                                                         *
X CHAR WILL CAUSE AN INT. CHECK IF STX/ETX. THEN IF BUFFER IS FULL.      *
FTER RX 40 BYTES, WAIT FOR EXT, THEN SET FLAG FOR BUFFER FULL. BUFFER    *
RANSFER CAN NOW TAKE PLACE. AFTER BUFFER TRANSFER COMPLETE, CAN START    *
0 RX MORE CHARS. SET A PB AS A CTS FOR THE HOST.                         *
--  40 CHARS ARE / INTO TUBE 1 & TUBE 2, EACH 16 CHARS                   *
0 IS USED FOR TX. WHEN SETTING UP TX_BUF, SET R0 TO START OF BUFFER      *
```

```
; IS USED FOR                                                          *
; IS USED TO COUNT RX CHARS                                            *
; IS USED TO COUNT TX CHARS                                            *
; IS USED FOR RX_MODE STATUS                                           *
; IS USED TO SAVE DPL                                                  *
; IS USED TO SAVE DPH                                                  *
PTR IS USED FOR RX_BUF, EXTERNAL DATA, OUTSIDE OFFH BOUNDRY            *
                                                                       *
X_MODE 0 = WAIT FOR STX                                                *
 1 = WAIT FOR DEVICE ADDRESS                                           *
 2 = RECEIVE CHARS IN COMBUF                                           *
SW.5 FLAG 0, IS USED TO INDICATE BUFFER FULL                           *
X_COUNT IS SETUP WHEN CONSTRUCTING TX_BUF                              *
X_COUNT INDICATES THE # OF BYTES IN RX_BUF                             *
X_BCC IS USED TO STORE THE RX CHARS BCC CALCULATION                    *
************************************************************************
_INT:
        PUSH        ACC
        PUSH        B
        PUSH        PSW
        PUSH        DPL
        PUSH        DPH
        MOV         A,P2            ;SAVE MSB OF ADDR
        PUSH        ACC

        SETB        PSW.3           ;SELECT REGISTER BANK 1, 8H - FH
        MOV         DPL,R5
```

```
        MOV         DPH,R6
        JNB         TI,RX           ;CHECK TO SEE IF ITS TI/RI

        CLR         TI              ;PREVIOUSLY TX BYTE
        MOV         A,R3
        LCALL       TEST

        DJNZ        R3,TX_CHAR
        CLR         TX_BUF_FULL     ;TX BUFFER NOW EMPTY
        SETB        P3.5            ;DS3695 RX MODE, GOES THRU 7404
        LJMP        COMMS_EXIT
AR:
        MOV         P2,#0           ;IN CONJUNCTION WITH MOVX R0/R1
        MOVX        A,@R0           ;READ TX_BUF
        MOV         SBUF,A          ;TX BYTE IN TX_BUF
        INC         R0              ;NEXT BYTE TO TRANSMIT
        LJMP        COMMS_EXIT


        MOV         A,R3            ;LAST BYTE IN BUFFER WILL STILL
                                    ;CAUSE INT
                                    ;THEREFORE TEST IF BUFFER EMPTY, VIA
                                    ;R3
                                    ;DON'T TX A BYTE, & DON'T TX ANOTHER
                                    ;BYTE IN MIDDLLE OF STRING
        ADD         A,#0            ;DIRECT ADD DOESN'T AFFECT ANY FLAGS
        JZ          EXIT            ;BUFFER EMPTY, EXIT
```

```
        INC         R0              ;NEXT BYTE IN DISPBUF, ALREADY TX 1
                                    ;BYTE
        MOV         SBUF,@R0        ;TX BYTE AT ADDR OF DISPBUF
        DEC         R3              ;48 BYTE COUNT
        SJMP        COMMS_EXIT

        MOV         A,SBUF      ;READ RX BUFFER
        CLR         RI          ;WON'T BE ANY DELAY WHEN READING A CHAR
        MOV         R4,RX_MODE  ;RX MODE STATUS

JNE     R4,#8,CHK_BCC_MODE  ;CHK IF IN BINARY ADDRESS MODE
JMP     CHK_MODE8
CC_MODE:
JNE     R4,#10,CHK_EOT1     ;CHK IF IN BCC MODE
JMP     CHK_MODE10
OT1:
JNE     A,#EOT,CHK_MODE0    ;!EOT, CHECK OTHER MODES
JNE     R4,#9,SET_MODE1     ;IF MODE 0, CHK FOR 'EOT'
JMP     CHK_MODE9
ODE1:
OV RX_MODE,#01      ;MODE = 1, WAIT FOR DEV. ADDR1
JMP     COMMS_EXIT

ODE0:
        CJNE        R4,#0,CHK_MODE1 ;IF MODE 0, CHK FOR 'EOT'
        CJNE        A,#EOT,INTERMEDIATE_EXIT_0  ;!EOT, RESET RX_MODE
        MOV         RX_MODE,#01     ;MODE = 1, WAIT FOR DEV. ADDR1
```

```
        LJMP        COMMS_EXIT

)DE1:
        CJNE        R4,#01,CHK_MODE2;IF MODE 1, CHK FOR 'ADDR1'
        CJNE        A,ADDR1,INTERMEDIATE_EXIT_0 ;!ADDR1, RESET RX_MODE
        MOV         RX_MODE,#02     ;MODE = 2, WAIT FOR DEV. ADDR2
        LJMP        COMMS_EXIT

)DE2:
        CJNE        R4,#02,CHK_MODE3;IF MODE 2, CHK FOR 'ADDR2'
        CJNE        A,ADDR2,INTERMEDIATE_EXIT_0 ;!ADDR2, RESET RX_MODE
        MOV         RX_MODE,#03     ;MODE = 3, WAIT FOR POL/SEL
        LJMP        COMMS_EXIT

ODE3:
        CJNE        R4,#03,CHK_MODE4;IF MODE 3, CHK FOR 'POL/SEL'
        CJNE        A,#POL,CHK_SEL  ;!POL, CHECK IF SEL
        CLR         POL_SEL         ;0 = POL, 1 = SEL
        MOV         RX_MODE,#04     ;MODE = 4, WAIT FOR ENQ
        LJMP        COMMS_EXIT

EL:
        CJNE        A,#SEL,INTERMEDIATE_EXIT_0  ;!SEL, RESET RX_MODE
        SETB        POL_SEL         ;0 = POL, 1 = SEL
        MOV         RX_MODE,#04     ;MODE = 4, WAIT FOR ENQ
        LJMP        COMMS_EXIT

10DE4:
        CJNE        R4,#04,CHK_MODE5;IF MODE 4, CHK FOR 'ENQ'
        CJNE        A,#ENQ,INTERMEDIATE_EXIT_0  ;!ENQ, RESET RX_MODE
                                    ;RECEIVED A VALID SEQUENCE
                                    ;POL/SEL. READY TO RX/TX
```

```
        JB          POL_SEL,SELECT  ;0 = POL, 1 = SEL
;*****************************************************************************

;*****************************************************************************
                                    ;TX_BUF_FULL, 0 = EMPTY, 1 = FULL
        JB          RE_TX_FLAG,RE_TX    ;1 = RE_TRANSMIT TX_BUF
        LCALL       CHK_TX_BUFFS    ;TRANSMIT DATA IN TX_BUF
        LCALL       TX_STRING
        JNB         TX_BUF_FULL,TX_EOT  ;NOTHING IN TX_BUFF, TX 'EOT'
        MOV         RX_MODE,#05     ;WAIT FOR ACK/NAK
        MOV         RETRY_CNT,#0
        LJMP        COMMS_EXIT
T:
        MOV         A,#EOT          ;NOTHING IN TX_BUF
        LCALL       TX_BYTE
        LJMP        EXIT_0          ;RESET RX_MODE
:
        MOV         A,#05
        CJNE        A,RETRY_CNT,RE_TX_FINAL
                                    ;AFTER 5 NAKS, TERMINATE ABNORMALLY
        CLR         RE_TX_FLAG      ;0 = NORMAL RX MODE
        SJMP        TX_EOT
(_FINAL:
        MOV         RX_MODE,#05     ;WAIT FOR ACK/NAK
        MOV         R3,TX_COUNT     ;DON'T CHANGE TX_COUNT, IN CASE RE-T>
        SETB        TX_BUF_FULL     ;CLEARED IN INT ROUTINE
        CLR         P3.5            ;ENABLE DS3695 TO TX, GOES THRU 7404
```

```
        MOV         P2,#0           ;IN CONJUNCTION WITH MOVX R0/R1
        MOV         R0,#TX_BUF      ;TX FIRST BYTE IN TX_BUF
        MOVX        A,@R0
        MOV         SBUF,A          ;START TX PROCESS
        MOV         R0,#TX_BUF+1    ;SET R0 = START ADDR +1  OF TX_BUF
                                    ;USED IN COMMS INT ROUTINE. FIRST
                                    ;BYTE IS USED TO INITIATE COMMS INT
        LJMP        COMMS_EXIT
;*************************************************************************
MEDIATE_EXIT_0:
        LJMP        EXIT_0
;*************************************************************************
T:
        JB          RX_BUF_FULL,TX_NAK  ;NOTHING IN RX_BUFF, TX 'ACK'
        MOV         A,#ACK          ;READY TO RECEIVE DATA
        LCALL       TX_BYTE
        MOV         RX_MODE,#06     ;MODE = 6, WAIT FOR DLE,STX
        LJMP        COMMS_EXIT
K:
        MOV         A,#NAK          ;TX_BUF IS FULL
        LCALL       TX_BYTE
        LJMP        EXIT_0          ;RESET RX_MODE, WAIT FOR NEXT POLL
;*************************************************************************
```

```
**************************************************************************
ODE5:
        CJNE        R4,#05,CHK_MODE6     ;IF MODE 5, CHK IF ACK/NAK
        CJNE        A,#ACK,CHK_NAK  ;!ACK, CHK IF A NAK
        MOV         A,#EOT          ;IF ACK, TRANSMIT 'EOT' RESET RX_MODE
        LCALL       TX_BYTE
        CLR         RE_TX_FLAG
        SJMP        EXIT_0          ;DATA RECEIVED NORMALLY
AK:
                                    ;TREAT ALL CHARS, AS A NAK
        SETB        TX_BUF_FULL     ;MAINTAIN CURRENT CONTETNTS OF BUFFER
                                    ;WAIT FOR NEXT POLL TO TX MESAGE
        SETB        RE_TX_FLAG
        INC         RETRY_CNT

        CJNE        A,#EOT,EXIT_0   ;EOT, SET RX_MODE
        MOV         RX_MODE,#01     ;MODE = 1, WAIT FOR DEV. ADDR1
        LJMP        COMMS_EXIT
        SJMP        EXIT_0
ODE6:
        CJNE        R4,#06,CHK_MODE7     ;IF MODE 6, CHK IF DLE
        CJNE        A,#DLE,EXIT_0   ;!DLE, RESET RX_MODE
        MOV         RX_MODE,#07     ;MODE = 7, WAIT FOR STX
        SJMP        COMMS_EXIT
ODE7:
        CJNE        R4,#07,CHK_MODE8     ;IF MODE 7, CHK IF STX
        CJNE        A,#STX,EXIT_0   ;!STX, RESET RX_MODE
```

```
        MOV         RX_MODE,#08     ;RX_MODE = 8, WAIT FOR BIN TERM ADDR
        MOV         RX_COUNT,#0
        MOV         RX_BCC,#0           ;BCC COUNTER
        CLR         RX_DLE
        MOV         DPTR,#RX_BUF
        SJMP        COMMS_EXIT
ODE8:
        CJNE        R4,#08,CHK_MODE9    ;IF MODE 8, CHK BINARY ADDR
CT
        CJNE        A,BIN_ADDR,EXIT_0   ;!TERM ADDR, RESET RX_MODE
        MOV         RX_MODE,#09         ;WAIT FOR TEXT
        XRL         A,RX_BCC
        XCH         A,RX_BCC            ;SAVE BCC INTO RX_BCC
        SJMP        COMMS_EXIT
ODE9:
        CJNE        R4,#09,CHK_MODE10   ;IF MODE 9, CHK IF DLE.DLE,
TX
        JB          RX_DLE,CHK_ETX
        CJNE        A,#DLE,CALC_RX_BCC  ;RECEIVED A DLE
        SETB        RX_DLE          ;WAIT FOR A DLE/ETX
        SJMP        COMMS_EXIT
TX:
        CJNE        A,#ETX,CALC_RX_BCC  ;END OF DATA FIELD
        XRL         A,RX_BCC
        XCH         A,RX_BCC        ;SAVE BCC INTO RX_BCC
        MOV         RX_MODE,#10     ;MODE = 10 CMP NEXT BYTE TO RX_BCC
        SJMP        COMMS_EXIT
```

```
RX_BCC:
        MOVX        @DPTR,A          ;WRITE TO RX_BUF
        INC         DPTR
        INC         RX_COUNT
        XRL         A,RX_BCC
        XCH         A,RX_BCC         ;SAVE BCC INTO RX_BCC
        CLR         RX_DLE
        SJMP        COMMS_EXIT
ODE10:
        CJNE        R4,#010,EXIT_0   ;IF MODE 10, CHK IF RX_BCC IS CORRECT

        CJNE        A,RX_BCC,TX_NAK ;CHECK IF BCC IS CORRECT
        SETB        RX_BUF_FULL
        MOV         A,#ACK
        LCALL       TX_BYTE
        SJMP        EXIT_0
_0:
        MOV         RX_MODE,#0       ;SET RX_MODE = 0, WAIT FOR EOT
S_EXIT:
        POP         ACC
        MOV         P2,A             ;RESTORE MSB OF ADDR
        MOV         R5,DPL
        MOV         R6,DPH
        POP         DPH
        POP         DPL
        POP         PSW
        POP         B
```

```
        POP         ACC
        RETI
;*******************************************************************

;*******************************************************************
NB:                                                                 *
     REGISTER BANK 01 IS SELECTED FOR THIS ROUTINE, WHICH IS DEDICATED*
     TO THE COMMS INTERRUPT ROUTINE.                                *
                                                                    *
TRANSMIT A SINGLE BYTE. BE CAREFUL NOT TO CHANGE TX_COUNT           *
                                                                    *
ON ENTRY:                                                           *
          ACC = BYTE TO BE TRANSMITTED                              *
;*******************************************************************
TE:
          MOV           TX_COUNT,#01    ;USED IN COMMS INT ROUTINE
          MOV           R3,TX_COUNT     ;DON'T CHANGE TX_COUNT, IN CASE
                                        ;RE-TX

          MOV       R3,#01          ;DON'T CHANGE TX_COUNT, IN CASE RE-TX
          SETB      TX_BUF_FULL     ;CLEARED IN INT ROUTINE
          CLR       P3.5            ;ENABLE DS3695 TO TX, GOES THRU 7404
          MOV       SBUF,A          ;START TX PROCESS

          RET
;*******************************************************************
```

```
**************************************************************************
AIN PROGRAM LOOP                                                         *
**************************************************************************

        CLR         EA              ;ALL INTERRUPTS OFF
        MOV         SP,#STACK       ;SET SP,DEFAULT 08H, USING RBANK1,
        LCALL       INIT_HARDWARE
        LCALL       TEST_COMMS
        CLR         ACK0_1              ;INITIALLY ACK0
        SETB        EA                  ;ALL INTERRUPTS ON
        LCALL       INIT_STACK_CHK      ;CHK FOR STACK OVERFLOW
        LCALL       FILL_MEM
_COMMS:
        LCALL       INIT_82530VARS
_CH_A:
        MOV         DPTR,#COM1_TXBUF    ;SETUP REGS FOR CODE_TO_XDATA
        MOV         R6,DPL
        MOV         R7,DPH
        MOV         DPTR,#A82530
        MOV         COUNT82,#22
        LCALL       CODE_TO_XDATA       ;COM1_TXBUF CONTAINS
ALIZATION
        MOV         DPTR,#COM1_TXBUF    ;INIT CHANNEL A OF 82530
        MOV         R7,#03
        MOV         COUNT82,#22
        LCALL       INIT_82530
_CH_B:
```

```
        MOV         DPTR,#COM2_TXBUF    ;SETUP REGS FOR CODE_TO_XDATA
        MOV         R6,DPL
        MOV         R7,DPH
        MOV         DPTR,#B82530
        MOV         COUNT82,#22
        LCALL       CODE_TO_XDATA       ;COM2_TXBUF CONTAINS
LIZATION
        MOV         DPTR,#COM2_TXBUF    ;INIT CHANNEL B OF 82530
        MOV         R7,#01
        MOV         COUNT82,#22
        LCALL       INIT_82530
T_STR:
        MOV         DPTR,#COM2_TXBUF    ;SETUP REGS FOR CODE_TO_XDATA
        MOV         R6,DPL
        MOV         R7,DPH
        MOV         DPTR,#MSG2
        MOV         COUNT82,#35
        LCALL       CODE_TO_XDATA
R:
        MOV         DPTR,#COM2_TXBUF    ;TRANSMIT DATA ON COM2
        MOV         R7,#02              ;TX ON CH B
        MOV         COUNT82,#35
        LCALL       TX_COMX
T:
82530:
        LCALL       SETUP_TX            ;IF DATA, THEN TX
        LCALL       SETUP_RX            ;IF DATA, THEN RX
```

```
NQUIRE:
        LCALL   ENQUIRE                 ;HOST REQUEST DRAW STATUS, LOCK
EC:
        LCALL   REC_READ                ;STATUS 0=CLOSED, 1=OPEN

        MOV     A,#'S'
        LCALL   CHK_QUE
        LCALL   SCAN_READ               ;IF QUE_FULL, WON'T READ SCANNER
        JNB     SCAN_FLAG,LCD           ;IF 1, DATA AVAILABLE
        MOV     A,#'S'
        LCALL   ADD_QUE
        CLR     SCAN_FLAG
        LCALL   CLS1
        MOV     R2,#16                  ;SETUP REGS, BEFORE CALLING XDAT
        MOV     R2,#20                  ;SETUP REGS, BEFORE CALLING XDAT
        MOV     DPTR,#SCAN_BUF
        LCALL   XDAT_TO_LCD_BUF
        MOV     DPTR,#LCD1_DAT_WR
        LCALL   LCD_WR

        JNB     LCD_FLAG,LOCK
        CLR     LCD_FLAG
        MOV     A,LCD_NUM
        CJNE    A,#30H,CHK_LCD1
        LCALL   CLS1
        MOV     DPTR,#LCD1_DAT_WR
        LCALL   LCD_WR
```

```
        LCALL       CLS2
        MOV         DPTR,#LCD2_DAT_WR
        LCALL       LCD_WR
        SJMP        LOCK
CD1:
        CJNE        A,#31H,CHK_LCD2
        LCALL       CLS1
        MOV         DPTR,#LCD1_DAT_WR
        LCALL       LCD_WR
        SJMP        LOCK
CD2:
        CJNE        A,#32H,LOCK
        LCALL       CLS2
        MOV         DPTR,#LCD2_DAT_WR
        LCALL       LCD_WR
:
        LCALL       LOCK_READ           ;READ LOCK STATUS
        JNB         LOCK_FLAG,CHK_CRD       ;STATUS HASN'T CHANGED
        LCALL       CLS1
        MOV         DPTR,#LOCK_STAT
        MOVX        A,@DPTR
        LCALL       HEX_TO_ASCII
        MOV         DPTR,#LCD1_DAT_WR
        LCALL       LCD_WR
CRD:
        LCALL       CRD_READ
        JNB         CRD_FLAG,KYB
```

```
        CLR         CRD_FLAG
        LCALL       CLS1
        MOV         R2,#32              ;SETUP REGS, BEFORE CALLING XDAT
        MOV         DPTR,#CRD_BUF
        LCALL       XDAT_TO_LCD_BUF
        MOV         DPTR,#LCD1_DAT_WR
        LCALL       LCD_WR

        MOV         A,#'K'
        LCALL       CHK_QUE
        LCALL       KYB_READ
        JNB         KEY_FLAG,CHK_FLASH
        MOV         A,#'K'
        LCALL       ADD_QUE
        LCALL       CLS1
        MOV         R0,#KEY_BUF
        MOV         P2,#0
        MOVX        A,@R0
        LCALL       HEX_TO_DEC
        MOV         DPTR,#LCD1_DAT_WR
        LCALL       LCD_WR
FLASH:
        DJNZ        LED_COUNT1,CHK_DRAW
        DJNZ        LED_COUNT2,CHK_DRAW
        LCALL       LED_FLASH
DRAW:
        LCALL       DRAW1_READ              ;READ DRAW STATUS
```

```
        LCALL       DRAW2_READ              ;READ DRAW STATUS
        MOV         DPTR,#DRAW_STAT         ;READ DRAW STAT
        MOVX        A,@DPTR
        CJNE        A,#01,EXIT1             ;DRAW OPEN, LED ON
        ORL         LED_BUF+1,#01           ;LED_ON
        SJMP        EXIT_START
:
        ANL         LED_BUF+1,#0EH          ;LED OFF
START:
        LCALL       CHK_RX_BUFFS
        LCALL       CHK_RAM
        LCALL       CHK_STACK               ;CHK FOR STACK OVERFLOW
        LJMP        START
;*******************************************************************************

;*******************************************************************************
;*******************************************************************************
MEM:
        MOV         DPTR,#300H          ;START ADDRESS ABOVE ALLOCATED MEM
        MOV         A,#55H
LOOP:
        MOVX        @DPTR,A              ;WRITE PATTERN TO RAM
        INC         DPTR
        MOV         R0,DPH
        CJNE        R0,#1FH,FILL_LOOP
        MOV         R0,DPL
        CJNE        R0,#0FFH,FILL_LOOP
```

```
        RET
;*******************************************************************************

;*******************************************************************************
;NSURE THAT CMOS ABOVE ALLOCATED MEMORY STILL CONTAINS 55H, AND IS NOT  *
;ORRUPTED                                                               *
;*******************************************************************************
;AM:
        MOV         DPTR,#300H          ;START ADDRESS ABOVE ALLOCATED

;OOP2:
        MOVX        A,@DPTR             ;READ PATTERN FROM RAM
        CJNE        A,#55H,RAM_ERR      ;PATTERN NOT THE SAME
        INC         DPTR
        MOV         R0,DPH
        CJNE        R0,#1FH,RAM_LOOP2
        MOV         R0,DPL
        CJNE        R0,#0FFH,RAM_LOOP2
        SJMP        RAM_EXIT
;ERR:
        MOV         R3,DPL
        MOV         R4,DPH
        MOV         A,R4
        INC         A
        LCALL       SEND_BYTE
        MOV         A,R3
        LCALL       SEND_BYTE
```

```
        LCALL       SEND
        MOV         A,R3
        MOV         DPTR,#LED_1
        MOVX        @DPTR,A

        MOV         DPTR,#LED_2
        MOV         A,R4
        MOVX        @DPTR,A

        LCALL       CLS1
        MOV         DPTR,#MSG4
        LCALL       CODE_TO_LCD
        MOV         DPTR,#LCD1_DAT_WR
        LCALL       LCD_WR

        MOV         COUNT,#10
        LCALL       ERR_BEEP            ;SIT IN PERMANENT LOOP
EXIT:

        RET
;*******************************************************************************
```

```
*****************************************************************
RANSMIT A BYTE                                                  *
                                                                *
N ENTRY:                                                        *
        ACC = BYTE TO TRANSMIT                                  *
*****************************************************************
BYTE:
        CLR         ES              ;DISABLE SERIAL INTS
        CLR         P3.5            ;ENABLE DS3695
        MOV         SBUF,A
        LCALL       WAIT
        SETB        P3.5            ;DISABLE DS3695

        RET
*****************************************************************

*****************************************************************
TRANSMIT A STRING OF DATA, DISABLE INTERRUPTS                   ;

ON ENTRY:
        DPTR POINTS TO SOURCE ADDRESS
*****************************************************************
':
        MOV         R0,#50
_LP1:
        MOVX        A,@DPTR
        LCALL       SEND_BYTE
```

```
        INC         DPTR
        DJNZ        R0,SEND_LP1

        RET
;*******************************************************************************

;*******************************************************************************
;AIT FOR A CERTAIN PERIOD OF TIME                                              *
;                                                                             *
;N ENTRY:                                                                     *
;       NOTHING                                                               *
;*******************************************************************************

        PUSH        ACC
        MOV         A,R0
        PUSH        ACC
        MOV         A,R1
        PUSH        ACC
        MOV         R0,#0FFH
        MOV         R1,#0FH
;1:
        DJNZ        R0,WAIT1
        MOV         R0,#0FFH
        DJNZ        R1,WAIT1
        POP         ACC
        MOV         R1,A
        POP         ACC
```

```
        MOV         R0,A
        POP         ACC

        RET
;*******************************************************************************

;*******************************************************************************
;RITE A PARTICULAR VALUE ONTO THE STACK WHICH IS CHECKED EACH TIME       *
;HROUGH THE MAIN LOOP, IF THIS VALUE DOES CHANGE THE STACK IS GROWING    *
;O HIGH AND A STACK OVERFLOW IS IMINENT                                  *
                                                                        *
;N ENTRY:                                                               *
    NOTHING                                                             *
;*******************************************************************************
_STACK_CHK:
        MOV         R0,#STACK_END-10        ;60 - 10 = STACK+50
        MOV         A,#0AAH
        MOV         @R0,A

        RET
;*******************************************************************************

;*******************************************************************************
CHECK IF THE STACK HAS GROWN BEYOND A CERTAIN POINT, IE STACK OVERFLOW
MIGHT OCCUR

ON ENTRY:
```

```
   NOTHING                                                              *
*****************************************************************************
ACK:
        MOV         R0,#STACK_END-10        ;60 - 10 = STACK+50
        MOV         A,@R0
        CJNE        A,#0AAH,STACK_ERR   ;STACK OVERFLOW
        SJMP        STACK_EXIT
ERR:
        LCALL       CLS1
        MOV         DPTR,#MSG3
        LCALL       CODE_TO_LCD
        MOV         DPTR,#LCD1_DAT_WR
        LCALL       LCD_WR
        MOV         COUNT,#06
        LCALL       ERR_BEEP            ;SIT IN PERMANENT LOOP
EXIT:

        RET
*****************************************************************************
```

```
;*****************************************************************
;RANSFER DATA FROM EXTERNAL DATA INTO LCD_BUF                     *
;                                                                 *
;N ENTRY:                                                         *
;       DPTR POINTS TO SOURCE BUFFER                              *
;       R2 = # OF BYTES TO TRANSFER FROM XDATA TO LCD_BUF         *
;       MAX BYTES TO TRANSFER == 32                               *
;*****************************************************************
;TO_LCD_BUF:
        MOV         R0,#LCD_BUF
;LP1:
        MOVX        A,@DPTR         ;READ SOURCE BUFFER
        MOV         P2,#0           ;IN CONJUNCTION WITH MOVX R0/R1
        MOVX        @R0,A           ;WRITE TO LCD_BUF
        INC         R0
        INC         DPTR
        DJNZ        R2,XDAT_LP1
        MOV         A,#EOS
        MOV         P2,#0           ;IN CONJUNCTION WITH MOVX R0/R1
        MOVX        @R0,A

        RET
;*****************************************************************

;:      DB          '*TESTING CARD   READER1*********',0
;32:    DB          2,'0 82530 IS FULLY  OPERATIONAL!!! ',3,0
;2:     DB          2,'0 THE  POWER  AT  POINT  OF  SALE',3,0
```

```
DB              ' STACK OVERFLOW @ STACK + 56 !!!',0

DB              'MEMORY ERROR!!!',0

DB              'PASSED MEM FILL',0

DB              'INTERRUPT ERROR',0


END
```

)1.ASM

```
 INPUT_OUTPUT_MODULE_1
    ALL I_O ROUTINES IN THIS MODULE

;*******************************************************************************
IC  INIT_HARDWARE, LCD_WR, CODE_TO_LCD, BEEP, TEST, KYB_READ
IC  IDATA_TO_TEMP, HEX_TO_ASCII, HEX_TO_DEC, CLS1, CLS2, ERR_BEEP

N   DATA (COUNT, TEMP_BUF, KEY_COUNT, RX_MODE, LED_COUNT1, LED_COUNT2)
N   DATA (ADDR1, ADDR2, BIN_ADDR, CRD_COUNT, QUE_COUNT)

N   BIT (KEY_FLAG, LOCK_FLAG, DRAW1_FLAG, TX_BUF_FULL, RX_BUF_FULL)
N   BIT (IS_DATA, RE_TX_FLAG, RX_DLE, TMOUT, CRD_FLAG, SCAN_FLAG)
    BIT (QUE_FLAG, QUE_FULL, CRD_ON_OFF, REC_FLAG, DRAW2_FLAG, ENQ_FLAG)
N   BIT (ENQ_FULL)

N   XDATA (LOCK_STAT, DRAW1_STAT, KEY_BUF, LCD_BUF, LED_BUF, REC_STAT)
N   XDATA (DRAW2_STAT)

N   CODE (WAIT, INIT_SCAN)
;*******************************************************************************

1_SEG   SEGMENT     CODE            ;RELOCATABLE CODE SEGMENT
        RSEG        I_O_1_SEG
```

```
;*************************************************************************
;NITIALIZE HARDWARE                                                      *
;NITIALIZE VARIABLES IN EXTERNAL DATA MEMORY AFTER THE RAM TEST          *
;S THE RAM TEST WILL OBVIOUSLY DESTROY THE VARIABLES                     *
;*************************************************************************
HARDWARE:
COMMS:
        SETB    P1.3            ;DRAWER SOLENOID, GOES THRU 7404
        SETB    P3.5            ;DS3695 RX MODE, GOES THRU 7404
        MOV     TMOD,#20H       ;T1 8 BIT AUTO RELOAD, MODE 2
        MOV     TL1,#0          ;RELOAD VALUE FOR T1
        MOV     TH1,#0FBH       ;BAUD RATE 4800, XTAL=9.216MHz
        ORL     PCON,#80H       ;DOUBLE BAUD RATE TO 9600
        SETB    TR1             ;START COUNTER 1

*********   19.2 KBAUD
        MOV     RCAP2H,#0FFH    ;T2 COUNT
        MOV     RCAP2L,#0F1H
        MOV     T2CON,#34H      ;T2 BAUD RATE GENERATOR
***************************

        MOV     SCON,#50H       ;SERIAL CONTRL, MODE 1, 8 BIT UART
                                ;1 START, 8 DATA, 1 STOP BIT
_TIMER0:
                                ;TIMER 0 = MODE 1
                                ;USED FOR TIMEOUT PERIODS
                                ;1 TIMER TICK = 1 M/C CYCLE =
```

```
                                    ;12 CLOCK CYCLES, = 1/12 CLK FREQ
                                    ;XTAL = 9.216MHz, TICK = 1.3uSEC
                                    ;XTAL = 1.8MHz, TICK = 6.7uSEC
        ORL         TMOD,#01        ;CONTRL BY TRx, 16 BIT, TIMER
INT:
        SETB        EA              ;ENABLE ALL INTERRRUPTS
        SETB        ET0             ;ENABLE TIMER 0 INT
        CLR         ES              ;DISABLE SERIAL PORT INT
        ORL         IP,#18H         ;INT PRIORITY, SERIAL & T0
        CLR         TR0             ;STOP TIMER 0
RAM:                                ;TEST LOWER 256 BYTES OF XDATA
                                    ;AS THE LCD_BUF RESIDES IN THIS AREA
                                    ;AS 0FFH IN LCD_BUFFER, NEVER FINDS
                                    ;EOS
                                    ;AND SITS IN LCD_WR PERMANENTLY
        MOV         DPTR,#0         ;START ADDR
        MOV         R1,#0FFH
        MOV         A,#33H          ;TEST PATTERN
_LOOP1:
        MOVX        @DPTR,A
        MOVX        A,@DPTR
        CJNE        A,#33H,TEST_ERR ;TERMINAL WILL SIT IN PERMANENT LOOP
        INC         DPTR
        DJNZ        R1,TEST_LOOP1
        SJMP        INIT_LEDS
T_ERR:
        MOV         COUNT,#03
```

```
        LCALL       ERR_BEEP
LEDS:
        MOV         A,#0
        MOV         DPTR,#LED_1
        MOVX        @DPTR,A
        MOV         A,#0
        MOV         DPTR,#LED_2
        MOVX        @DPTR,A
        CLR         TMOUT
LCD1:
        MOV         R0,#1FH      ;AT POWER UP, LCD IS INTENALLY BUSY
        MOV         DPTR,#LCD1_STAT ;WAIT UNTIL THEY ARE READY
1:
        CLR         TR0             ;STOP TIMER_0
        MOV         TH0,#0          ;USED FOR TIMEOUT LOOP
        MOV         TL0,#0
        SETB        TR0             ;START TIMER_0
:
        MOVX        A,@DPTR         ;READ BUSY FLAG, LCD_STAT
        JB          TMOUT,WAIT3     ;TIMEOUT  HAS OCCURRED
        JB          ACC.7,WAIT2     ;LCD IS BUSY IF D7 = 1
        CLR         TMOUT
        CLR         TR0             ;STOP TIMER_0
        SJMP        WAIT_LCD2       ;LCD OPERATING NORMALLY
3:
        CLR         TMOUT
        DJNZ        R0,WAIT1
```

```
        MOV         COUNT,#02       ;LCD ERROR HAS OCCURRED
        LCALL       ERR_BEEP        ;WAIT IN PERMANENT LOOP
LCD2:
        MOV         R0,#1FH     ;AT POWER UP, LCD IS INTENALLY BUSY
        MOV         DPTR,#LCD2_STAT ;WAIT UNTIL THEY ARE READY
1:
        CLR         TR0             ;STOP TIMER_0
        MOV         TH0,#0          ;USED FOR TIMEOUT LOOP
        MOV         TL0,#0
        SETB        TR0             ;START TIMER_0
5:
        MOVX        A,@DPTR         ;READ BUSY FLAG, LCD_STAT
        JB          TMOUT,WAIT6     ;TIMEOUT  HAS OCCURRED
        JB          ACC.7,WAIT5     ;LCD IS BUSY IF D7 = 1
        CLR         TMOUT
        CLR         TR0             ;STOP TIMER_0
        SJMP        INIT_LCD        ;LCD OPERATING NORMALLY
6:
        CLR         TMOUT
        DJNZ        R0,WAIT4
        MOV         COUNT,#02       ;LCD ERROR HAS OCCURRED
        LCALL       ERR_BEEP        ;WAIT IN PERMANENT LOOP
_LCD:
        MOV         R0,#LCD_BUF
        MOV         P2,#0               ;IN CONJUNCTION WITH MOVX R0/R1
        MOV         A,#FUNCTION         ;FUNCTION SET
        MOVX        @R0,A
```

```
INC         R0
MOV         A,#0                ;EOS
MOVX        @R0,A
MOV         DPTR,#LCD1_CMD_WR
LCALL       LCD_WR
MOV         DPTR,#LCD2_CMD_WR
LCALL       LCD_WR

MOV         R0,#LCD_BUF
MOV         P2,#0               ;IN CONJUNCTION WITH MOVX R0/R1
MOV         A,#MODE             ;MODE SET
MOVX        @R0,A
INC         R0
MOV         A,#0                ;EOS
MOVX        @R0,A
MOV         DPTR,#LCD1_CMD_WR
LCALL       LCD_WR
MOV         DPTR,#LCD2_CMD_WR
LCALL       LCD_WR

MOV         R0,#LCD_BUF
MOV         P2,#0               ;IN CONJUNCTION WITH MOVX R0/R1
MOV         A,#DSP_ON_OFF       ;LCD ON
MOVX        @R0,A
INC         R0
MOV         A,#0                ;EOS
MOVX        @R0,A
```

```
        MOV     DPTR,#LCD1_CMD_WR
        LCALL   LCD_WR
        MOV     DPTR,#LCD2_CMD_WR
        LCALL   LCD_WR

        LCALL   CLS1
        LCALL   CLS2

        MOV     DPTR,#MSG1
        LCALL   CODE_TO_LCD
        MOV     DPTR,#LCD2_DAT_WR
        LCALL   LCD_WR
'EST:                           ;32KBYTES X 8
        MOV     R0,#LCD_BUF
        MOV     P2,#0           ;IN CONJUNCTION WITH MOVX R0/R1
        MOV     A,#CLR_DSP      ;CLEAR DISPLAY & RESET CURSOR
        MOVX    @R0,A
        INC     R0
        MOV     A,#0            ;EOS
        MOVX    @R0,A
        MOV     DPTR,#LCD1_CMD_WR
        LCALL   LCD_WR
        MOV     DPTR,#MSG3
        LCALL   CODE_TO_LCD
        MOV     DPTR,#LCD1_DAT_WR
        LCALL   LCD_WR
```

```
        MOV         DPTR,#0             ;START ADDRESS
        MOV         R0,#0FFH
        MOV         R1,#7FH
        MOV         A,#33H              ;WRITE PATTERN TO RAM
)OP1:
        MOVX        @DPTR,A
        INC         DPTR
        DJNZ        R0,RAM_LOOP1
        MOV         R0,#0FFH
        DJNZ        R1,RAM_LOOP1        ;8000H LOCATIONS
        MOV         DPTR,#0             ;START ADDRESS
        MOV         R0,#0FFH
        MOV         R1,#7FH
OOP2:
        MOVX        A,@DPTR             ;READ PATTERN FROM RAM
        INC         DPTR
        CJNE        A,#33H,RAM_ERR      ;PATTERN NOT THE SAME
        DJNZ        R0,RAM_LOOP2
        MOV         R0,#0FFH
        DJNZ        R1,RAM_LOOP2        ;8000H LOCATIONS
        LCALL       CLS1
        MOV         DPTR,#MSG4
        LCALL       CODE_TO_LCD
        MOV         DPTR,#LCD1_DAT_WR
        LCALL       LCD_WR
        SJMP        INIT_VARIABLES
```

```
RR:
        MOV         R0,#LCD_BUF
        MOV         P2,#0               ;IN CONJUNCTION WITH MOVX R0/R1
        MOV         A,#CLR_DSP          ;CLEAR DISPLAY & RESET CURSOR
        MOVX        @R0,A
        INC         R0
        MOV         A,#0                ;EOS
        MOVX        @R0,A
        MOV         DPTR,#LCD1_CMD_WR
        LCALL       LCD_WR
        MOV         DPTR,#MSG2
        LCALL       CODE_TO_LCD
        MOV         DPTR,#LCD1_DAT_WR
        LCALL       LCD_WR
        MOV         COUNT,#04
        LCALL       ERR_BEEP
_VARIABLES:
        CLR         ENQ_FULL
        CLR         ENQ_FLAG
        CLR         REC_FLAG
        CLR         CRD_ON_OFF
        CLR         QUE_FULL
        CLR         QUE_FLAG
        CLR         CRD_FLAG
        CLR         TMOUT
        CLR         RX_DLE
        CLR         RE_TX_FLAG
```

```
        CLR     DRAW1_FLAG
        CLR     DRAW2_FLAG
        CLR     IS_DATA
        CLR     KEY_FLAG
        CLR     LOCK_FLAG
        CLR     TX_BUF_FULL
        CLR     RX_BUF_FULL
        MOV     QUE_COUNT,#0
        MOV     CRD_COUNT,#0
        MOV     KEY_COUNT,#0
        MOV     RX_MODE,#0
        MOV     ADDR1,#0
        MOV     ADDR2,#0
        MOV     LED_COUNT1,#0FFH
        MOV     LED_COUNT2,#09H
        MOV     R0,#LED_BUF         ;INIT LED_BUF
        MOV     R1,#04
        MOV     A,#0
OOP1:
        MOV     P2,#0               ;IN CONJUNCTION WITH MOVX R0/R1
        MOVX    @R0,A
        INC     R0
        DJNZ    R1,INIT_LOOP1
YB:
        MOV     DPTR,#KYB_CMD       ;ENCODED SCAN, 2 KEY LOCKOUT
        MOV     A,#24H              ;F= 333KHz. DIVIDE 1.8MHz CLK BY
                                    ;4
```

```
     MOV        A,#2EH              ;F= 1.6MHz. DIV 9.216MHz CLK BY
                                     ;15
     MOVX       @DPTR,A
:K_ADDR_STAT:                       ;LOCK AND DIP SW ON SAME 'LS245
     MOV        DPTR,#LOCK_JMP      ;READ STATUS
     MOVX       A,@DPTR
     MOV        DPTR,#LOCK_STAT     ;STORE STATUS
     MOVX       @DPTR,A
1_ADDR:
     ANL        A,#0FH              ;LS NIBBLE FOR DEVICE ADDR
     MOV        BIN_ADDR,A
     LCALL      CLS1
     MOV        A,BIN_ADDR
     LCALL      HEX_TO_DEC          ;2 BYTE ASCII EQUIV IN LCD_BUF+3
     MOV        DPTR,#LCD_BUF+3     ;COMMS PROTOCOL USES ASCII
     MOVX       A,@DPTR             ;READ ADDR2
     MOV        ADDR2,A
     DEC        DPL
     MOVX       A,@DPTR             ;READ ADDR1
     MOV        ADDR1,A
     MOV        DPTR,#LCD1_DAT_WR
     LCALL      LCD_WR
C_STAT:
     MOV        DPTR,#REC_STAT      ;CLOSED = 0, OPEN = 1
     MOV        A,#0                ;INIT ACC
     MOV        C,P1.7              ;READ REC STATUS
     RLC        A                   ;ACC CONTAINS 0/1
```

```
    MOVX        @DPTR,A             ;SAVE REC STATUS
W1_STAT:
    MOV         DPTR,#DRAW1_STAT    ;CLOSED = 0, OPEN = 1
    MOV         A,#0                ;INIT ACC
    MOV         C,P1.4              ;READ DRAW STATUS
    RLC         A                   ;ACC CONTAINS 0/1
    MOVX        @DPTR,A             ;SAVE DRAW STATUS
W2_STAT:
    MOV         DPTR,#DRAW2_STAT    ;CLOSED = 0, OPEN = 1
    MOV         A,#0                ;INIT ACC
    MOV         C,P1.6              ;READ DRAW STATUS
    RLC         A                   ;ACC CONTAINS 0/1
    MOVX        @DPTR,A             ;SAVE DRAW STATUS
T_SCAN:
    CLR         SCAN_FLAG
    LCALL       INIT_SCAN
M_NUM:
    LCALL       CLS1
    MOV         DPTR,#MSG5
    LCALL       CODE_TO_LCD

    MOV         DPTR,#LCD_BUF+22
    MOV         A,ADDR1
    MOVX        @DPTR,A
    INC         DPTR
    MOV         A,ADDR2
    MOVX        @DPTR,A
```

```
        MOV         DPTR,#LCD1_DAT_WR
        LCALL       LCD_WR
        LCALL       BEEP
EXIT:
        SETB        ES                  ;ENABLE SERIAL INTERRUPT

        RET
;*************************************************************************

;*************************************************************************
;READ 8279 FOR KEYBOARD DEPRESSIONS                                      *
;NB:!!!!!                                                                *
;FOR CASES WHERE A 1 BYTE REGISTER IS USED INSTEAD OF THE DPTR           *
;FOR EXTRENAL DATA TRANSFERS, THE ADDRESS AREA IS ONLY 256 BYTES         *
;P0 IS THE LOWER 8 BIT ADDRESS, P2 IS THE HIGHER 8 BIT ADDR              *
;FIRST WRITE 00 TO P2, BEFORE WR/RD FROM EXTERNAL DATA SPACE             *
;R0 IS USED TO ACCESS KEY_BUF IN EXTRENAL DATA                           *
;KEY_COUNT MUST BE ADDED TO KEY_BUF TO GET CORRECT OFFSET, KEY_COUNT     *
;MUST ONLY BE CLEARED ONCE THE KEY_BUF HAS BEEN EMPTIED                  *
;WHEN KEY_BUF IS FULL, SET A FLAG                                        *
;                                                                        *
;KEY_FLAG        INDICATES KEY VALUE IN KEY_BUF                          *
;*************************************************************************
READ:
        PUSH        DPL
        PUSH        DPH
```

```
;OOP1:
        JB          QUE_FULL,DUMMY_READ
        MOV         A,KEY_COUNT
        CJNE        A,#08,CHK_IRQ    ;CHK IF KEY_BUF IS FULL
        SJMP        DUMMY_READ
IRQ:
        JNB         INT0,KYB_EXIT    ;IF 8279 IRQ 'L', NO KEY DEPRESSED
        MOV         DPTR,#KYB_CMD    ;READ 8279 STATUS
        MOVX        A,@DPTR
        ANL         A,#0FH           ;NUMBER OF KEYS IN FIFO, BUFFER FULL
        JZ          KYB_EXIT         ;IF ACC = 0, THEN NO KEYS IN FIFO
        MOV         DPTR,#KYB_CMD    ;READ STATUS
        MOVX        A,@DPTR
        ANL         A,#30H           ;CHECK OVER/UNDERRUN FLAG
        JNZ         KYB_ERR          ;1 OF THE 2 BITS WAS SET
        MOV         A,#50H           ;WR TO 8279, ALLOWING US TO READ FIFO
        MOVX        @DPTR,A
        MOV         DPTR,#KYB_DAT    ;READ FIFO
        MOVX        A,@DPTR          ;READING FIFO

        MOV         DPTR,#KEY_TAB    ;CONVERT RAW KEY VALUE
        MOVC        A,@A+DPTR

        MOV         DPTR,#KEY_BUF
        MOV         B,A              ;STORE CONTENTS OF ACC
        MOV         A,DPL            ;LOWER ADDR OF KEY_BUF
        ADD         A,KEY_COUNT      ;EVERY KEY, INCREASE OFFSET TO
```

```
3UF
        MOV         DPL,A           ;OFFSET ADDR OF KEY_BUF
        MOV         A,B             ;RESTORE A
        MOVX        @DPTR,A         ;WRITE KEY_VAL TO KEY_BUF
        SETB        KEY_FLAG        ;KEY DATA AVAILABLE
        INC         KEY_COUNT       ;COUNT # OF KEYS IN BUFFER
        LCALL       BEEP
        SJMP        KYB_LOOP1       ;RD NEXT KEY DEPRESSION
ERR:
        MOV         KEY_COUNT,#0
        SJMP        KYB_EXIT
MY_READ:
        JNB         INT0,KYB_EXIT   ;IF 8279 IRQ 'L', NO KEY DEPRESSED
        MOV         DPTR,#KYB_CMD   ;READ STATUS
        MOV         A,#50H          ;WR TO 8279, ALLOWING US TO READ FIFO
        MOVX        @DPTR,A
        MOV         DPTR,#KYB_DAT   ;READ FIFO
        MOVX        A,@DPTR         ;READING FIFO
_EXIT:
        POP         DPH
        POP         DPL

        RET
;*******************************************************************************
```

```
;*************************************************************************
;CONVERT HEX VALUE INTO ITS DECIMAL ASCII EQUIVALENT.                     *
;                                                                         *
;ON ENTRY:                                                                *
;        ACC = BYTE TO BE CONVERTED                                       *
;*************************************************************************
TO_DEC:
        MOV         R1,A            ;SAVE ACC
        MOV         R0,#LCD_BUF+3
        MOV         A,#SPACE
        MOV         P2,#0           ;IN CONJUNCTION WITH MOVX R0/R1
        MOVX        @R0,A
        DEC         R0
        MOV         P2,#0           ;IN CONJUNCTION WITH MOVX R0/R1
        MOVX        @R0,A
        DEC         R0
        MOV         P2,#0           ;IN CONJUNCTION WITH MOVX R0/R1
        MOVX        @R0,A
        DEC         R0
        MOV         P2,#0           ;IN CONJUNCTION WITH MOVX R0/R1
        MOVX        @R0,A           ;CLEAR OLD DATA FROM LCD_BUF

        MOV         R0,#LCD_BUF+4
        MOV         A,#EOS
        MOV         P2,#0           ;IN CONJUNCTION WITH MOVX R0/R1
        MOVX        @R0,A
        DEC         R0
```

```
)P1:
        MOV         B,#10           ;DIVIDE ACC BY 10
        MOV         A,R1            ;RESTORE ACC
        DIV         AB              ;REMAINDER IN B
        ORL         B,#30H          ;ASCII NUMBER
        MOV         R1,A            ;SAVE ACC
        MOV         A,B
        MOV         P2,#0           ;IN CONJUNCTION WITH MOVX R0/R1
        MOVX        @R0,A           ;WRITE TO LCD_BUFF
        DEC         R0
        MOV         A,R1            ;RESTORE ACC
        SUBB        A,#09           ;CHK IF QUOTIENT < 10
        MOV         A,R1
        JNC         H_LOOP1
        ORL         A,#30H
        MOV         P2,#0           ;IN CONJUNCTION WITH MOVX R0/R1
        MOVX        @R0,A

        RET
*************************************************************************

*************************************************************************
THIS ROUTINE CONVERTS A HEX BYTE INTO ITS 2 BYTE ASCII EQUIV + EOS.     *
MOVE INTO ACC THE SOURCE  BYTE OF THE HEX VALUE TO BE CONVERTED         *
THE STRING WILL BE WRITTEN TO LCD_BUF, WITH EOS.                        *
                                                                        *
ON ENTRY:                                                               *
```

```
        ACC = BYTE TO BE CONVERTED                                      *
*****************************************************************************
TO_ASCII:
        MOV         R0,#LCD_BUF+1   ;DESTINATION BUFFER
        MOV         P2,#0           ;IN CONJUNCTION WITH MOVX R0/R1
        MOVX        @R0,A           ;WR TO LCD_BUF+1
        DEC         R0
        SWAP        A
        ANL         A,#0FH
        ADD         A,#30H
        MOV         R1,A
        CLR         C               ;CHECK FOR ALPHA CHAR
        MOV         A,#39H
        SUBB        A,R1
        MOV         A,R1            ;A = NUMERIC CHAR
        JNC         HEX_LOOP1
        ADD         A,#07           ;A = ALPHA CHAR
LOOP1:
        MOV         P2,#0           ;IN CONJUNCTION WITH MOVX R0/R1
        MOVX        @R0,A           ;WRITE LCD_BUF
        INC         R0              ;BACK AT LCD_BUF+1
        MOV         P2,#0           ;IN CONJUNCTION WITH MOVX R0/R1
        MOVX        A,@R0           ;READ LCD_BUF+1
        ANL         A,#0FH
        ADD         A,#30H
        MOV         R1,A            ;R1 USED FOR SUBB
        CLR         C               ;CHECK FOR ALPHA CHAR
```

```
        MOV         A,#39H
        SUBB        A,R1
        MOV         A,R1
        JNC         HEX_LOOP2
        ADD         A,#07           ;ALPHA CHAR
LOOP2:
        MOV         P2,#0           ;IN CONJUNCTION WITH MOVX R0/R1
        MOVX        @R0,A           ;WRITE LCD_BUF+1
        INC         R0
        MOV         P2,#0           ;IN CONJUNCTION WITH MOVX R0/R1
        MOV         A,#EOS
        MOVX        @R0,A

        RET
;*******************************************************************************

;*******************************************************************************
; WRITE TO LCD DISPLAY, DISPLAY CONTENTS OF LCD_BUF                            *
;                                                                              *
; ON ENTRY:                                                                    *
;       DPTR = LCD ADDRR IE LCD1 / LCD2                                        *
;*******************************************************************************
WR:
        JB          LCD_ERR,END_LCD_WR
        MOV         R0,#LCD_BUF
        MOV         COUNT,#1        ;INIT COUNTER
LOOP1:
```

```
        PUSH        DPL
LOOP6:
        CLR         TR0             ;STOP TIMER_0
        MOV         TH0,#0          ;USED FOR TIMEOUT LOOP
        MOV         TL0,#0
        MOV         DPL,#01         ;READ LCD ADDR X0001H
        SETB        TR0             ;START TIMER_0
LOOP2:
        MOVX        A,@DPTR         ;READ BUSY FLAG, LCD_STAT
        JB          TMOUT,LCD_ERR1  ;TIMEOUT  HAS OCCURRED
        JB          ACC.7,LCD_LOOP2 ;LCD IS BUSY IF D7 = 1
        CLR         TR0             ;STOP TIMER_0
        POP         DPL             ;RESTORE ORIGINAL ADDRESS
        MOV         P2,#0           ;IN CONJUNCTION WITH MOVX R0/R1
        MOVX        A,@R0           ;READ LCD BUFFER
        CJNE        A,#EOS,LCD_LOOP3
        SJMP        END_LCD_WR      ;EOS FOUND, THEREFORE EXIT
LOOP3:
        MOVX        @DPTR,A         ;WRITE TO DISPLAY
        MOV         A,#16           ;AFTER 16 CHARS, WRITE TO LINE #2
        CJNE        A,COUNT,LCD_LOOP4

        MOV         R7,#0FFH
        DJNZ        R7,$
        MOV         R7,#0FFH
        DJNZ        R7,$
```

```
        MOV         A,#LINE_2
        PUSH        DPL
        MOV         DPL,#0          ;WRITE LCD X000H CMD_WR
        MOVX        @DPTR,A
        POP         DPL             ;RESTORE ORIGINAL ADDRESS
        MOV         R7,#0FFH
        DJNZ        R7,$
        MOV         R7,#0FFH
        DJNZ        R7,$
        SJMP        LCD_LOOP5

LOOP4:
        MOV         A,#32           ;AFTER 32 CHARS, WRITE TO LINE #1
        CJNE        A,COUNT,LCD_LOOP5
        MOV         A,#LINE_1
        PUSH        DPL
        MOV         DPL,#0          ;WRITE LCD X000H CMD_WR
        MOVX        @DPTR,A
        POP         DPL             ;RESTORE ORIGINAL ADDRESS
LOOP5:
        INC         R0              ;NEXT CHARACTER TO DISPLAY
        INC         COUNT
        SJMP        LCD_LOOP1
ERR1:
        POP         DPL             ;MUST POP OFF STACK,ELSE THE
                                    ;WRONG VALUE IS ON THE STACK
                                    ;FOR THE NEXT INSTRUCTION
```

```
        MOV         COUNT,#02
        LCALL       ERR_BEEP
CD_WR:

        RET
;*************************************************************************

;*************************************************************************
;LEAR LCD #1                                                            *
;B: THIS ROUTINE OVERWRITES LCD_BUF                                     *
;*************************************************************************

        MOV         R0,#LCD_BUF
        MOV         P2,#0               ;IN CONJUNCTION WITH MOVX R0/R1
        MOVX        A,@R0
        PUSH        ACC                 ;SAVE FIRST 2 BYTES OF LCD_BUF
        INC         R0
        MOVX        A,@R0
        PUSH        ACC

        MOV         R0,#LCD_BUF
        MOV         A,#CLR_DSP          ;CLEAR DISPLAY & RESET CURSOR
        MOVX        @R0,A
        INC         R0
        MOV         A,#0                ;EOS
        MOVX        @R0,A
        MOV         DPTR,#LCD1_CMD_WR
```

```
        LCALL       LCD_WR

        MOV         R0,#LCD_BUF+1
        MOV         P2,#0               ;IN CONJUNCTION WITH MOVX R0/R1
        POP         ACC
        MOVX        @R0,A
        DEC         R0
        POP         ACC
        MOVX        @R0,A

        RET
;*******************************************************************************

;*******************************************************************************
;CLEAR LCD #2                                                                  *
;NB: THIS ROUTINE OVERWRITES LCD_BUF                                           *
;*******************************************************************************
;
        MOV         R0,#LCD_BUF
        MOV         P2,#0               ;IN CONJUNCTION WITH MOVX R0/R1
        MOVX        A,@R0
        PUSH        ACC                 ;SAVE FIRST 2 BYTES OF LCD_BUF
        INC         R0
        MOVX        A,@R0
        PUSH        ACC

        MOV         R0,#LCD_BUF
```

```
        MOV         P2,#0               ;IN CONJUNCTION WITH MOVX R0/R1
        MOV         A,#CLR_DSP          ;CLEAR DISPLAY & RESET CURSOR
        MOVX        @R0,A
        INC         R0
        MOV         A,#0                ;EOS
        MOVX        @R0,A
        MOV         DPTR,#LCD2_CMD_WR
        LCALL       LCD_WR

        MOV         R0,#LCD_BUF+1
        MOV         P2,#0               ;IN CONJUNCTION WITH MOVX R0/R1
        POP         ACC
        MOVX        @R0,A
        DEC         R0
        POP         ACC
        MOVX        @R0,A

        RET
*******************************************************************************
```

```
;*******************************************************************************
;TRANSFER DATA FROM CODE AREA TO EXTERNAL DATA INTO LCD_BUF                     *
;FIRST SETUP SOURCE REGISTERS = DPTR                                            *
;                                                                              *
;ON ENTRY:                                                                     *
;       DPTR POINTS TO SOURCE BUFFER                                           *
;*******************************************************************************
_TO_LCD:
        MOV         R0,#LCD_BUF     ;DESTINATION REGISTER
_LOOP1:
        MOV         A,#0
        MOVC        A,@A+DPTR       ;READ CODE
        MOV         P2,#0           ;IN CONJUNCTION WITH MOVX R0/R1
        MOVX        @R0,A           ;WRITE TO INTERNAL DATA SPACE
        INC         R0
        INC         DPTR
        CJNE        A,#EOS,CODE_LOOP1
        MOV         A,#0            ;ENSURE THAT EOS IS ADDED INTO BUFFER
        MOV         P2,#0           ;IN CONJUNCTION WITH MOVX R0/R1
        MOVX        @R0,A

        RET
;*******************************************************************************
```

```
;*******************************************************************
;RANSFER DATA FROM INTENAL DATA AREA TO INTERNAL DATA INTO TEMP_BUF     *
                                                                       *
)N ENTRY:                                                              *
        R1 POINTS TO SOURCE ADDRR                                      *
;*******************************************************************
\_TO_TEMP:
        MOV         R0,#TEMP_BUF    ;DESTINATION REGISTER
\_LOOP1:
        MOV         A,@R1           ;READ SOURCE BUFFER
        MOV         @R0,A           ;WRITE TO INTERNAL DATA SPACE
        INC         R0
        INC         R1
        CJNE        A,#EOS,IDATA_LOOP1

        RET
;*******************************************************************

;*******************************************************************
3EEP THE BUZZER                                                        *
;*******************************************************************
:
        PUSH        DPL
        PUSH        DPH
        MOV         DPTR,#BUZZER
        MOVX        @DPTR,A
        POP         DPH
```

```
        POP         DPL
        RET
;*******************************************************************************

;*******************************************************************************
; WRITE A CERTAIN VALUE TO LEDS                                                *
;                                                                              *
; ON ENTRY:                                                                    *
;       ACC = VALUE TO WRITE TO LEDS                                           *
;*******************************************************************************
:
        PUSH        DPL
        PUSH        DPH
        MOV         B,P2
        PUSH        B
        MOV         DPTR,#LED_2
        MOVX        @DPTR,A

        JNB         P1.6,$          ;WAIT FOR 'H'
        JB          P1.6,$          ;WAIT FOR 'L'

        POP         B
        MOV         P2,B
        POP         DPH
        POP         DPL
```

```
        RET
;*******************************************************************************


;*******************************************************************************
;BEEP THE BUZZER TO INDICATE A PARTICAULAR ERROR. COUNT CONTAINS                 *
;# OF TIMES TO BEEP.
;2 BEEPS = LCD ERROR                                                            *
;3 BEEPS = RAM ERROR, LOWER 256 BYTES                                           *
;4 BEEPS = RAM ERROR, SOMEWHERE IN THE 32K BLOCK                                *
;5 BEEPS = ROM ERROR                                                            *
;6 BEEPS = STACK OVERFLOW                                                       *
;                                                                               *
;ON ENTRY:                                                                      *
;        COUNT = # OF BEEPS                                                     *
;*******************************************************************************
;BEEP:
;LOOP1:
        LCALL       BEEP
        MOV         R0,#0FFH
        MOV         R1,#0FFH
;LOOP2:
        DJNZ        R0,ERR_LOOP2
        MOV         R0,#0FFH
        DJNZ        R1,ERR_LOOP2
        DJNZ        COUNT,ERR_LOOP1
        JMP         $                   ;*******SIT IN PERMANENT LOOP
```

```
        RET
;*******************************************************************************


;*******************************************************************************
; TEST LCD1                                                                    *
;*******************************************************************************
ST_LCD1:
        MOV         R0,#LCD_BUF
        MOV         R1,#20H
        MOV         A,#41H
        MOV         P2,#0           ;IN CONJUNCTION WITH MOVX R0/R1
LOOP1:
        MOVX        @R0,A
        INC         R0
        INC         A
        DJNZ        R1,T_LOOP1
        LCALL       CLS1
        MOV         DPTR,#LCD1_DAT_WR
        LCALL       LCD_WR

        RET
;*******************************************************************************
```

```
;*****************************************************************
; TEC 72 KEY KEYBOARD, FOR H.KEOWN'S (V1.0) INTELLIGENT TERMINAL  *
;*****************************************************************
;_TAB:
        DB      00,04,13,22,31,54,45,27,00,03   ;00 - 09
        DB      12,21,30,53,44,26,00,19,46,65   ;10 - 19
        DB      56,38,47,39,00,00,00,00,00,00   ;20 - 29
        DB      00,00,00,00,00,00,00,00,00,00   ;30 - 39
        DB      00,02,11,20,29,50,41,23,00,10   ;40 - 49
        DB      37,55,66,51,42,24,00,01,28,64   ;50 - 59
        DB      67,52,43,25,36,18,09,63,72,00   ;60 - 69
        DB      00,00,35,17,08,62,71,00,00,00   ;70 - 79
        DB      40,48,49,58,57,00,00,00,00,00   ;80 - 89
        DB      00,00,00,00,00,00,00,00,00,00   ;90 - 99
        DB      00,00,00,00,32,14,05,59,68,00   ;100 - 109
        DB      00,00,33,15,06,60,69,00,00,00   ;110 - 119
        DB      34,16,07,61,70,00,00,00         ;120 - 129

;*****************************************************************

;*****************************************************************
; TEC 84 KEY KEYBOARD, ITALIAN KEYBOARD                          *
;*****************************************************************
KEY_TAB:
        DB      00,00,00,00,00,00,00,00,84,83   ;00 - 09
        DB      82,81,79,80,78,76,36,35,34,33   ;10 - 19
        DB      31,32,30,28,24,23,22,21,19,20   ;20 - 29
```

```
        DB      18,16,12,11,10,09,07,08,06,04   ;30 - 39
        DB      48,47,46,45,43,44,42,40,60,59   ;40 - 49
        DB      58,57,55,56,54,52,72,71,70,69   ;50 - 59
        DB      67,68,66,64,00,00,00,00,00,00   ;60 - 69
        DB      00,00,77,75,74,73,00,00,00,00   ;70 - 79
        DB      29,27,26,25,00,00,00,00,17,15   ;80 - 89
        DB      14,13,00,00,00,00,05,03,02,01   ;90 - 99
        DB      00,00,00,00,41,39,38,37,00,00   ;100 - 109
        DB      00,00,53,51,50,49,00,00,00,00   ;110 - 119
        DB      65,63,62,61,00,00,00,00         ;120 - 129

;*************************************************************************

;*************************************************************************
KEY_TEMPLATE:
        DB      00,00,00,00,00,00,00,00,00,00   ;00 - 09
        DB      00,00,00,00,00,00,00,00,00,00   ;10 - 19
        DB      00,00,00,00,00,00,00,00,00,00   ;20 - 29
        DB      00,00,00,00,00,00,00,00,00,00   ;30 - 39
        DB      00,00,00,00,00,00,00,00,00,00   ;40 - 49
        DB      00,00,00,00,00,00,00,00,00,00   ;50 - 59
        DB      00,00,00,00,00,00,00,00,00,00   ;60 - 69
        DB      00,00,00,00,00,00,00,00,00,00   ;70 - 79
        DB      00,00,00,00,00,00,00,00,00,00   ;80 - 89
        DB      00,00,00,00,00,00,00,00,00,00   ;90 - 99
        DB      00,00,00,00,00,00,00,00,00,00   ;100 - 109
        DB      00,00,00,00,00,00,00,00,00,00   ;110 - 119
```

```
        DB          00,00,00,00,00,00,00,00          ;120 - 129

;*************************************************************************


;*************************************************************************
MSG1:   DB          '*ADS* THE POWER AT POINT OF SALE',0
SG1:    DB          ' THE  POWER  AT  POINT  OF  SALE',0
SG2:    DB          '******RAM ERROR AT POWER-UP*****',0
SG3:    DB          ' PERFORMING RAM      TEST       ',0
SG4:    DB          'PASSED RAM TEST!                ',0
SG5:    DB          'PASSED RAM TEST!TERM #      V1.2',0

INCLUDE (EQUATES.ASM)

        END
```

I02.ASM

```
AME     INPUT_OUTPUT_MODULE_2
        ALL I_O ROUTINES IN THIS MODULE

;*************************************************************************
UBLIC   LED_ON_OFF, LED_FLASH, LOCK_READ, DRAW1_READ, DRAW1_OPEN, CRD_READ
UBLIC   DRAW2_READ, DRAW2_OPEN, WAIT, REC_READ

XTRN    DATA (COUNT, TEMP_BUF, LED_COUNT1, LED_COUNT2, CRD_COUNT)

XTRN    BIT (LED_FLAG, LOCK_FLAG, DRAW1_FLAG, TMOUT, CRD_FLAG, CRD_ON_OFF)
XTRN    BIT (REC_FLAG, DRAW2_FLAG)

XTRN    XDATA (LOCK_STAT, DRAW1_STAT, KEY_BUF, RX_BUF, TX_BUF, LED_BUF,
RD_BUF)
 RN     XDATA (REC_STAT, DRAW2_STAT, LCD_BUF)

XTRN    CODE (INIT_HARDWARE, LCD_WR, CODE_TO_LCD, BEEP, TEST, KYB_READ)
 TRN    CODE (IDATA_TO_TEMP, HEX_TO_ASCII, HEX_TO_DEC, CLS1)
;*************************************************************************
I_O_2_SEG   SEGMENT     CODE            ;RELOCATABLE CODE SEGMENT

            RSEG        I_O_2_SEG
```

```
;*****************************************************************
;*  TRANSFER CARD DATA VIA A PORT PIN. WHEN DATA IS AVAILABLE, DATA      *
;*  LINE IS TAKEN 'L' TO INFORM HOST THAT DATA IA AVAILABLE. DATA IS THEN *
;*  CLOCKED OUT ON THE FALLING EDGE OF THE CLOCK. TRANSFER 1 BYTE FOR THE *
;*  STATUS AND 37 BYTES FOR CARD DATA                                    *
;*  38 BYTES X 8 BITS = 304 CLOCK PULSES. LSB IS TRANSFERED FIRST.       *
;*  IF THERE'S AN ERROR IN READING THE CARD, THEN ONLY THE STATUS BYTE    *
;*  IS TRANSFERED TO THE HOST.                                           *
;*  P3.3 = SERIAL CARD DATA FROM CARD READER TO HOST.                    *
;*  P3.4 = SERIAL DATA CLOCK FROM HOST TO CARD READER.                   *
;*                                                                       *
;*  ON ENTRY:                                                            *
;*          NOTHING                                                      *
;*                                                                       *
;*  ON EXIT:                                                             *
;*          CRD_BUF CONTAINS DATA FROM CARD READER                       *
;*****************************************************************
CRD_READ:
          JB          P3.3,CRD_EXIT   ;LINE 'H', NO DATA AVAIALBLE
          MOV         CRD_COUNT,#01   ;DEFAULT TO 1 BYTE IN CRD_BUF
          MOV         R0,#CRD_BUF
          LCALL       SER_DATA        ;ACC = BYTE READ FROM PORT
          MOV         P2,#0           ;IN CONJUNCTION WITH MOVX R0/R1
          MOVX        @R0,A           ;SAVE STATUS BYTE
          INC         R0
          CJNE        A,#01,CRD_LP2   ;CHK STATUS BYTE, 01=OK 0FFH=BAD
          MOV         CRD_COUNT,#38   ;USED IN CHK_TX_BUFFS
```

```
        MOV         R1,#37          ;37 DATA BYTES TO TRANSFER
RD_LP1:
        LCALL       SER_DATA        ;ACC = BYTE READ FROM PORT .
        MOV         P2,#0           ;IN CONJUNCTION WITH MOVX R0/R1
        MOVX        @R0,A           ;SAVE CARD DATA
        INC         R0
        DJNZ        R1,CRD_LP1      ;READ 40 BYTES OF CARD DATA
RD_LP2:
        JNB         CRD_ON_OFF, CRD_EXIT    ;0 = OFF, 1 = ON
        SETB        CRD_FLAG        ;STILL NEED TO TX STATUS BYTE
RD_EXIT:

        RET
*************************************************************************

*************************************************************************
*  READ 1 BYTE FROM P3.3, CLOCK ON P3.4                                 *
*                                                                       *
*  1 M/C CYCLE = 1.3 uSEC. ALLOW FOR 20 M/C CYCLES FOR THE BIT          *
*  TRANSMISSION LOOP. ALLOW 100 uSEC FROM CLK 'H' TO CLK 'L' AND        *
*  ALSO 100 uSEC FROM THE TIME THE BIT IS READ TILL THE NEXT CLOCK      *
*  TRANSISSION                                                          *
*                                                                       *
*  ON ENTRY:                                                            *
*          NOTHING                                                      *
*                                                                       *
*  ON EXIT:                                                             *
```

```
*           ACC = BYTE READ                                                    *
*******************************************************************************
ER_DATA:
            MOV         R2,#08          ;8 BITS
            CLR         A
ER_LP1:
            SETB        P3.4            ;CLK 'H'
            LCALL       WAIT
            CLR         P3.4            ;CLK 'L', READ DATA ON FALLING EDGE
            LCALL       WAIT
            SETB        P3.3            ;SET BIT AS AN INPUT
            MOV         C,P3.3
            RRC         A
            DJNZ        R2,SER_LP1
            SETB        P3.4            ;SET CLK 'H'

            RET
*******************************************************************************
```

```
;*****************************************************************************
IT:
          PUSH        ACC
          MOV         ACC,#30
IT_LP1:
          DJNZ        ACC,WAIT_LP1
          POP         ACC

          RET
;*****************************************************************************

;*****************************************************************************
;*****************************************************************************

;
          MOV         R1,#8
LP1:
          MOVX        A,@DPTR
          LCALL       TEST
          INC         DPTR
          DJNZ        R1,T_LP1

          RET
;*****************************************************************************
```

```
;*************************************************************************
;* THE ASCII STRING THAT WE RECEIVE WILL HAVE 8 BYTES FOR LED ON/OFF AND  *
;* LED FLASH. THESE 8 BYTES ARE COMPRESSED INTO 4 BYTES, AND PLACED IN    *
;* LED_BUF.                                                               *
;* THE LED BUFFER HAS 8 BYTES, THE FIRST 4 INDICATES WHETHER THE LED IS   *
;* ON/OFF, THE NEXT 4 BYTES IS WHETHER THE LED SHOULD FLASH OR NOT. FIRST *
;* LOGICALLY AND OFF THE MSNIBBLE OF EACH BYTE, THEN COMPRESS THE 2 BYTES *
;* INTO 1 BYTE. WE ARE THEREFORE LEFT WITH 2 BYTES FOR LED ON/OFF AND 2   *
;* BYTES FOR LED FLASH.                                                   *
;* R0 WILL BE USED TO READ FROM RX_BUF (8 BYTES)                          *
;* R1 WILL BE USED TO WRITE TO LED_BUF (4 BYTES)                          *
;*                                                                        *
;* TRANSFER 8 BYTES FROM RX_BUF, COMPRESS THIS INTO 4 BYTES, AND PLACE    *
;* THE DATA IN LED_BUF. SWITCH ON THE RELEVANT LEDS.                      *
;* MESSAGE FORMAT:                                                        *
;*          RX_BUF CONTAINS 8 BYTES, FIRST 4 INDICATES IF THE LEDS WILL   *
;*          BE ON/OFF AND THE NEXT 4 IF THE LEDS WILL FLASH. BOTH         *
;*          LED ON/OFF AND FLASH BITS HAVE TO BE '1' FOR THE ASSOSIATED   *
;*          LED TO FLASH. LED ON/OFF = '1' MEANS LED IS PERMANENTLY ON    *
;*                                                                        *
;*          THE LS NIBBLE OF EACH BYTE REPRESENTS 4 LEDS, WE HAVE TO      *
;*          ADD 30H TO THE BYTE SO THAT THE RX ROUTINE DOESN'T CONFFUSE   *
;*          THIS WITH A CONTROL CHAR. AFTER RECIEVING THE MESSAGE 30H     *
;*          IS STRIPPED FROM THE BYTE, AND 2 BYTES CAN BE COMPRESSED      *
;*          INTO 1 BYTE, THEREBY REPRESENTING 8 LEDS.                     *
;*          LED_BUF THEREFORE ONLY REQUIRES 4 BYTES, 2 = LED ON/OFF       *
;*          AND 2 FOR LED FLASH                                           *
```

```
;*************************************************************************
ED_ON_OFF:
        CLR         LED_FLAG
        MOV         DPTR,#RX_BUF+2  ;READ RX DATA, EXCLUDE I/O, TYPE #
        MOV         R1,#LED_BUF     ;BUFFER ADDR
        MOV         COUNT,#8        ;8 BYTE LED BUFFER COUNTER
ED_LOOP:
        MOVX        A,@DPTR         ;READ RX_BUF
        ANL         A,#0FH          ;MASK OFF MS NIBBLE
        MOV         P2,#0           ;IN CONJUNCTION WITH MOVX R0/R1
        MOVX        @R1,A           ;LOWER 4 BITS INTO LED_BUF
        DEC         COUNT           ;BYTE COUNTER
        INC         DPTR            ;NEXT BYTE IN COMBUF
        MOVX        A,@DPTR         ;READ RX_BUF
        ANL         A,#0FH          ;MASK OFF MS NIBBLE
        SWAP        A               ;SHIFT LEFT BY 4 BITS
        MOV         B,A             ;SAVE ACC
        MOV         P2,#0           ;IN CONJUNCTION WITH MOVX R0/R1
        MOVX        A,@R1           ;READ LED_BUF
        ORL         A,B             ;OR NEXT NIBBLE
        MOV         P2,#0           ;IN CONJUNCTION WITH MOVX R0/R1
        MOVX        @R1,A           ;RESULT INTO LED_BUF
        INC         DPTR
        INC         R1              ;WRITE TO NEXT ADDR
        DJNZ        COUNT,LED_LOOP  ;NEXT BYTE
                                    ;TRANSFER FROM 8 BYTE RX_BUF
                                    ;INTO 4 BYTE LED_BUF COMPLETE
```

```
MOV         P2,#0           ;IN CONJUNCTION WITH MOVX R0/R1
MOV         R0,#LED_BUF
MOVX        A,@R0           ;THERE ARE CASES WHEN LED ON/OFF
                            ;BIT=0
MOV         B,A             ;AND FLASH=1,FIRST AND RELEVANT BYTES
MOV         R0,#LED_BUF+2   ;AND BYTES 1 & 3. THEREFORE ENSURE
MOVX        A,@R0           ;THAT IF LED ON/OFF = 0, LED IS OFF
ANL         A,B             ;IRRESPECTIVE OF LED FLASH BIT
MOVX        @R0,A           ;STORE RESULT AT BYTE 3 LOCATION

MOV         R0,#LED_BUF+1
MOVX        A,@R0           ;READ LED_BUF, BYTE 2
MOV         B,A
MOV         R0,#LED_BUF+3
MOVX        A,@R0           ;READ LED_BUF, BYTE 4
ANL         A,B
MOVX        @R0,A           ;STORE RESULT AT BYTE 4 LOCATION

                            ;IMMEDIATELY AFTER RECEPTION, WRITE
                            ;TO LEDS
MOV         R0,#LED_BUF
MOVX        A,@R0           ;READ LED_BUF
MOV         DPTR,#LED_1
MOVX        @DPTR,A         ;OUTPUT LED STATUS TO PORT
INC         R0
MOVX        A,@R0           ;READ LED_BUF+1
MOV         DPTR,#LED_2
```

```
        MOVX        @DPTR,A          ;OUTPUT LED STATUS TO PORT
        MOV         LED_COUNT1,#0    ;THIS WILL CAUSE MAIN LOOP TO
                                     ;CALL LED_FLASH
        MOV         LED_COUNT2,#1

        RET
;*************************************************************************

;*************************************************************************
; MAIN PROGRAM LOOP WILL CHECK LED_LOOP BYTE, AND WILL UPDATE THE LEDS    *
; AFTER A CERTAIN PERIOD OF TIME. THIS WILL CREATE THE FLASHING EFFECT    *
;*************************************************************************
D_FLASH:
        MOV         P2,#0            ;IN CONJUNCTION WITH MOVX R0/R1
        MOV         R1,#LED_BUF
        MOVX        A,@R1            ;BYTE 1, LED ON/OFF
        MOV         B,A
        INC         R1
        INC         R1               ;BYTE 3, LED FLASH
        MOVX        A,@R1            ;READ BYTE 3 FLASH
        XRL         A,B              ;XOR BYTES 1 AND 3
        MOV         R1,#LED_BUF
        MOVX        @R1,A            ;SAVE RESULT OF XOR
        MOV         DPTR,#LED_1
        MOVX        @DPTR,A
        INC         R1               ;BYTE 2, LED ON/OFF
        MOV         P2,#0            ;IN CONJUNCTION WITH MOVX R0/R1
```

```
        MOVX    A,@R1
        MOV     B,A
        INC     R1
        INC     R1              ;BYTE 4, LED FLASH
        MOVX    A,@R1
        XRL     A,B
        MOV     R1,#LED_BUF+1   ;BYTE 2, LED ON/OFF
        MOVX    @R1,A           ;RESULT OF XOR
        MOV     DPTR,#LED_2
        MOVX    @DPTR,A
        MOV     LED_COUNT1,#0FFH;UPDATE LEDS COUNTER
        MOV     LED_COUNT2,#0FH     ;1.8MHz XTAL
        MOV     LED_COUNT2,#09H     ;9.216MHz XTAL

        RET
;*******************************************************************************
```

```
;*****************************************************************************
; READ LOCK STATUS, IF IT CHANGES, SET LOCK_FLAG. STORE STATUS OF LOCK IN     *
; LOCK_STAT                                                                   *
;*****************************************************************************
CK_READ:
        MOV         DPTR,#LOCK_JMP  ;READ CURRENT LOCK AND JUMPER STATUS
        MOVX        A,@DPTR
        MOV         B,A             ;SAVE ACC
        MOV         DPTR,#LOCK_STAT ;READ PREVIOUS STATUS
        MOVX        A,@DPTR
        CJNE        A,B,LOCK1       ;STATUS HAS CHANGED
        SJMP        LOCK_EXIT
CK1:
        MOV         A,B
        MOV         DPTR,#LOCK_STAT
        MOVX        @DPTR,A         ;SAVE NEW STATUS
        SETB        LOCK_FLAG       ;LOCK STATUS HAS CHANGED
        LCALL       BEEP
CK_EXIT:

        RET
;*****************************************************************************
```

```
;*************************************************************************
; READ RECEIPT STATUS. CLOSED = 0, OPEN = 1. REC_ FLAG ONLY INDICATES THAT*
; THE STATUS OF THE RECEIPT SWITCH HAS CHANGED. THE ACTUAL STATUS IS      *
; STORED IN REC_STAT.                                                     *
;*************************************************************************
C_READ:
        MOV         A,#0                ;INIT ACC
        MOV         C,P1.7              ;READ CURRENT DRAW STATUS
        RLC         A
        MOV         B,A
        MOV         DPTR,#REC_STAT      ;READ PREVIOUS STATUS
        MOVX        A,@DPTR
        CJNE        A,B,NEW_REC_STAT    ;REC HAS CHANGED
        SJMP        REC_EXIT
C_REC_STAT:
        MOV         A,B                 ;SAVE CURRENT REC STATUS
        MOVX        @DPTR,A
        SETB        REC_FLAG            ;INDICATES STATUS HAS CHANGED
C_EXIT:
        RET
;*************************************************************************


;*************************************************************************
; READ DRAW #1 STATUS. CLOSED = 0, OPEN = 1. USE DRAW1_FLAG               *
; STATUS DICTATED BY EXISTING DRAW MECHANISM. DRAW1_FLAG ONLY INDICATES   *
; THAT THE STATUS OF THE DRAW HAS CHANGED. THE ACTUAL STATUS IS STORED    *
; IN DRAW1_STAT.                                                          *
```

```
*************************************************************************
AW1_READ:
        MOV         A,#0                ;INIT ACC
        MOV         C,P1.4              ;READ CURRENT DRAW STATUS
        RLC         A
        MOV         B,A
        MOV         DPTR,#DRAW1_STAT    ;READ PREVIOUS STATUS
        MOVX        A,@DPTR
        CJNE        A,B,NEW1_STAT       ;DRAW HAS CHANGED
        SJMP        DRAW1_EXIT
W1_STAT:
        MOV         A,B                 ;SAVE CURRENT DRAW STATUS
        MOVX        @DPTR,A
        SETB        DRAW1_FLAG          ;INDICATES STATUS HAS CHANGED
AW1_EXIT:

        RET
*************************************************************************


*************************************************************************
 READ DRAW #2 STATUS. CLOSED = 0, OPEN = 1. USE DRAW2_FLAG              *
 STATUS DICTATED BY EXISTING DRAW MECHANISM. DRAW2_FLAG ONLY INDICATES   *
 THAT THE STATUS OF THE DRAW HAS CHANGED. THE ACTUAL STATUS IS STORED    *
 IN DRAW2_STAT.                                                         *
*************************************************************************
```

```
AW2_READ:
        MOV         A,#0                ;INIT ACC
        MOV         C,P1.6              ;READ CURRENT DRAW STATUS
        RLC         A
        MOV         B,A
        MOV         DPTR,#DRAW2_STAT    ;READ PREVIOUS STATUS
        MOVX        A,@DPTR
        CJNE        A,B,NEW2_STAT       ;DRAW HAS CHANGED
        SJMP        DRAW2_EXIT
W2_STAT:
        MOV         A,B                 ;SAVE CURRENT DRAW STATUS
        MOVX        @DPTR,A
        SETB        DRAW2_FLAG          ;INDICATES STATUS HAS CHANGED
W2_EXIT:

        RET
*************************************************************************

*************************************************************************
 OPEN CASH DRAW #1                                                      *
*************************************************************************
W1_OPEN:
        CLR         TR0                 ;STOP TIMER_0
        MOV         TH0,#0FAH           ;XTAL =1.8MHz  FOR TIMEOUT LOOP
        MOV         TH0,#0C3H           ;XTAL =9.216MHz  FOR TIMEOUT LOOP
        MOV         TL0,#0
        SETB        TR0                 ;START TIMER_0
```

```
        CLR         P1.3                ;SOLENOID ON
        JNB         TF0,$               ;WAIT FOR TIMEOUT
        SETB        P1.3                ;SOLENOID OFF
        CLR         TMOUT               ;REFRER TIMER 0 INT ROUTINE

        RET
;*************************************************************************

;*************************************************************************
; OPEN CASH DRAW #2                                                      *
;*************************************************************************
AW2_OPEN:
        CLR         TR0                 ;STOP TIMER_0
        MOV         TH0,#0C3H           ;XTAL =9.216MHz  FOR TIMEOUT LOOP
        MOV         TL0,#0
        SETB        TR0                 ;START TIMER_0
        CLR         P1.5                ;SOLENOID ON
        JNB         TF0,$               ;WAIT FOR TIMEOUT
        SETB        P1.5                ;SOLENOID OFF
        CLR         TMOUT               ;REFRER TIMER 0 INT ROUTINE

        RET
;*************************************************************************

;*************************************************************************
;LUDE (EQUATES.ASM)
        END
```

COMMS.ASM

```
ME COMMS_MODULE
     LOW LEVEL AS WELL AS UPPER LEVEL COMMS ROUTINES

*************************************************************************
BLIC  TEST_COMMS, CHK_TX_BUFFS, CHK_RX_BUFFS, TX_STRING

RN    DATA (COUNT, TEMP_BUF, KEY_COUNT, TX_COUNT, RX_COUNT, LCD_NUM)
RN    DATA (LCD_COUNT, TX_BCC, RX_BCC, ADDR1, ADDR2, CRD_COUNT, QUE_COUNT)
RN    DATA (BIN_ADDR)

RN    BIT (KEY_FLAG, LOCK_FLAG, DRAW1_FLAG, TX_BUF_FULL, RX_BUF_FULL)
      BIT (LCD_FLAG, BCC_FLAG, IS_DATA, CRD_FLAG, QUE_FLAG, QUE_FULL)
RN    BIT (CRD_ON_OFF, REC_FLAG, DRAW2_FLAG)

RN    XDATA (LOCK_STAT, DRAW1_STAT, KEY_BUF, LCD_BUF, TX_BUF, RX_BUF,
_BUF)

RN    XDATA (TX_DATA, CRD_BUF, KEY_QUE, REC_STAT, DRAW2_STAT)

RN    CODE (INIT_HARDWARE, LCD_WR, CODE_TO_LCD, BEEP, TEST, KYB_READ)
RN    CODE (IDATA_TO_TEMP, HEX_TO_ASCII, HEX_TO_DEC, LED_ON_OFF, LED_FLASH)
RN    CODE (LOCK_READ, DRAW1_OPEN, DRAW2_OPEN, CLS1, CLS2)


*****     HOST ENQUIRY OF TERMINAL STATUS, LOCK, DRAW, RECEIPT
```

```
TRN   BIT (ENQ_FLAG, ENQ_FULL)
RN    XDATA (ENQ_BUF)
BLIC  ENQUIRE

*****    82530 COMMS CHIP
TRN   DATA (TX82_STAT, COUNT82, RX1_COUNT, RX2_COUNT)
TRN   BIT (TXBUF1_FULL, TXBUF2_FULL, RXBUF1_FULL, RXBUF2_FULL)
RN    XDATA (COM1_TXBUF, COM1_RXBUF, COM2_TXBUF, COM2_RXBUF)

*************************************************************************

MS_SEG   SEGMENT     CODE            ;RELOCATABLE CODE SEGMENT
         RSEG        COMMS_SEG

*************************************************************************
 NB:                                                                    *
    REGISTER BANK 01 IS SELECTED FOR THIS ROUTINE, WHICH IS DEDICATED   *
    TO THE COMMS INTERRUPT ROUTINE.                                     *
                                                                        *
 THE STRING TO BE TRANSMITTED IS CONSTRUCTED IN THIS ROUTINE. TX_DATA   *
 BUFFER MUST BE SCANNED FOR A 'DLE', IF FOUND A 'DLE' MUST BE INSERTED. *
 ALSO CALCULATE BCC                                                     *
 STRING FORMAT:  DLE, STX, TEXT, DLE, ETX, BCC                          *
                 TEXT = DEVICE #, DATA                                  *
                                                                        *
 R0 USED FOR READING TX_DATA BUFFER                                     *
 R1 USED FOR WRITING TO TX_BUF                                          *
```

```
; R3 USED IN INTERRUPT ROUTINE TO COUNT # OF TRANSMITTED CHARS          *
;     THIS IS USED IN CASE OF A RE-TRANSMIT                            *
; R7 USED TO COUNT THE # OF BYTES TRANSFERD FROM TX_DATA TO TX_BUF     *
;                                                                      *
; ON ENTRY:                                                            *
;          NOTHING                                                     *
; ON EXIT:                                                             *
;         R0 POINTS TO START OF TX_BUF, USED IN COMMS INT ROUTINE       *
;         TX_COUNT CONTAINS # OF BYTES IN TX_BUF                       *
;         TX_BUF CONTAINS TRANSMIT DATA                                *
;**********************************************************************
STRING:
        MOV         A,R1
        PUSH        ACC

        JNB         IS_DATA,INTERMEDIATE_JMP    ;0 = NO DATA IN BUFFERS
        MOV         TX_COUNT,#0
        MOV         P2,#0           ;IN CONJUNCTION WITH MOVX R0/R1
        MOV         R1,#TX_BUF      ;CREATING TX STRING
_STX:
        MOV         A,#DLE          ;DLE
        MOVX        @R1,A
        INC         TX_COUNT
        INC         R1

        MOV         A,#STX          ;STX
        MOVX        @R1,A
```

```
        INC         TX_COUNT
        INC         R1
XT:
        MOV         A,ADDR1         ;1 BYTE HEX ADDR
        SWAP        A
        ANL         A,#0F0H         ;MS NIBBLE OF ADDR
        MOV         B,A
        MOV         A,ADDR2         ;LS NIBBLE OF ADDR
        ANL         A,#0FH
        ORL         A,B             ;ACC CONTAINS 1 BYTE ADDR

        MOV         A,BIN_ADDR
        MOVX        @R1,A
        INC         TX_COUNT
        INC         R1

        MOV         R0,#TX_DATA     ;READING DATA STRING
        MOV         P2,#0           ;IN CONJUNCTION WITH MOVX R0/R1
        MOVX        A,@R0           ;# OF BYTES IN DATA STRING
        MOV         R7,A
        INC         R0              ;NEXT BYTE TO READ FROM TX_DATA
        SJMP        TX_LOOP1
ERMEDIATE_JMP:
        LJMP        TX_STRING_EXIT  ;JMPS ARE OUT OF RANGE
LOOP1:
        MOV         P2,#0           ;IN CONJUNCTION WITH MOVX R0/R1
        MOVX        A,@R0           ;READ DATA BUFFER
```

```
        CJNE    A,#DLE,WR_TX_BUF
        MOVX    @R1,A           ;IF DLE FOUND, INSERT A 2ND DLE
        INC     R1
        INC     TX_COUNT
R_TX_BUF:
        MOVX    @R1,A           ;WR TO TX_BUF
        INC     R0              ;NEXT BYTE TO READ
        INC     R1              ;NEXT BYTE TO WRITE
        INC     TX_COUNT
        DJNZ    R7,TX_LOOP1
        CLR     IS_DATA
E_ETX:
        MOV     A,#DLE          ;DLE
        MOVX    @R1,A
        INC     TX_COUNT
        INC     R1

        MOV     A,#ETX          ;ETX
        MOVX    @R1,A
        INC     TX_COUNT
        INC     R1
                                ;START OF BCC CALCULATION
        MOV     R0,#TX_BUF+2    ;EXCLUDE 'DLE.STX'  FROM BCC
        MOV     R7,#0           ;BCC COUNTER
        CLR     BCC_FLAG
C:
        MOVX    A,@R0           ;READ TX_BUF
```

```
        JB      BCC_FLAG,CHK_ETX
        CJNE    A,#DLE,CALC_BCC
        SETB    BCC_FLAG
        INC     R0
        SJMP    BCC             ;READ NEXT BYTE
K_ETX:
        CJNE    A,#ETX,CALC_BCC ;END OF DATA FIELD
        XRL     A,R7
        XCH     A,R7            ;SAVE BCC INTO R7
        SJMP    BCC_COMPLETE
C_BCC:
        XRL     A,R7
        XCH     A,R7            ;SAVE BCC INTO R7
        CLR     BCC_FLAG
        INC     R0
        SJMP    BCC
_COMPLETE:
        MOV     A,R7            ;BCC VALUE
        MOVX    @R1,A           ;WRITE BCC TO TX_BUF
        INC     TX_COUNT
BYTE:
        MOV     R3,TX_COUNT     ;DON'T CHANGE TX_COUNT, IN CASE RE-TX
        SETB    TX_BUF_FULL     ;CLEARED IN INT ROUTINE
        CLR     P3.5            ;ENABLE DS3695 TO TX, GOES THRU 7404
        MOV     P2,#0           ;IN CONJUNCTION WITH MOVX R0/R1
        MOV     R0,#TX_BUF      ;TX FIRST BYTE IN TX_BUF
        MOVX    A,@R0
```

```
        MOV         SBUF,A          ;START TX PROCESS
        MOV         R0,#TX_BUF+1    ;SET R0 = START ADDR +1  OF TX_BUF
                                    ;USED IN COMMS INT ROUTINE. FIRST
                                    ;BYTE IS USED TO INITIATE COMMS INT
STRING_EXIT:
        POP         ACC
        MOV         R1,A

        RET
;*******************************************************************************

;*******************************************************************************
; NB:                                                                          *
;     REGISTER BANK 01 IS SELECTED FOR THIS ROUTINE, WHICH IS DEDICATED        *
;     TO THE COMMS INTERRUPT ROUTINE.                                          *
;                                                                              *
; THIS ROUTINE WILL BE CALLED FROM COMMS INT ROUTINE, TO CHECK THE             *
; VARIOUS BUFFER FULL FLAGS. IF SET, TRANSFER DATA TO TX_DATA BUFFER, AND      *
; CLEAR THE CORRESPONDING FLAGS.                                               *
; MAIN TYPE  IE I/O  (1 BYTE), SUB TYPE (1 BYTE) IE DEVICE TYPE, THIS          *
; FOLLOWED BY THE DATA                                                         *
;                                                                              *
; MESSAGE FORMAT FOR TX_DATA:                                                  *
;             1 BYTE      BINARY TERMINAL #                                    *
;             2 BYTE      'I' = INPUT DEVICE                                   *
;             3 BYTE      'K' = KEYBOARD AND SCANNERSL, 'L' = LOCK             *
;                         'D' = DRAWER, 'P' = RECEIPT ON/OFF                   *
```

```
                'S' = SERIAL DEVICE (82530)                          *
4 BYTE          IN THE CASE OF THE LOCK INDICATES                    *
                LOCK STATUS                                          *
                                                                     *
4 BYTE          IN THE CASE OF THE KEYBOARD INDICATES                *
                KEY AND SCAN DATA, IN SEQUENTIAL ORDER               *
                                                                     *
4 BYTE          IN THE CASE OF THE RECEIPT ON/OFF INDICATES          *
                RECEIPT STATUS                                       *
                                                                     *
4 BYTE          IN THE CASE OF THE DRAWERS INDICATES                 *
                1 = DRAWER #1                                        *
                2 = DRAWER #2                                        *
                                                                     *
4 BYTE          IN THE CASE OF SERIAL DEVICE (82530) INDICATES       *
                1 = RX DATA ON CH A                                  *
                2 = RX DATA ON CH B                                  *
                                                                     *
5 BYTE          IN THE CASE OF THE DRAWERS INDICATES.                *
                DRAWER STATUS                                        *
                                                                     *
5 BYTE          IN THE CASE OF SERIAL DEVICE (82530) INDICATES       *
                BINARY # OF BYTES IN RX STRING (EXCLUDE              *
                ACTUAL BINARY#).                                     *
                                                                     *
R0 USED FOR READING DATA BUFFERS  FROM VARIOUS INPUT DEVICES         *
R1 USED FOR WRITING TO TX_DATA BUFFER                                *
```

```
;R4 AND R5 USED FOR TEMPORY STORAGE OF DPTR                              *
;R6 USED FOR GENERAL COUNTING                                           *
;R7 USED AS A COUNTER FOR THE NUMBER OF BYTES IN TX_DATA STRING          *
;    THIS IS USED IN TX_STRING, WHEN CHECKING FOR A 'DLE'               *
;************************************************************************
_TX_BUFFS:
        MOV         A,R1                    ;SAVE R0 AND R1
        PUSH        ACC
        MOV         A,R0
        PUSH        ACC

        JB          TX_BUF_FULL,INTER_CHK_EXIT1   ;TX_BUFFER IS FULL
        MOV         R7,#0
        MOV         P2,#0                 ;CONJUNCTION WITH MOVX R0/R1
        MOV         R1,#TX_DATA+1           ;CREATING TX_DATA  STRING

        MOV         A,#'I'                  ;MAIN TYPE, INPUT
        MOVX        @R1,A
        INC         R1
        INC         R7
_LOCK:
        JNB         LOCK_FLAG,CHK_KYB       ;LOCK STATUS HASN'T CHANGED
        JNB         LOCK_FLAG,CHK_KEY_QUE   ;LOCK STATUS HASN'T CHANGED
        CLR         LOCK_FLAG
        MOV         P2,#0                 ;IN CONJUNCTION WITH MOVX R0/R1

        MOV         A,#'L'                  ;SUB TYPE, LOCK
```

```
        MOVX    @R1,A
        INC     R1                      ;NEXT BYTE TO WRITE
        INC     R7

        MOV     R0,#LOCK_STAT           ;READ STATUS
        MOVX    A,@R0
        MOVX    @R1,A
        INC     R7
        LJMP    CHK_EXIT
;*************************************************************************
; REFER TO ADD_QUE FOR MORE DETAILS ON KEY DATA                          *
;*************************************************************************
K_KYB:
        JNB     KEY_FLAG,CHK_CRD        ;NO KEYS DEPRESSED
        CLR     KEY_FLAG
        MOV     P2,#0                ;IN CONJUNCTION WITH MOVX R0/R1
        MOV     R0,#KEY_BUF             ;SOURCE ADDRR

        MOV     A,#'K'                  ;SUB TYPE, KEY ENTRY
        MOVX    @R1,A
        INC     R1                      ;NEXT BYTE TO WRITE
        INC     R7
K_LOOP1:
        MOVX    A,@R0                   ;TRANSFER FROM KEY_BUF TO
        MOVX    @R1,A                   ;TX_DATA BUFFER
        INC     R0
        INC     R1
```

```
        INC         R7
        DJNZ        KEY_COUNT,CHK_LOOP1     ;NEXT BYTE TO TRANSFER
        MOV         KEY_COUNT,#0            ;CLEAR COUNTER
        LJMP        CHK_EXIT
K_KEY_QUE:
        JNB         QUE_FLAG,CHK_CRD        ;NO KEY/SCAN DATA
        CLR         QUE_FLAG
        CLR         QUE_FULL
        MOV         P2,#0                ;IN CONJUNCTION WITH MOVX R0/R1
        MOV         R0,#KEY_QUE             ;SOURCE ADDRR

        MOV         A,#'K'                  ;SUB TYPE, KEY/SCAN ENTRY
        MOVX        @R1,A
        INC         R1                      ;NEXT BYTE TO WRITE
        INC         R7
_LOOP1:
        MOVX        A,@R0                   ;TRANSFER FROM KEY_BUF TO
        MOVX        @R1,A                   ;TX_DATA BUFFER
        INC         R0
        INC         R1
        INC         R7
        DJNZ        QUE_COUNT,CHK_LOOP1     ;NEXT BYTE TO TRANSFER
        MOV         QUE_COUNT,#0            ;CLEAR COUNTER
        LJMP        CHK_EXIT
ER_CHK_EXIT1:
        LJMP        CHK_EXIT1               ;INTERMEDIATE JMP
_CRD:
```

```
        JNB         CRD_FLAG,CHK_REC        ;NO CARD ENTRY
        CLR         CRD_FLAG
    ************
        MOV         P2,#0                ;IN CONJUNCTION WITH MOVX R0/R1
        MOV         R0,#CRD_BUF             ;SOURCE ADDRR
        MOVX        A,@R0                   ;TRANSFER FROM CRD_BUF TO
        CJNE        A,#01,CHK_CRD10
        MOV         CRD_COUNT,#38
        SJMP        CHK_CRD11
K_CRD10:
        MOV         CRD_COUNT,#01
r_CRD11:
t*****************
        MOV         P2,#0                ;IN CONJUNCTION WITH MOVX R0/R1
        MOV         R0,#CRD_BUF             ;SOURCE ADDRR

        MOV         A,#'C'               ;SUB TYPE, MAGNETIC CARD ENTRY
        MOVX        @R1,A
        INC         R1                      ;NEXT BYTE TO WRITE
        INC         R7
_CRD1:
        MOVX        A,@R0                   ;TRANSFER FROM CRD_BUF TO
        MOVX        @R1,A                   ;TX_DATA BUFFER
        INC         R0
        INC         R1
        INC         R7
        DJNZ        CRD_COUNT,CHK_CRD1      ;NEXT BYTE TO TRANSFER
```

```
        MOV         CRD_COUNT,#0            ;CLEAR COUNTER
        LJMP        CHK_EXIT
_REC:
        JNB         REC_FLAG,CHK_DRAW1      ;RECEIPT STATUS HASN'T CHANGED
        CLR         REC_FLAG
        MOV         P2,#0                   ;IN CONJUNCTION WITH MOVX R0/R1
        MOV         R0,#REC_STAT

        MOV         A,#'P'                  ;SUB TYPE, REC STATUS
        MOVX        @R1,A
        INC         R1                      ;NEXT BYTE TO WRITE
        INC         R7

        MOVX        A,@R0                   ;READ STATUS
        MOVX        @R1,A                   ;WRITE TO TX_BUF
        INC         R7
        LJMP        CHK_EXIT
K_DRAW1:
        JNB         DRAW1_FLAG,CHK_DRAW2    ;DRAW STATUS HASN'T CHANGED
        CLR         DRAW1_FLAG
        MOV         P2,#0                   ;IN CONJUNCTION WITH MOVX R0/R1
        MOV         R0,#DRAW1_STAT

        MOV         A,#'R'                  ;SUB TYPE, DRAW STATUS
        MOVX        @R1,A
        INC         R1                      ;NEXT BYTE TO WRITE
        INC         R7
```

```
        MOV     A,#'1'                  ;DRAW #1
        MOVX    @R1,A
        INC     R1                      ;NEXT BYTE TO WRITE
        INC     R7

        MOVX    A,@R0                   ;READ STATUS
        MOVX    @R1,A                   ;WRITE TO TX_BUF
        INC     R7
        LJMP    CHK_EXIT
K_DRAW2:
        JNB     DRAW2_FLAG,CHK_ENQ_BUF  ;DRAW STATUS HASN'T CHANGED
        CLR     DRAW2_FLAG
        MOV     P2,#0                   ;IN CONJUNCTION WITH MOVX R0/R1
        MOV     R0,#DRAW2_STAT

        MOV     A,#'R'                  ;SUB TYPE, DRAW STATUS
        MOVX    @R1,A
        INC     R1                      ;NEXT BYTE TO WRITE
        INC     R7

        MOV     A,#'2'                  ;DRAW #2
        MOVX    @R1,A
        INC     R1                      ;NEXT BYTE TO WRITE
        INC     R7

        MOVX    A,@R0                   ;READ STATUS
        MOVX    @R1,A                   ;WRITE TO TX_BUF
```

```
        INC         R7
        LJMP        CHK_EXIT
ENQ_BUF:
        JNB         ENQ_FULL,CHK_RXCOM1     ;NO HOST ENQUIRY
        MOV         R6,#4
        MOV         P2,#0                   ;IN CONJUNCTION WITH MOVX
/R1
        MOV         R0,#ENQ_BUF             ;SOURCE ADDRR

        MOV         A,#'E'                  ;SUB TYPE, ENQUIRY
        MOVX        @R1,A
        INC         R1                      ;NEXT BYTE TO WRITE
        INC         R7
_TX_LOOP2:
        MOVX        A,@R0                   ;TRANSFER FROM ENQ_BUF TO
        MOVX        @R1,A                   ;TX_DATA BUFFER
        INC         R0
        INC         R1
        INC         R7
        DJNZ        R6,CHK_TX_LOOP2         ;NEXT BYTE TO TRANSFER
        CLR         ENQ_FULL
        LJMP        CHK_EXIT
K_RXCOM1:
        JNB         RXBUF1_FULL,CHK_RXCOM2  ;NO DATA FROM CH A OF 82530
        MOV         A,#'S'                  ;SUB TYPE, SERIAL DEVICE
        MOVX        @R1,A
        INC         R1                      ;NEXT BYTE TO WRITE
```

```
        INC         R7

        MOV         A,#'1'                  ;COM1
        MOVX        @R1,A
        INC         R1                      ;NEXT BYTE TO WRITE
        INC         R7

        MOV         A,RX1_COUNT             ;# OF BYTES IN DATA FIELD
        MOV         R6,A                    ;R6 USED IN TRANSFER LOOP
        MOVX        @R1,A
        INC         R1                      ;NEXT BYTE TO WRITE
        INC         R7

        MOV         DPTR,#COM1_RXBUF
        LCALL       TRANS_DATA              ;TRANSFER FROM COMXBUF TO TXBUF
        CLR         RXBUF1_FULL
        MOV         RX1_COUNT,#0
        SJMP        CHK_EXIT
_RXCOM2:
        JNB         RXBUF2_FULL,CHK_EXIT1   ;NO DATA FROM CH B OF 82530
        MOV         A,#'S'                  ;SUB TYPE, SERIAL DEVICE
        MOVX        @R1,A
        INC         R1                      ;NEXT BYTE TO WRITE
        INC         R7

        MOV         A,#'2'                  ;COM2
        MOVX        @R1,A
```

```
        INC     R1                      ;NEXT BYTE TO WRITE
        INC     R7

        MOV     A,RX2_COUNT             ;# OF BYTES IN DATA FIELD
        MOV     R6,A                    ;R6 USED IN TRANSFER LOOP
        MOVX    @R1,A
        INC     R1                      ;NEXT BYTE TO WRITE
        INC     R7

        MOV     DPTR,#COM2_RXBUF
        LCALL   TRANS_DATA              ;TRANSFER FROM COMXBUF TO TXBUF
        CLR     RXBUF2_FULL
        MOV     RX2_COUNT,#0
        SJMP    CHK_EXIT
EXIT:
        MOV     R1,#TX_DATA             ;THERE IS DATA TO TRANSMIT
        MOV     P2,#0                   ;IN CONJUNCTION WITH MOVX R0/R1
        MOV     A,R7                    ;# OF BYTES IN DATA STRING
        MOVX    @R1,A
        SETB    IS_DATA                 ;INDICATES TO TX_STRING THAT
EXIT1:
        POP     ACC
        MOV     R0,A
        POP     ACC
        MOV     R1,A
```

```
        RET
;*******************************************************************************


;*******************************************************************************
;TRANSFER DATA STRING FROM 1 BUFFER TO ANOTHER IN XDATA MEMORY.                 *
;DPTR POINTS TO SOURCE BUFFER                                                  *
;R1 POINTS TO TX_DATA BUFFER OF 8032                                           *
;R6 IS USED TO COUNT # OF BYTES TO TRANSFER                                    *
;                                                                             *
;ON ENTRY:                                                                     *
;    DPTR POINTS TO SOURCE BUFFER (COM1/2_RXBUF)                               *
;    R6 = # OF BYTES TO TRANSFER                                               *
;                                                                             *
;ON EXIT:                                                                      *
;    TX_BUF (8032) WILL CONTAIN DATA FROM COM1/2_RXBUF                         *
;*******************************************************************************
TS_DATA:
        MOVX        A,@DPTR                 ;READ FROM COM1/2_RXBUF
        INC         DPTR
        MOVX        @R1,A                   ;WRITE TO TX_DATA BUFFER
        INC         R1
        INC         R7
        DJNZ        R6,TRANS_DATA

        RET
;*******************************************************************************
```

```
*************************************************************************
 TEST COMMS                                                             *
*************************************************************************
T_COMMS:
        MOV         R0,#TX_BUF
        MOV         R1,#20H
        MOV         A,#30H
        MOV         P2,#0           ;IN CONJUNCTION WITH MOVX R0/R1
OOP1:
        MOVX        @R0,A
        INC         R0
        INC         A
        DJNZ        R1,T_LOOP1
                                    ;TX_BUF NOW CONTAINS STRING

        SETB        PSW.3           ;SELECT REGISTER BANK 1, 8H - FH
        MOV         TX_COUNT,#20H   ;USED IN COMMS INT ROUTINE
        MOV         R3,TX_COUNT     ;DON'T CHANGE TX_COUNT, IN CASE RE-TX
        MOV         R0,#TX_BUF      ;SET R0 = START ADDR OF TX_BUF
                                    ;USED IN COMMS INT ROUTINE
        CLR         PSW.3           ;SELECT REGISTER BANK 0
        SETB        TX_BUF_FULL     ;CLEARED IN INT ROUTINE
        CLR         P1.7            ;ENABLE DS3695 TO TX, GOES THRU 7404
        CLR         P3.5            ;ENABLE DS3695 TO TX, GOES THRU 7404
        MOV         SBUF,#0FH       ;START TX PROCESS
```

```
        RET
;*****************************************************************************

;*****************************************************************************
; THIS ROUTINE CHECKS IF RX_BUF_FULL FLAG IS SET, INDICATING THAT DATA     *
; HAS BEEN RECEIVED                                                        *
;                                                                          *
; MESSAGE FORMAT:                                                          *
;           1 BYTE      'O' = OUTPUT DEVICE                                *
;                                                                          *
;           2 BYTE      'L' = LEDS, 'D' = DISPLAY, 'B' = BUZZER            *
;                       'R' = DRAWER, 'C' = CARD READER, 'E' = HOST         *
;                       ENQUIRY, 'S' TRANSMIT STRING FOR 82530             *
;                                                                          *
;           3 BYTE      IN THE CASE OF AN LCD STRING INDICATES             *
;                       0 = BOTH DISPLAYS                                  *
;                       1 = LCD_1                                          *
;                       2 = LCD_2                                          *
;                                                                          *
;           3 BYTE      IN THE CASE OF THE DRAWERS INDICATES               *
;                       1 = DRAWER #1                                      *
;                       2 = DRAWER #2                                      *
;                                                                          *
;           3 BYTE      IN THE CASE OF THE CARD READER                     *
;                       '1' = TRANSMIT CARD DATA, ie READ CARD             *
;                       '0' = DON'T TRANSMIT CARD DATE ie DON'T READ       *
;                                                                          *
```

```
;       3 BYTE      IN THE CASE OF A HOST ENQUIRY                    *
;                   'R' = DRAWER STATUS                              *
;                   'L' = LOCK STATUS                                *
;                   'P' = RECEIPT STATUS                             *
;                                                                    *
;       3 BYTE      IN THE CASE OF TX STRING FOR 82530               *
;                   BINARY # OF DATA BYTES IN DATA FIELD             *
;                   + CMD/DATA BYTE (# OF DATA BYTES +1 FOR C/D)     *
;                                                                    *
;       4 BYTE      IN THE CASE OF THE DRAWERS INDICATES             *
;                   '1' = DRAWER #1                                  *
;                   '2' = DRAWER #2                                  *
;                                                                    *
;       4 BYTE      IN THE CASE OF TX STRING FOR 82530               *
;                   '1' = COM1                                       *
;                   '2' = COM2                                       *
;                                                                    *
;       5 BYTE      IN THE CASE OF TX STRING FOR 82530               *
;                   'C' = COMMAND STRING                             *
;                   'D' = DATA STRING                                *
;                                                                    *
; R0 USED FOR READING                                                *
; R1 USED FOR WRITING TO BUFFER                                      *
; COUNT USED TO COUNT # OF BYTES TO TRANSFER FROM RX_BUF TO OUTPUT_BUF *
;*********************************************************************
_RX_BUFFS:
        JB          RX_BUF_FULL,CHK_BUFS    ;DATA AVAIALBALE IN RX_BUF
```

```
        LJMP        CHK_RX_EXIT1
BUFS:
        MOV         DPTR,#RX_BUF
        MOVX        A,@DPTR          ;READ RX_BUF
        INC         DPTR
        CJNE        A,#'O',INTERMEDIATE_RX_EXIT ;NOT FOR AN OUTPUT DEVICE
        MOVX        A,@DPTR          ;READ RX_BUF
        INC         DPTR
LEDS:
        CJNE        A,#'L',CHK_LCD   ;LED'S
        LCALL       LED_ON_OFF       ;TRANSFER FROM RX_BUF TO LED_BUF
        LJMP        CHK_RX_EXIT
LCD:
        CJNE        A,#'D',CHK_BUZ   ;LCD'S
        SETB        LCD_FLAG
        MOV         LCD_COUNT,#0
        MOVX        A,@DPTR          ;READ LCD #(1/2/BOTH) FROM RX_BUF
        MOV         LCD_NUM,A        ;LCD NUMBER
        INC         DPTR
        DEC         RX_COUNT         ;REMOVE 'OD1/2' FROM RX_COUNT
        DEC         RX_COUNT
        DEC         RX_COUNT
        MOV         R1,#LCD_BUF      ;TRANSFER FROM TX_BUF TO LCD_BUF
        MOV         P2,#0            ;IN CONJUNCTION WITH MOVX R0/R1
LOOP2:
        MOV         A,RX_COUNT
        CJNE        A,#0,CHK0        ;# OF CHAR IN RX_BUF
```

```
        SJMP        ADD_EOS

?:
        MOV         A,LCD_COUNT
        CJNE        A,#32,CHK1      ;MAX OF 32 BYTES IN LCD_BUF
        SJMP        ADD_EOS
1:
        MOVX        A,@DPTR         ;READ RX_BUF
                                    ;CLEAR THE DISPLAY, BY SENDING AN EMPTY
                                    ;MESSAGE TO THE DISPLAY.
        CJNE        A,#DLE,CHK_SPACE    ;ETX REACHED, PLACE EOS
        SJMP        ADD_EOS
_SPACE:
        CJNE        A,#HT,WR_LCD_BUF ;CHK FOR HT, ie SPACES
        DEC         RX_COUNT        ;ELSE 2 EXTRA CHARS ARE TRANSFERED
        DEC         RX_COUNT        ;TO LCD_BUF
        INC         DPTR            ;FIND # OF SPACES TO INSERT
        MOVX        A,@DPTR
        ANL         A,#03FH         ;MASK OFF MS 2 BITS, ACC = # OF ' '
        MOV         B,A
        INC         DPTR
_SPACE:
        MOV         A,#' '
        MOV         P2,#0           ;IN CONJUNCTION WITH MOVX R0/R1
        MOVX        @R1,A           ;WRITE SPACES TO LCD_BUF
        INC         R1              ;INSERT NEXT SPACE
        INC         LCD_COUNT       ;COUNT # OF BYTES IN LCD_BUF
        MOV         A,#32           ;ENSURE ONLY 32 BYTES IN LCD_BUF
```

```
        CJNE        A,LCD_COUNT,INS_NEXT_SPACE;MAX OF 32 BYTES IN LCD_BUF
        SJMP        ADD_EOS
NEXT_SPACE:
        DJNZ        B,INS_SPACE
        SJMP        CHK_LOOP2
LCD_BUF:
        MOV         P2,#0           ;IN CONJUNCTION WITH MOVX R0/R1
        MOVX        @R1,A           ;WRITE TO OUTPUT_BUF
        INC         DPTR
        INC         R1
        INC         LCD_COUNT       ;COUNT # OF BYTES IN LCD_BUF
        DJNZ        RX_COUNT,CHK_LOOP2
EOS:
        MOV         A,#0            ;EOS
        MOV         P2,#0           ;IN CONJUNCTION WITH MOVX R0/R1
        MOVX        @R1,A
        SJMP        CHK_RX_EXIT
BUZ:
        CJNE        A,#'B',CHK_DRAWER1  ;BUZZER
        LCALL       BEEP
        SJMP        CHK_RX_EXIT
RMEDIATE_RX_EXIT:
        SJMP        CHK_RX_EXIT
DRAWER1:
        CJNE        A,#'R',CHK_CARD     ;NOT A DRAWER OPEN COMMAND
        MOVX        A,@DPTR
        CJNE        A,#'1',CHK_DRAWER2
```

```
        LCALL       DRAW1_OPEN
        SJMP        CHK_RX_EXIT
DRAWER2:
        CJNE        A,#'2',CHK_RX_EXIT
        LCALL       DRAW2_OPEN
        SJMP        CHK_RX_EXIT
_CARD:
        CJNE        A,#'C',CHK_ENQUIRE  ;CARD STATUS, ON = 31H, OFF = 30H
        MOVX        A,@DPTR
        CJNE        A,#'0',CARD_ON
        CLR         CRD_ON_OFF          ;DON'T TX CARD DATA
        SJMP        CHK_RX_EXIT
D_ON:
        CJNE        A,#'1',CHK_RX_EXIT
        SETB        CRD_ON_OFF          ;TX CARD DATA
        SJMP        CHK_RX_EXIT
ENQUIRE:
        CJNE        A,#'E',CHK_COMX     ;STATUS ENQUIRY FROM HOST
        SETB        ENQ_FLAG            ;MAIN LOOP CHKS THIS FLAG
        SJMP        CHK_RX_EXIT
_COMX:
        CJNE        A,#'S',CHK_RX_EXIT  ;CMD/DATA FOR COM1/2 ON 82530
        LCALL       RXBUF_COMXBUF       ;TRANSFER FROM RXBUF TO
/2_BUF
        SJMP        CHK_RX_EXIT
_RX_EXIT:
        CLR         RX_BUF_FULL         ;SET IN INT ROUTINE
```

```
RX_EXIT1:

        RET
;*******************************************************************************

;*******************************************************************************
;TRANSFER FROM RX_BUF(8032) TO COMX_BUF(82530)                                  *
;R6 AND R7 IS USED FOR TEMPORY STORAGE OF DPTR                                  *
;                                                                              *
;ON ENTRY:                                                                     *
;        DPTR POINTS TO SOURCE BUFFER (RX_BUF)                                 *
;                                                                              *
;ON EXIT:                                                                      *
;        COMX_BUF CONTAINS RELEVANT DATA                                       *
;        TXBUF1/2_FULL IS SET (USED WHEN TRANSMITTING VIA 82530)               *
;*******************************************************************************
F_COMXBUF:
        MOVX    A,@DPTR             ;CHK IF COM1 / COM2
        INC     DPTR
COM1:
        CJNE    A,#'1',CHK_COM2     ;CMD/DATA FOR COM1
        MOVX    A,@DPTR             ;# OF BYTES TO TRANSFER

        MOV     B,A                 ;ENSURE NOT > 50 BYTES
        MOV     A,#50
        CLR     C
        SUBB    A,B
```

```
        JNC         COM1_LP1
        MOV         A,#50
        MOVX        @DPTR,A             ;OVERWRITE # OF BYTES TO TX
        MOV         B,A
_LP1:
        MOV         R5,B                ;# OF BYTES TO TRANSFER
        MOV         R4,#01              ;USED TO SET TXBUF1_FULL
        PUSH        DPL
        PUSH        DPH
        MOV         DPTR,#COM1_TXBUF
        MOV         R6,DPL              ;SAVE COM1_TXBUF
        MOV         R7,DPH
        POP         DPH
        POP         DPL
        SJMP        RXBUF_LP1
_COM2:
        CJNE        A,#'2',RXBUF_EXIT   ;CMD/DATA FOR COM2
        MOVX        A,@DPTR             ;# OF BYTES TO TRANSFER

        MOV         B,A                 ;ENSURE NOT > 50 BYTES
        MOV         A,#50
        CLR         C
        SUBB        A,B
        JNC         COM2_LP1
        MOV         A,#50
        MOVX        @DPTR,A             ;OVERWRITE # OF BYTES TO TX
        MOV         B,A
```

```
2_LP1:
        MOV         R5,B                ;# OF BYTES TO TRANSFER
        MOV         R4,#02              ;USED TO SET TXBUF2_FULL
        PUSH        DPL
        PUSH        DPH
        MOV         DPTR,#COM2_TXBUF
        MOV         R6,DPL              ;SAVE COM2_TXBUF
        MOV         R7,DPH
        POP         DPH
        POP         DPL
UF_LP1:
        MOVX        A,@DPTR             ;READ SOURCE ADDR
        INC         DPTR
        PUSH        DPL                 ;SAVE DPTR
        PUSH        DPH
        MOV         DPL,R6
        MOV         DPH,R7
        MOVX        @DPTR,A             ;WRITE DESTINATION ADDR
        INC         DPTR
        MOV         R6,DPL              ;SAVE DESTINATION ADDR
        MOV         R7,DPH
        POP         DPH                 ;RESTORE SOURCE ADDR
        POP         DPL
        DJNZ        R5,RXBUF_LP1
_TXBUF_FULL:
        CJNE        R4,#01,CHK_TXBUF2
        SETB        TXBUF1_FULL
```

```
        SJMP        RXBUF_EXIT
TXBUF2:
        CJNE        R4,#02,RXBUF_EXIT
        SETB        TXBUF2_FULL
F_EXIT:

        RET
;*************************************************************************

;*************************************************************************
; HOST REQUEST VARIOUS INPUT STATUS CONDITIONS IE LOCK, DRAW ETC         *
; THE 4 BYTES WILL BE IN FOLLOWING SEQUENCE:                             *
;    DRAW1_STAT, DRAW2_STAT, RECEIPT_STAT, LOCK_STAT                     *
;*************************************************************************
IRE:
        JNB         ENQ_FLAG,ENQ_EXIT
        MOV         DPTR,#ENQ_BUF
        MOV         R0,#DRAW1_STAT
        MOV         P2,#0           ;IN CONJUNCTION WITH MOVX R0/R1
        MOVX        A,@R0
        MOVX        @DPTR,A
        INC         DPTR

        MOV         R0,#DRAW2_STAT
        MOV         P2,#0           ;IN CONJUNCTION WITH MOVX R0/R1
        MOVX        A,@R0
        MOVX        @DPTR,A
```

```
        INC         DPTR

        MOV         R0,#REC_STAT    ;RECEIPT ON/OFF
        MOV         P2,#0           ;IN CONJUNCTION WITH MOVX R0/R1
        MOVX        A,@R0
        MOVX        @DPTR,A
        INC         DPTR

        MOV         R0,#LOCK_STAT
        MOV         P2,#0           ;IN CONJUNCTION WITH MOVX R0/R1
        MOVX        A,@R0
        MOVX        @DPTR,A
        CLR         ENQ_FLAG
        SETB        ENQ_FULL        ;ENQ_BUF IS NOW FULL
EXIT:

        RET
;*******************************************************************************


LUDE (EQUATES.ASM)


        END
```

SCAN.ASM

```
E SCANNER_MODULE
      MODULE FOR SLOT SCANNER INTERFACE (OCIA)

*************************************************************************
LIC   TIMEOUT, SCAN_READ, INIT_SCAN, ADD_QUE, CHK_QUE

RN    BIT (TMOUT, SCAN_FLAG, QUE_FLAG, KEY_FLAG, QUE_FULL)

'N    DATA (SCAN_COUNT, QUE_COUNT, KEY_COUNT)

RN    XDATA (SCAN_BUF, KEY_QUE, KEY_BUF)

?N    CODE (WAIT, TEST, BEEP)
*************************************************************************

'_SEG     SEGMENT     CODE              ;RELOCATABLE CODE SEGMENT
          RSEG        SCAN_SEG

'*************************************************************************

*************************************************************************
CALL THIS ROUTINE BEFORE POLLING THE SCANNER OR KEYBOARD, ELSE DATA     *
WILL BE PLACED IN KEY_BUF OR SCAN_BUF, AND WE WOULD NOT NO WHICH CAME   *
FIRST. THIS WOULD THEREFORE DEFEAT THE WHOLE AIM OF THE KEY_QUE, WHICH  *
ENSURES THAT THE KEY/SCAN DATA IS IN THE SEQUENCE AS IT WAS INPUT BY    *
```

```
THE USER.                                                              *
                                                                       *
CHECK IF THE KEY QUE CAN ACCOMAODATE MORE SCAN OR KEY DATA             *
                                                                       *
ON ENTRY:                                                              *
        ACC = 'K' INDICATES KEY DATA                                   *
              'S' INDICATES SCAN DATA                                  *
                                                                       *
ON EXIT:                                                               *
        QUE_FULL FLAG WILL BE SET IF KEY_QUE IS FULL                   *
***********************************************************************

QUE:
KEY:
        CJNE        A,#'K',CHK_SCAN ;CHK IF KEY OR SCAN DATA
        CLR         C               ;CHK IF KEY_QUE IS FULL
        MOV         A,QUE_COUNT
        ADD         A,KEY_COUNT
        MOV         B,A
        MOV         A,#24
        SUBB        A,B
        JC          CHK_QUE_FULL        ;KEY_QUE IS FULL
        SJMP        CHK_QUE_EXIT
SCAN:
        CJNE        A,#'S',CHK_QUE_EXIT ;! KEY OR SCAN DATA, THEN ERROR
        CLR         C               ;CHK IF KEY_QUE IS FULL
        MOV         A,QUE_COUNT
        ADD         A,SCAN_COUNT
```

```
        MOV         B,A
        MOV         A,#24
        SUBB        A,B
        JC          CHK_QUE_FULL        ;KEY_QUE IS FULL
        SJMP        CHK_QUE_EXIT
_QUE_FULL:
        SETB        QUE_FULL
_QUE_EXIT:
        RET
;*************************************************************************
;*************************************************************************
;ONE HAS TO ENSURE THAT KEY DATA AND SCAN DATA ARE IN THE CORRECT        *
;SEQUENCE ENTERED. ONE CAN'T HAVE 1 KEYSTROKE WHICH WAS ENTERED          *
;AFTER A SCAN APPEARING BEFORE THE SCAN DATA AT THE HOST COMPUTER.       *
;TO OVERCOME THIS PROBLEM, A KEY_QUE IS USED, AND THE DATA FROM THE      *
;SCANNER AND KEYBOARD IS ENTERED SEQUENTIALLY INTO THIS BUFFER. THIS IS  *
;THE BUFFER WHICH WILL NOW BE TRANSMITTED TO THE HOST, CONTAINING BOTH   *
;KEY AND SCAN DATA.                                                      *
;                                                                        *
;ON ENTRY:                                                               *
;       ACC = 'K' INDICATES KEY DATA                                     *
;             'S' INDICATES SCAN DATA                                    *
;                                                                        *
;ON EXIT:                                                                *
;*************************************************************************
```

```
_QUE:
        MOV         B,A
        MOV         A,#KEY_QUE      ;CORRECT OFFSET INTO KEY_QUE
        ADD         A,QUE_COUNT
        MOV         R1,A            ;COMBINATION OF KEY & SCAN DATA
        MOV         A,B             ;RESTORE ACC
_K:
        CJNE        A,#'K',CHK_S    ;CHK IF KEY OR SCAN DATA
        MOV         R0,#KEY_BUF     ;SOURCE ADDRR
        MOV         R7,KEY_COUNT
        MOV         KEY_COUNT,#0
        CLR         KEY_FLAG
        SJMP        ADD_LP1
_S:
        CJNE        A,#'S',ADD_EXIT ;! KEY OR SCAN DATA, THEN ERROR
        MOV         R0,#SCAN_BUF    ;SOURCE ADDRR
        MOV         R7,SCAN_COUNT
        MOV         SCAN_COUNT,#0
        MOV         P2,#0           ;IN CONJUNCTION WITH MOVX R0/R1
        MOV         A,#0FFH         ;INDICATES SCAN DATA
        MOVX        @R1,A
        INC         R1
        INC         QUE_COUNT
        MOV         A,R7            ;INDICATES # OF BYTES IN SCAN FIELD
        MOVX        @R1,A
        INC         R1
        INC         QUE_COUNT
```

```
        CLR         SCAN_FLAG
LP1:
        MOV         P2,#0           ;IN CONJUNCTION WITH MOVX R0/R1
        MOVX        A,@R0           ;TRANSFER FROM KEY/SCAN_BUF TO
        MOVX        @R1,A           ;KEY_QUE
        INC         R0
        INC         R1
        INC         QUE_COUNT
        DJNZ        R7,ADD_LP1      ;NEXT BYTE TO TRANSFER
        SETB        QUE_FLAG        ;DATA  IN KEY_QUE
        SJMP        ADD_EXIT
FULL:
        SETB        QUE_FULL        ;PREVENT INPUT FROM KEY & SCAN
EXIT:

        RET
*****************************************************************************

*****************************************************************************
 INITIALIZE SLOT SCANNER                                                    *
*****************************************************************************
T_SCAN:
        MOV         SCAN_COUNT,#0
        CLR         SCAN_FLAG
        CLR         P1.1            ;CLK 'L'
        MOV         R7,#08
T_SCAN_LP1:
```

```
        SETB        P1.2            ;RESET 'H'
        LCALL       WAIT
        CLR         P1.2            ;RESET 'L'
        LCALL       WAIT
        DJNZ        R7,INIT_SCAN_LP1

        RET
;*************************************************************************

;*************************************************************************
; READ SLOT SCANNER                                                      *
;*************************************************************************
N_READ:
        JB          SCAN_FLAG,SCAN_EXIT     ;SCAN_BUF IS FULL
        JB          QUE_FULL,SCAN_EXIT      ;KEY_QUE IS FULL
        JB          P1.0,SCAN_EXIT  ;P1.0 = 0, DATA AVAILABLE
        MOV         R6,#20
        MOV         R0,#SCAN_BUF
        MOV         SCAN_COUNT,#0
N_READ_LP1:
        LCALL       TIMEOUT
_DATA:
        JB          P1.0,CHK_TMOUT  ;P1.0 = 0, DATA AVAILABLE
        SJMP        READ_BYTE
_TMOUT:
        JB          TMOUT,SCAN_ERR
        SJMP        CHK_DATA
```

```
AD_BYTE:
        MOV         B,#06          ;26 uSEC DELAY BEFORE READING NEXT BYTE
        LCALL       WAIT1
        LCALL       SCAN
        INC         SCAN_COUNT
        LCALL       TEST
        JNB         P1.6,$
        JB          P1.6,$

        MOV         B,A            ;SAVE ACC
        ANL         A,#3FH
        MOV         P2,#0          ;IN CONJUNCTION WITH MOVX R0/R1
        MOVX        @R0,A
        INC         R0
        JB          B.6,SCAN_EXIT1 ;LAST BYTE WAS READ
        DJNZ        R6,SCAN_READ_LP1
N_EXIT1:
        SETB        SCAN_FLAG      ;SCAN DATA AVAILABLE
        SJMP        SCAN_EXIT
_ERR:
        MOV         SCAN_COUNT,#0
N_EXIT:
        CLR         TMOUT

        RET
*****************************************************************************
```

```
;*********************************************************************
; READ 1 BYTE FROM SLOT SCANNER. (1 BYTE = 9 BITS)                   *
; BIT FORMAT: 1 BIT IS THE START BIT, FOLLOWED BY 6 DATA BITS (B0 - B5)  *
;FOLLOWED BY 1 BIT WHICH INDICATES THAT THIS WAS THE LAST BYTE READ  *
; (B6 = 1 LAST BYTE READ), FOLLOWED BY A PARITY BIT (B7)             *
;                                                                    *
; P1.0 = RETURN DATA                                                 *
; P1.1 = CLOCK                                                       *
; P1.2 = RESET                                                       *
;                                                                    *
; ON ENTRY:                                                          *
;        NOTHING                                                     *
;                                                                    *
; ON EXIT:                                                           *
;        ACC = BYTE READ FROM SCANNER                                *
;*********************************************************************
IN:
        MOV         R7,#09          ;9 BITS / BYTE
        CLR         A
        CLR         C
IN_LP1:
        SETB        P1.1            ;CLK 'H'
        MOV         B,#06           ;26 uSEC TIME DELAY
        LCALL       WAIT1
        CLR         P1.1            ;CLK 'L'
        MOV         B,#30           ;94 uSEC TIME DELAY
        LCALL       WAIT1
```

```
        RRC         A
        MOV         C,P1.0          ;READ DATA BITS
        DJNZ        R7,SCAN_LP1
        CPL         A
        MOV         B,#250          ;600 uSEC DELAY BEFORE READING NEXT BYTE
        LCALL       WAIT1           ;THIS LONG DELAY IS TO ACCOMADATE THE
                                       LAST BYTE

        RET
;*******************************************************************************

;*******************************************************************************
;USED FOR A TIMEOUT LOOP. DURATION OF TIMING PERIOD IS 170 mSEC, WITH NO *
;INTERUPTS TAKING PLACE.                                                 *
;                                                                        *
;ON ENTRY:                                                               *
;       NOTHING                                                          *
;ON EXIT:                                                                *
;       CHECK 'TMOUT' FLAG IF SET, THEN A TIMEOUT OCCUREED               *
;       NB: ENSURE TO CLEAR THE FLAG.                                    *
;*******************************************************************************
TOUT:
        CLR         TR0             ;STOP TIMER_0
        MOV         TH0,#0          ;USED FOR TIMEOUT LOOP
        MOV         TL0,#0
        SETB        TR0             ;START TIMER_0
```

```
        RET
;*******************************************************************************


;*******************************************************************************
; TIME DELAY. FIRST SETUP REG B WITH # OF TIMES TO LOOP                        *
;                                                                              *
; ON ENTRY:                                                                    *
;       B = # OF TIMES TO LOOP                                                 *
; ON EXIT:                                                                     *
;       NOTHING                                                                *
;*******************************************************************************
T1:
T1_LP1:
        DJNZ        B,WAIT1_LP1

        RET
;*******************************************************************************


CLUDE (EQUATES.ASM)

        END
```

```
;*****************************************************************************
;NOTE1:                                                                       *
;NB!!!!!!!!!!    A COMMON ERROR WHEN PUSHING A VALUE ONTO THE STACK          *
;                INSIDE A LOOP, IS FORGETTING TO RESTORE THE STACK TO        *
;                IT'S CORRECT OFFSET WHEN ONE JUMPS OUT OF THE LOOP          *
;                WHEN AN ERROR CONDITION OCCURS. NORMALLY INSIDE THE         *
;                LOOP THE PUSHES AND POPS TO THE STACK WILL BE FINE,         *
;                IT'S ONLY WHEN AN ERROR CONDITION OCCURS AND WE JUMP        *
;                OUT OF  THE LOOP, AND THE PREVIOUS VALUES WHICH WE          *
;                PUSHED ONTO THE STACK IS STILL ON THE STACK. AN             *
;                ASSOSIATED 'POP' HAS TO TAKE PLACE INSIDE THE               *
;                'ERROR ROUTINE' TO PLACE THE STACK AT THE CORRECT           *
;                OFFSET.                                                     *
;*****************************************************************************

;*****************************************************************************
;NOTE2:                                                                       *
;NBB!!!!!!!!!!                                                               *
;   IF THE NUMBER OF TRANSMIT DATA BITS IS CHANGED IN WR5, THEN THE           *
;   VALUE THAT IS WRITTEN TO THIS REGISTER HAS TO BE ENTERED INTO             *
;   'RTSA' AND 'RTSB', IN THE FUNCTION 'INIT82530VARS'.                       *
;   THE REASON WHY WE KEEP AN IMAGE OF THIS REGISTER, IS THAT WR5             *
;   CANNOT BE READ, SO THAT WHEN WE TOGGLE THE RTS LINE WE USE THE            *
;   VALUE STORED IN 'RTSA/B', MODIFY THIS VALUE, AND WRITE IT TO WR5.         *
;*****************************************************************************
```

```
    COMMS_82530_MODULE
    MODULE FOR 82530 COMMS CHIP

*************************************************************************
IC  INIT_82530, B82530, A82530, SETUP_TX, TX_COMX, CODE_TO_XDATA
IC  INIT_82530VARS, SETUP_RX

RN  BIT (TMOUT, TXBUF1_FULL, TXBUF2_FULL, RXBUF1_FULL, RXBUF2_FULL)

RN  DATA (TX82_STAT, COUNT82, PARITY, ST_BITS, TX_DATA_BITS)
RN  DATA (RX_DATA_BITS, BAUD, RTSA, RTSB, RX1_COUNT, RX2_COUNT)

RN  XDATA (COM1_TXBUF, COM2_TXBUF, COM1_RXBUF, COM2_RXBUF)

RN  CODE (WAIT, TEST, BEEP, CLS1, CODE_TO_LCD, LCD_WR )

*************************************************************************

MS_82530_SEG SEGMENT     CODE            ;RELOCATABLE CODE SEGMENT
        RSEG         COMMS_82530_SEG

*************************************************************************
```

```
;*************************************************************************
;SETUP DPTR WITH SOURCE BUFFER, COM1/2_RXBUF. R7 WITH RX1/2_COUNT        *
;                                                                        *
;DEBUG 82530 RECIEVE SOFTWARE                                           *
;*************************************************************************

;*************************************************************************
;RECIEVE BYTE FROM 82530, PLACE THIS IN COM1/2_RXBUF                     *
;USE R7 IN CONJUNCTION WITH FUNCTION 'CHK_CHAN' AS A LOOKUP FOR THE      *
;ADDRESSES FOR THE 82530.                                               *
;                                                                       *
;ON ENTRY:                                                              *
;    DPTR POINTS TO SOURCE BUFFRER (COM1_RXBUF/COM2_RXBUF)              *
;                                                                       *
;    R7 = 2    COMDATB         ;DATA REG. CH B                          *
;         4    COMDATA         ;DATA REG. CH A                          *
;                                                                       *
;    REG 6 = # OF BYTES CURENTLY IN COM1/2_RXBUF SO AS TO PLACE NEW     *
;            RX BYTE AT CORRECT OFFSET                                  *
;                                                                       *
;ON EXIT:                                                               *
;    RX1/2_COUNT = NEW # OF BYTES IN COM1/2_RXBUF (ALSO USED IN ROUTINE *
;    TO TRANSMIT DATA TO THE HOST)                                      *
;*************************************************************************
COMX:
        MOV         A,#40                   ;ENSURE NOT > 40 BYTES
        CLR         C
```

```
        SUBB        A,R6
        JNC         RX_COMX_LP1
        MOV         R6,#40
        SJMP        RX_EXIT
OMX_LP1:
        PUSH        DPL
        PUSH        DPH
        DEC         R7              ;LOAD DPTR WITH CORRECT CMD REG OFFSET
        LCALL       CHK_CHAN
        INC         R7
        MOV         A,#01
        MOVX        @DPTR,A         ;SELECT RR1
        MOVX        A,@DPTR         ;RR1 FOR ERR CONDITION
        MOV         B,A             ;SAVE RR1
        MOVX        A,@DPTR         ;RR0 CHECK IF CHAR AVAIL
        POP         DPH
        POP         DPL

        JNB         ACC.0,RX_EXIT   ;1 = CHAR AVAILABLE
        MOV         A,B             ;CONTENTS OF RR1, PARITY/OVERUN ERR
        ANL         A,#70H          ;CHECK FOR PARITY/OVERUN/FRAME  ERR
        CJNE        A,#0,PAR_ERR    ;!= 0, THEREFORE ERR CONDITION
D_CHAR:
        PUSH        DPL
        PUSH        DPH
        LCALL       CHK_CHAN
        MOVX        A,@DPTR         ;READ CHAR
```

```
        MOV         B,A             ;SAVE RX CHAR
        POP         DPH
        POP         DPL

        CLR         C               ;CALC CORRECT OFFSET FOR COM1/2_RXBUF
        MOV         A,R6
        ADDC        A,DPL
        MOV         DPL,A
        JNC         RX_COMX_LP2
        INC         DPH
OMX_LP2:
        MOV         A,B             ;RESTORE RX CHAR
        MOVX        @DPTR,A         ;WR RX CHAR TO COM1/2_RXBUF
        INC         R6
        SJMP        RX_COMX         ;CHK IF ANOTHER BYTE AVAILABLE
L_ERR:
        DEC         R7              ;LOAD DPTR WITH CORRECT CMD REG
SET
        LCALL       CHK_CHAN
        INC         R7
        MOV         A,#30H          ;ERROR RESET
        MOVX        @DPTR,A         ;WR0
EXIT:

        RET
*****************************************************************************
```

```
;*************************************************************************
SETUP DPTR WITH SOURCE BUFFER, R7 WITH VALUE TO ADDRESS OF 82530,        *
COUNT82 FOR # OF BYTES TO TRANSMIT, BEFORE CALLING TX_COMX.              *
                                                                         *
FORMAT OF DATA FIELD:                                                    *
                    'C'MD / 'D'ATA                                       *
                    # OF BYTES IN DATA FILED, INCLUDING C/D & # OF       *
                    BYTES IN DATA FIELD                                  *
                    DATA STRING                                          *
                                                                         *
FORMAT OF COMMAND FIELD:                                                 *
                    BAUD RATE, BINARY 48, 96, 19, 38(DEFUALT 9600)       *
                    DATA BITS, BINARY 7, 8          (DEFAULT 7)          *
                    STOP BITS, BINARY 1, 2          (DEFAULT 2)          *
                    PARITY  'N', 'O', 'E'           (DEFAULT 'E')        *
                                                                         *
NB: COMMAND STRING MUST BE IN ABOVE SEQUENCE                             *
                                                                         *
ON ENTRY:                                                                *
    NOTHING                                                              *
                                                                         *
ON EXIT:                                                                 *
    DPTR POINTS TO SOURCE BUFFRER (COM1_TXBUF / COM2_TXBUF)              *
                                                                         *
    R7 = 1    COMSCB          ;COMMS STATUS/CONTROL REG. CH B            *
         2    COMDATB         ;DATA REG. CH B                            *
         3    COMSCA          ;COMMS STATUS/CONTROL REG. CH A            *
```

```
        4    COMDATA         ;DATA REG. CH A                                *
                                                                            *
    COUNT82 = # OF BYTES TO WRITE TO 82530                                  *
                                                                            *
*****************************************************************************
UP_TX:
        JNB         TXBUF1_FULL,CHK_TXBUF2
        MOV         DPTR,#COM1_TXBUF
        MOVX        A,@DPTR          ;# OF BYTES TO TX
        INC         DPTR
        DEC         A                ;REMOVE # OF BYTES & CMD/DATA BYTE
        DEC         A                ;FROM DATA STRING
        MOV         COUNT82,A
        MOVX        A,@DPTR          ;CHECK IF CMD/DATA
        INC         DPTR
_CMD1:
        CJNE        A,#'C',CHK_DAT1 ;CHECK IF CMD FOR COM1
        LCALL       COMMS_OPTIONS
ATE_RTSA:
        MOV         R0,#RTSA
        MOV         A,TX_DATA_BITS
        MOV         @R0,A

        MOV         DPTR,#COM1_TXBUF    ;SETUP REGS FOR CODE_TO_XDATA
        MOV         R6,DPL
        MOV         R7,DPH
        MOV         DPTR,#A82530
```

```
        MOV         COUNT82,#22
        LCALL       CODE_TO_XDATA       ;COM1_TXBUF CONTAINS ORIGINAL
                                        ;INTIALIZATION CODE

        MOV         DPTR,#COM1_TXBUF
        LCALL       SETUP_CMD_STR

        MOV         DPTR,#COM1_TXBUF    ;INIT CHANNEL A OF 82530
        MOV         R7,#03
        MOV         COUNT82,#22
        LCALL       INIT_82530
        CLR         TXBUF1_FULL
        SJMP        SETUP_EXIT
_DAT1:
        CJNE        A,#'D',SETUP_EXIT   ;CHECK IF DATA FOR COM1

        MOV         A,RTSA          ;RTS 'H'
        CLR         ACC.2
        MOV         R7,#03
        LCALL       TOGGLE_RTS

        MOV         R7,#04
        LCALL       TX_COMX         ;DPTR POINTS TO START OF DATA STRING
                                    ;OF COM1_TXBUF
        CLR         TXBUF1_FULL
        MOV         A,RTSA          ;RTS 'L'
        SETB        ACC.2
```

```
        MOV         R7,#03
        LCALL       TOGGLE_RTS
        SJMP        SETUP_EXIT
_TXBUF2:
        JNB         TXBUF2_FULL,SETUP_EXIT
        MOV         DPTR,#COM2_TXBUF
        MOVX        A,@DPTR             ;# OF BYTES TO TX
        INC         DPTR
        DEC         A                   ;REMOVE # OF BYTES & CMD/DATA BYTE
        DEC         A                   ;FROM DATA STRING
        MOV         COUNT82,A
        MOVX        A,@DPTR             ;CHECK IF CMD/DATA
        INC         DPTR
_CMD2:
        CJNE        A,#'C',CHK_DAT2 ;CHECK IF CMD FOR COM2
        LCALL       COMMS_OPTIONS
ATE_RTSB:
        MOV         R0,#RTSB
        MOV         A,TX_DATA_BITS
        MOV         @R0,A

        MOV         DPTR,#COM2_TXBUF    ;SETUP REGS FOR CODE_TO_XDATA
        MOV         R6,DPL
        MOV         R7,DPH
        MOV         DPTR,#B82530
        MOV         COUNT82,#22
        LCALL       CODE_TO_XDATA       ;COM2_TXBUF CONTAINS ORIGINAL
```

```
                                    ;INTIALIZATION CODE

        MOV         DPTR,#COM2_TXBUF
        LCALL       SETUP_CMD_STR

        MOV         DPTR,#COM2_TXBUF    ;INIT CHANNEL B OF 82530
        MOV         R7,#01
        MOV         COUNT82,#22
        LCALL       INIT_82530
        CLR         TXBUF2_FULL
        SJMP        SETUP_EXIT
_DAT2:
        CJNE        A,#'D',SETUP_EXIT   ;CHECK IF DATA FOR COM2
        MOV         A,RTSB              ;RTS 'H'
        CLR         ACC.2
        MOV         R7,#01
        LCALL       TOGGLE_RTS

        MOV         R7,#02
        LCALL       TX_COMX             ;DPTR POINTS TO START OF DATA STRING
                                        ;OF COM2_TXBUF
        CLR         TXBUF2_FULL

        MOV         A,RTSB              ;RTS 'L'
        SETB        ACC.2
        MOV         R7,#01
        LCALL       TOGGLE_RTS
```

```
        SJMP        SETUP_EXIT
UP_EXIT:

        RET
;*******************************************************************************

;*******************************************************************************
; TOGGLE THE RTS LINE                                                          *
;                                                                              *
; ON ENTRY:                                                                    *
;     DPTR POINTS TO SOURCE BUFFRER (COM1_TXBUF/COM2_TXBUF)                    *
;     R7 = 1    COMSCB          ;COMMS STATUS/CONTROL REG. CH B                *
;          3    COMSCA          ;COMMS STATUS/CONTROL REG. CH A                *
;                                                                              *
;     ACC = MODIFIED RTS BYTE                                                  *
;                                                                              *
; ON EXIT:                                                                     *
;     DPTR POINTS TO SOURCE BUFFRER (COM1_TXBUF/COM2_TXBUF)                    *
;*******************************************************************************
GLE_RTS:
        PUSH        DPL
        PUSH        DPH
        LCALL       CHK_CHAN
        MOV         B,A             ;SAVE RTS STATUS
        MOV         A,#05           ;WR 5
        MOVX        @DPTR,A
        MOV         A,B             ;RESTORE RTS STATUS
```

```
        MOVX    @DPTR,A
        POP     DPH
        POP     DPL

        RET
;*******************************************************************************


;*******************************************************************************
; SETUP INITIALIZATION STRING TO INITIALIZE 82530                              *
;                                                                              *
; ON ENTRY:                                                                    *
;     DPTR POINTS TO SOURCE BUFFRER (COM1_TXBUF/COM2_TXBUF)                    *
;*******************************************************************************
TUP_CMD_STR:
        MOV     R0,#RX_DATA_BITS ;READ RX BIT RATE
        MOV     A,@R0
        INC     DPTR
        INC     DPTR
        INC     DPTR
        MOVX    @DPTR,A          ;SET NEW BIT RATE

        MOV     R0,#PARITY       ;READ PARITY BYTE, INCLUDES ST_BITS
        MOV     A,@R0
        INC     DPTR
        INC     DPTR
        MOVX    @DPTR,A          ;SET NEW PARITY AND STOP BITS
        MOV     R0,#TX_DATA_BITS ;READ TX BIT RATE
```

```
        MOV         A,@R0
        INC         DPTR
        INC         DPTR
        MOVX        @DPTR,A         ;SET NEW TX BIT RATE

        MOV         R0,#BAUD        ;READ NEW BAUD RATE
        MOV         A,@R0
        INC         DPTR
        INC         DPTR
        INC         DPTR
        INC         DPTR
        INC         DPTR
        INC         DPTR
        INC         DPTR
        MOVX        @DPTR,A         ;SET NEW BAUD RATE

        RET
*******************************************************************************


*******************************************************************************
  SETUP COMMS OPTIONS TO INITIALIZE 82530 FROM HOST                            *
                                                                               *
  ON ENTRY:                                                                    *
      DPTR POINTS TO SOURCE BUFFRER (COM1_TXBUF/COM2_TXBUF)                    *
      OFFSET @ BAUD RATE                                                       *
*******************************************************************************
MS_OPTIONS:
        MOVX        A,@DPTR         ;SET BAUD RATE
```

```
        INC         DPTR
K48:                                ;SET BAUD RATE
        CJNE        A,#48H,CHK96    ;4.8K BAUD
        MOV         A,#16H
        SJMP        SET_BAUD
96:
        CJNE        A,#96H,CHK19    ;9.6K BAUD
        MOV         A,#0AH
        SJMP        SET_BAUD
K19:
        CJNE        A,#19H,CHK38    ;19.2K BAUD
        MOV         A,#04
        SJMP        SET_BAUD
K38:
        CJNE        A,#38H,CHK_DATA_BITS    ;38.4K BAUD
        MOV         A,#01
_BAUD:
        MOV         R0,#BAUD
        MOV         @R0,A

K_DATA_BITS:
        MOVX        A,@DPTR         ;SET # OF DATA BITS
        INC         DPTR
K7:
        CJNE        A,#7H,CHK8
        MOV         A,#41H          ;# OF RX BITS
        MOV         B,#28H          ;# OF TX BITS
```

```
        SJMP    SET_DATA_BITS

'K8:
        CJNE    A,#8H,CHK_ST_BITS

        MOV     A,#0C1H         ;# OF RX BITS

        MOV     B,#68H          ;# OF TX BITS

:T_DATA_BITS:
        MOV     R0,#RX_DATA_BITS

        MOV     @R0,A

        MOV     R0,#TX_DATA_BITS

        MOV     @R0,B

HK_ST_BITS:
        MOVX    A,@DPTR         ;SET # OF STOP BITS

        INC     DPTR

HK1:
        CJNE    A,#1H,CHK2

        MOV     A,#47H          ;1 STOP, EVEN PARITY

        SJMP    SET_STOP_BITS

'K2:
        CJNE    A,#8H,CHK_ST_BITS

        MOV     A,#4FH          ;2 STOP, EVEN PARITY

'T_STOP_BITS:
        MOV     R0,#ST_BITS

        MOV     @R0,A

'K_PARITY:
        MOVX    A,@DPTR         ;SET PARITY

        INC     DPTR
```

```
K_E:
        CJNE    A,#'E',CHK_N
        MOV     A,#03H          ;EVEN PARITY
        SJMP    SET_PARITY
HK_N:
        CJNE    A,#'N',CHK_O
        MOV     A,#00H          ;NO PARITY
        SJMP    SET_PARITY
K_O:
        CJNE    A,#'O',CO_EXIT
        MOV     A,#01H          ;ODD PARITY
ET_PARITY:
        MOV     R0,#PARITY
        MOV     @R0,A

        MOV     R0,#ST_BITS
        MOV     A,@R0
        ANL     A,#0FCH         ;REMOVE LOWER 2 BITS
        ORL     PARITY,A
_EXIT:

        RET
;*******************************************************************************
```

```
;*************************************************************************
; TRANSMIT DATA STRING TO COM1/2.                                        *
; USE R7 IN CONJUNCTION WITH FUNCTION 'CHK_CHAN' AS A LOOKUP FOR THE     *
; ADDRESSES FOR THE 82530.                                               *
; TX82_STAT IS TO INDICATE WETHER TRANSMISSION WAS SUCSESSFUL OR NOT.    *
; THIS STATUS BYTE APPLIES TO BOTH COM1 & COM2.                          *
;                                                                        *
; ON ENTRY:                                                              *
;     DPTR POINTS TO SOURCE BUFFRER (COM1_TXBUF/COM2_TXBUF)              *
;                                                                        *
;     R7 = 1    COMSCB          ;COMMS STATUS/CONTROL REG. CH B          *
;          2    COMDATB         ;DATA REG. CH B                          *
;          3    COMSCA          ;COMMS STATUS/CONTROL REG. CH A          *
;          4    COMDATA         ;DATA REG. CH A                          *
;                                                                        *
;     COUNT82 = # OF BYTES TO WRITE TO 82530                             *
;                                                                        *
; ON EXIT:                                                               *
;     NOTHING                                                            *
;*************************************************************************
COMX:
LP4:
        MOVX        A,@DPTR         ;READ SOURCE ADDR, COM1/2_TXBUF
        MOV         B,A             ;SAVE ACC
        INC         DPTR
        PUSH        DPL             ;SAVE DPTR
        PUSH        DPH
```

```
        LCALL       CHK_CTS         ;CHECK IF DEVICE READY TO ACCEPT DATA
        MOV         A,TX82_STAT     ;CHECK IF CTS ROUTINE TIMED OUT
        CJNE        A,#0FFH,TX_LP1
        SJMP        TX_ERR
*     DPTR WILL POINT TO COMSCA / COMSCB                                    *
*     TX82_STAT = 0FFH    CTS ERROR                                         *

X_LP1:
        CLR         TR0             ;STOP TIMER_0
        MOV         TH0,#0          ;USED FOR TIMEOUT LOOP
        MOV         TL0,#0          ;80 msec DELAY
        SETB        TR0             ;START TIMER_0
X_LP2:
        MOVX        A,@DPTR         ;WAIT FOR TX REG TO BE EMPTY
        JNB         ACC.2,CHK_TMOUT1
        SJMP        TX_LP3
K_TMOUT1:
        JB          TMOUT,TX_ERR
        SJMP        TX_LP2
_LP3:
        CLR         TR0             ;STOP TIMER_0
        LCALL       CHK_CHAN        ;LOAD DPTR WITH ADDR OF 82530
        MOV         A,B             ;RESTORE ACC
        MOVX        @DPTR,A         ;WRITE TO 82530
        POP         DPH             ;RESTORE DPTR
        POP         DPL
        DJNZ        COUNT82,TX_LP4  ;TX NEXT BYTE
```

```
        SJMP        EXIT_TX_COMX
_ERR:
        CLR         TMOUT
        POP         DPH             ;RESTORE STACK TO CORRECT POSITION
        POP         DPL
        LCALL       CLS1
        MOV         DPTR,#MSG3
        LCALL       CODE_TO_LCD
        MOV         DPTR,#LCD1_DAT_WR
        LCALL       LCD_WR
XIT_TX_COMX:

        RET
;*******************************************************************************

;*******************************************************************************
; CHECK WHICH CHANNEL IS TO BE USED, SO AS TO CHECK THE CORRECT CTS LINE  *
; NB: THE DPTR WILL BE CHANGED DEPENDING  ON THE CONTENTS OF R7           *
;                                                                         *
; ON ENTRY:                                                               *
;     R7 = 1   COMSCB         ;COMMS STATUS/CONTROL REG. CH B             *
;          2   COMDATB        ;DATA REG. CH B                             *
;          3   COMSCA         ;COMMS STATUS/CONTROL REG. CH A             *
;          4   COMDATA        ;DATA REG. CH A                             *
;                                                                         *
; ON EXIT:                                                                *
;     DPTR WILL POINT TO COMSCA / COMSCB                                  *
```

```
;     TX82_STAT = 0FFH     CTS ERROR                                          *
;*************************************************************************
HK_CTS:
        MOV         TX82_STAT,#0
        CJNE        R7,#02,CTS_LP1  ;CH B
        MOV         DPTR,#COMSCB    ;RR0 FOR CTS STATUS
        SJMP        CTS_LP2
TS_LP1:
        CJNE        R7,#04,EXIT_CHK_CTS ;CH A
        MOV         DPTR,#COMSCA
TS_LP2:
        CLR         TR0             ;STOP TIMER_0
        MOV         TH0,#0          ;USED FOR TIMEOUT LOOP
        MOV         TL0,#0          ;80 msec DELAY
        SETB        TR0             ;START TIMER_0
TS_LP3:
        MOVX        A,@DPTR         ;WAIT FOR CTS 'L'
        JNB         ACC.5,CHK_CTS_TMOUT ;BIT =1, THEN CTS = 0 ON 82530
        SJMP        EXIT_CHK_CTS
_CTS_TMOUT:
        JB          TMOUT,CTS_ERR
        SJMP        CTS_LP3
_ERR:
        CLR         TMOUT           ;CLEAR TMOUT FLAG
        MOV         TX82_STAT,#0FFH ;CTS ERR
_CHK_CTS:
        CLR         TR0             ;STOP TIMER_0
```

```
        RET
;*******************************************************************************

;*******************************************************************************
;* CHECK WHICH CHANNEL IS TO BE USED, AS WELL AS IF IT IS A COMMAND OR      *
;* DATA STRING.                                                            *
;* NB: THE DPTR WILL BE CHANGED DEPENDING  ON THE CONTENTS OF R7           *
;*                                                                         *
;* ON ENTRY:                                                               *
;*     R7 = 1   COMSCB          ;COMMS STATUS/CONTROL REG. CH B            *
;*          2   COMDATB         ;DATA REG. CH B                            *
;*          3   COMSCA          ;COMMS STATUS/CONTROL REG. CH A            *
;*          4   COMDATA         ;DATA REG. CH A                            *
;*                                                                         *
;* ON EXIT:                                                                *
;*     DPTR WILL POINT TO ONE OF THE ABOVE ADDRESSES                       *
;*******************************************************************************
HK_CHAN:
        CJNE        R7,#01,CHK_LP1  ;CH B, COMMAND
        MOV         DPTR,#COMSCB
        SJMP        EXIT_CHK_CHAN
HK_LP1:
        CJNE        R7,#02,CHK_LP2  ;CH B, DATA
        MOV         DPTR,#COMDATB
        SJMP        EXIT_CHK_CHAN
HK_LP2:
        CJNE        R7,#03,CHK_LP3  ;CH A, COMMAND
```

```
        MOV         DPTR,#COMSCA
        SJMP        EXIT_CHK_CHAN
_LP3:
        CJNE        R7,#04,EXIT_CHK_CHAN    ;CH A, DATA
        MOV         DPTR,#COMDATA
IT_CHK_CHAN:

        RET
;*************************************************************************

;*************************************************************************
;  INITIALIZE VARIABLES FOR 82530 MODULE                                 *
;*************************************************************************
IT_82530VARS:
        CLR         TXBUF1_FULL
        CLR         TXBUF2_FULL
        CLR         RXBUF1_FULL
        CLR         RXBUF2_FULL

        MOV         RX1_COUNT,#0
        MOV         RX2_COUNT,#0
        MOV         TX82_STAT,#0
        MOV         PARITY,#0
        MOV         ST_BITS,#0
        MOV         TX_DATA_BITS,#0
        MOV         RX_DATA_BITS,#0
        MOV         BAUD,#0
```

```
* NB!!!!  PLEASE REFER TO NOTE5, @ START OF MODULE CONCERNING RTSA/B      *

         MOV          RTSA,#6AH
         MOV          RTSB,#6AH

         RET
*************************************************************************

*************************************************************************
* INITIALIZE 82530                                                      *
*                                                                       *
* ON ENTRY:                                                             *
*     DPTR POINTS TO SOURCE BUFFRER (COM1_TXBUF/COM2_TXBUF)             *
*                                                                       *
*     R7 = 1   COMSCB          ;COMMS STATUS/CONTROL REG. CH B          *
*          2   COMDATB         ;DATA REG. CH B                          *
*          3   COMSCA          ;COMMS STATUS/CONTROL REG. CH A          *
*          4   COMDATA         ;DATA REG. CH A                          *
*                                                                       *
*     COUNT82 = # OF BYTES TO WRITE TO 82530                            *
*                                                                       *
* ON EXIT:                                                              *
*     NOTHING                                                           *
*************************************************************************
IT_82530:
         PUSH         DPL
         PUSH         DPH
```

```
_82530:                         ;CHECK IF 82530 IS WORKING
        LCALL   CHK_CHAN
        MOV     A,#0CH          ;SELECT REG C
        MOVX    @DPTR,A         ;WR TO 82530
        MOV     A,#0AH          ;DATA FOR REG C
        MOVX    @DPTR,A         ;WR TO 82530

        MOV     A,#0CH          ;SELECT REG C
        MOVX    @DPTR,A         ;WR TO 82530
        MOVX    A,@DPTR         ;RD 82530
        POP     DPH
        POP     DPL             ;RESTORE STACK
        CJNE    A,#0AH,INIT82_ERR
NIT_82530_LP1:
        CLR     A
        MOVX    A,@DPTR         ;READ INITIALIZATION CODE
        INC     DPTR
        PUSH    DPL             ;SAVE DPTR
        PUSH    DPH
        LCALL   CHK_CHAN
        MOVX    @DPTR,A         ;WRITE TO 82530
        POP     DPH             ;RESTORE STACK TO CORRECT POSITION
        POP     DPL
        DJNZ    COUNT82,INIT_82530_LP1
        INC     R7              ;CLEAR 3 BYTE FIFO
        LCALL   CHK_CHAN
        MOVX    A,@DPTR
```

```
        MOVX    A,@DPTR
        MOVX    A,@DPTR
        SJMP    INIT82_EXIT
IT82_ERR:
        LCALL   CLS1
        MOV     DPTR,#MSG1
        LCALL   CODE_TO_LCD
        MOV     DPTR,#LCD1_DAT_WR
        LCALL   LCD_WR
NIT82_EXIT:

        RET
;*************************************************************************

;*************************************************************************
;  THE SERIAL PORTS HAVE BEEN INITIALIZED FOR THE 2/3X16 INTELLIGENT     *
;  DISPLAYS. 9600,N,8,1                                                  *
;*************************************************************************

;*************************************************************************
;  INITIALIZATION FOR 82530A, REGISTER #, FOLLOWED BY DATA FOR THAT REG  *
;                                                                        *
;  NB!!!!  WHEN CHANGING WR5, PLEASE REFER TO NOTE5, @ START OF MODULE   *
;*************************************************************************
2530:
        DB      09,  83H        ;RESET CH A
        DB      03,  0C1H       ;RX MODE = 8 BITS
```

```
        DB          04,  44H        ;16 CLOCK, 1 STOP, NO PARITY
        DB          05,  6AH        ;TX MODE = 8 BITS, RTS 'L'
        DB          0AH, 00
        DB          0BH, 55H
        DB          0CH, 0AH        ;9600 BAUD
        DB          0DH, 00
        DB          0EH, 03
        DB          0FH, 00
        DB          01,  00         ;NO INTERRUPTS
;*************************************************************************

;*************************************************************************
; INITIALIZATION FOR 82530B, REGISTER #, FOLLOWED BY DATA FOR THAT REG    *
;                                                                         *
; NB!!!!  WHEN CHANGING WR5, PLEASE REFER TO NOTE5, @ START OF MODULE     *
;*************************************************************************
2530:
        DB          09,  4BH        ;RESET CH B
        DB          03,  0C1H       ;RX MODE = 8 BITS
        DB          04,  44H        ;16 CLOCK, 1 STOP, NO PARITY
        DB          05,  6AH        ;TX MODE = 8 BITS, RTS 'L'
        DB          0AH, 00
        DB          0BH, 55H
        DB          0CH, 0AH        ;9600 BAUD
        DB          0DH, 00
        DB          0EH, 03
        DB          0FH, 00
```

```
        DB          01,  00         ;NO INTERRUPTS
;*******************************************************************************

;*******************************************************************************
;* TRANSFER DATA FROM CODE AREA TO EXTERNAL DATA.                              *
;* FIRST SETUP SOURCE REGISTERS = DPTR                                         *
;*                                                                             *
;* ON ENTRY:                                                                   *
;*         DPTR POINTS TO SOURCE BUFFER                                        *
;*                                                                             *
;*         COUNT82 = # OF BYTES TO TRANSFER                                    *
;*                                                                             *
;*         R7 POINTS TO THE MSB OF THE DESTINATION BUFFER                      *
;*         R6 POINTS TO THE LSB OF THE DESTINATION BUFFER                      *
;*                                                                             *
;* ON EXIT:                                                                    *
;*         DESTINATION BUFFER WILL CONTAIN CODE DATA                           *
;*******************************************************************************
DE_TO_XDATA:
        CLR         A
        MOVC        A,@A+DPTR       ;READ SOURCE ADDR
        INC         DPTR
        PUSH        DPL             ;SAVE DPTR
        PUSH        DPH

        MOV         DPL,R6          ;RESTORE DESTINATION ADDR
        MOV         DPH,R7
```

```
        MOVX    @DPTR,A         ;WRITE TO DESTINATION BUFFER
        INC     DPTR
        MOV     R6,DPL          ;SAVE DPTR FOR DESTINATION ADDR
        MOV     R7,DPH

        POP     DPH             ;RESTORE DPTR TO SOURCE ADDR
        POP     DPL
        DJNZ    COUNT82,CODE_TO_XDATA   ;TRANSFER NEXT BYTE

        RET
;*****************************************************************************

G1:     DB      '* 82530 IS NOT * OPERATIONAL!!! ',0
G2:     DB      ' 82530 IS FULLY  OPERATIONAL!!! ',0
G3:     DB      'TX REG NOT EMPTYOR CTS NEVER LOW',0

NCLUDE (EQUATES.ASM)

        END
```

```
CRD.ASM
;*******************************************************************************
;  DATA SEGMENT                                                                 *
;*******************************************************************************
ATA_SEG     SEGMENT     DATA
            RSEG        DATA_SEG        ;RELOCATABLE INTERNAL DATA

RC:         DS          01              ;RESULT OF XOR OF ALL BYTES
RD_BUF:     DS          45              ;CARD BUFFER 45 BYTES
TACK:       DS          40              ;40 BYTES FOR THE STACK

;*******************************************************************************

;*******************************************************************************
;           CODE SEGMENT                                                        *
;*******************************************************************************
            CSEG        AT 0            ;ABSOLUTE SEGMENT FOR MAIN MODULE

            ORG         0
            USING       1               ;USING RB 0,1
            MOV         SP,#STACK
            LJMP        START

NCLUDE (EQUATES.ASM)
```

```
;*******************************************************************
;    MAIN ROUTINE                                                   *
;*******************************************************************
RT:
        LCALL       CRD_READ        ;SWIPE CARD
        LCALL       TRANS_DATA
        SJMP        START
;*******************************************************************

;*******************************************************************
; TRANSFER CARD DATA VIA A PORT PIN. WHEN DATA IS AVAILABLE, DATA    *
; LINE IS TAKEN 'L' TO INFORM HOST THAT DATA IS AVAILABLE. DATA IS THEN *
; CLOCKED OUT ON THE FALLING EDGE OF THE CLOCK. TRANSFER 1 BYTE FOR THE *
; STATUS AND 37 BYTES FOR CARD DATA                                  *
; 38 BYTE X 8 BITS = 304 CLOCK PULSES. LSB IS TRANSFERED FIRST.      *
; IF THERE'S AN ERROR IN READING THE CARD, THEN ONLY THE STATUS BYTE  *
; IS TRANSFERED TO THE HOST.                                         *
; P1.3 = SERIAL CARD DATA TO HOST.                                   *
; P1.4 = SERIAL DATA CLOCK FROM HOST.                                *
;                                                                    *
; 1 M/C CYCLE = 1.3 uSEC. ALLOW FOR 20 M/C CYCLES FOR THE BIT        *
; TRANSMISSION LOOP. ALLOW 100 uSEC FROM CLK 'H' TO CLK 'L' AND      *
; ALSO 100 uSEC FROM THE TIME THE BIT IS READ TILL THE NEXT CLOCK    *
; TRANSISSION                                                        *
;                                                                    *
; ON ENTRY:                                                          *
;       CRD_BUF CONTAINS THE DATA READ FROM THE MAGNETIC CARD        *
```

```
*************************************************************************

NS_DATA:
        SETB    P1.4            ;SET BIT AS AN INPUT FOR CLOCK
        MOV     R0,#CRD_BUF
        MOV     A,@R0           ;CHECK STATUS BYTE
        CLR     P1.3            ;INFORM HOST DATA AVAILABLE
        CJNE    A,#01,TRANS_LP3
        MOV     R3,#38          ;38 BYTES TO TRANSFER
        SJMP    TRANS_LP1
RANS_LP3:
        MOV     R3,#01          ;AN ERR IN READING CARD, TRANS 1 BYTE
RANS_LP1:
        MOV     A,@R0
        MOV     R2,#08          ;8 BITS
RANS_LP2:
        JNB     P1.4,$          ;WAIT FOR CLK 'H'
        JB      P1.4,$          ;WAIT FOR CLK 'L'
        RRC     A               ;SHIFT DATA INTO CARRY FLAG
        MOV     P1.3,C
        DJNZ    R2,TRANS_LP2    ;SERIALIZE DATA
        INC     R0
        DJNZ    R3,TRANS_LP1    ;NEXT BYTE TO TRANSFER
        SETB    P1.3

        RET
*************************************************************************
```

```
;*****************************************************************************
;     P1.0 = SWIPE DURATION                                                   *
;     P1.1 = CLOCK                                                            *
;     P1.2 = DATA                                                             *
;     CRD_FLAG = 1; VALID SWIPE                                               *
;     CARD BYTES MADE OF 5 BITS, 40 BYTES TOTAL. THIS INCLUDES 'START AND     *
;     END ' SENTINEL AND THE LRC (LONGITUDANAL CHARACTER CHECK) AND 37        *
;     DATA BYTES.                                                             *
;     STATUS BYTE: GOOD SWIPE = 01, BAD SWIPE = 0FFH                          *
;                                                                             *
;     ONLY STORE THE 37 DATA BYTES IN CRD_BUF, NOT START & END SENTINEL       *
;     AND LRC. THE FIRST BYTE IN THE STRING IS THE STATUS BYTE.               *
;     SETB 6 SO AS TO READ ASCII CHARS > 30H                                  *
;*****************************************************************************
RD_READ:
        MOV         R0,#CRD_BUF+1   ;ADDR OF CRD BUFFER, ALLOW FOR STATUS
        MOV         R3,#38      ;READ 37 DATA BYTES, 1 BYTE END SENTINEL
        MOV         LRC,#0          ;XOR OF ALL BYTES
        JB          P1.0,$          ;WAIT FOR SWIPE 'L'
        MOV         A,#0
RD_LOOP9:
        JNB         P1.1,$          ;WAIT FOR CLK 'H'
        JB          P1.1,$          ;WAIT FOR CLK 'L'
        MOV         C,P1.2          ;READ DATA BIT
        CPL         C               ;INVERT INCOMING DATA
        RRC         A
        ANL         A,#0F8H         ;MASK OFF MS NIBBLEL
```

```
        CJNE    A,#58H,CRD_LOOP9    ;START SENTINEL
        MOV     LRC,#11             ;START SENTINEL
RD_LOOP1:
        MOV     R2,#5               ;BITS / BYTE
        MOV     A,#0FFH
        SETB    C                   ;INITIALIZE REGS
RD_LOOP2:
        JNB     P1.1,$              ;WAIT FOR CLK 'H'
        JB      P1.1,$              ;WAIT FOR CLK 'L'
        MOV     C,P1.2              ;READ DATA BIT
        CPL     C                   ;INVERT INCOMING DATA
        RRC     A                   ;MOV DATA BIT INTO A
        DJNZ    R2,CRD_LOOP2        ;NEXT BIT
        CLR     C                   ;
        RRC     A
        CLR     C                   ;CLEAR BITS 6, 7 & 8
        RRC     A
        CLR     C
        RRC     A
        XRL     LRC,A               ;STORE RESULT OF XOR IN LRC
                                    ;START ODD PARITY CHECK
        MOV     R2,#8               ;CHECK PARITY / BYTE
        MOV     R4,#0               ;COUNT # OF '1'
RD_LOOP5:
        JNB     ACC.1,CRD_LOOP4         ;IF SET, INC R4
        INC     R4                  ;'1' FOUND
```

```
_LOOP4:
        RR      A               ;NEXT BIT TO TEST
        DJNZ    R2,CRD_LOOP5
        CJNE    R4,#1,CRD_LOOP6 ;ODD PARITY ONE '1'
        SJMP    CRD_LOOP8       ;READ NEXT BYTE
)_LOOP6:
        CJNE    R4,#3,CRD_LOOP7 ;THREE '1'
        SJMP    CRD_LOOP8       ;READ NEXT BYTE
)_LOOP7:
        CJNE    R4,#5,CRD_ERR   ;PARITY ERROR
                                ;END ODD PARITY CHECK
)_LOOP8:
        ORL     A,#30H          ;DISPALYABLE CHARS
        MOV     @R0,A
        JB      P1.0,CRD_ERR    ;IF SWIPE GOES 'H'

        ANL     A,#3FH          ;REQUIRE LS NIBBLE
        CJNE    A,#'?',CRD_LOOP3    ;END SENTINEL
                                ;*** READ LRC BYTE
        MOV     R2,#5           ;BITS / BYTE
        MOV     A,#0FFH
        SETB    C               ;INITIALIZE REGS
)_LOOP10:
        JNB     P1.1,$          ;WAIT FOR CLK 'H'
        JB      P1.1,$          ;WAIT FOR CLK 'L'
        MOV     C,P1.2          ;READ DATA BIT
        CPL     C               ;INVERT INCOMING DATA
```

```
        RRC     A               ;MOV DATA BIT INTO A
        DJNZ    R2,CRD_LOOP10   ;NEXT BIT
        CLR     C               ;
        RRC     A
        CLR     C               ;CLEAR BITS 6, 7 & 8
        RRC     A
        CLR     C
        RRC     A
        ORL     A,#30H          ;DISPALYABLE CHARS
        MOV     @R0,A           ;SAVE LRC BYTE
        ORL     LRC,#30H
                                ;*** LRC CHECK
        ANL     LRC,#0FH        ;USE LS NIBBLE
        ANL     A,#0FH          ;LRC BYTE
        CJNE    A,LRC,CRD_ERR   ;LRC IS INVALID
        MOV     @R0,#EOT        ;FOR LCD STRING, END OF STRING
        MOV     R0,#CRD_BUF
        MOV     @R0,#01         ;STATUS BYTE 01 = GOOD, 0FFH = BAD
        SJMP    CRD_EXIT
_ERR:
        MOV     R0,#CRD_BUF
        MOV     @R0,#0FFH
        SJMP    CRD_EXIT
_LOOP3:
        INC     R0              ;NEXT BYTE IN CRD_BUFFER
        DJNZ    R3,CRD_LOOP1    ;NEXT BYTE
_EXIT:
```

```
        RET                             ;END CARD READING ROUTINE
;*****************************************************************************

        END
```

EQUATES.ASM

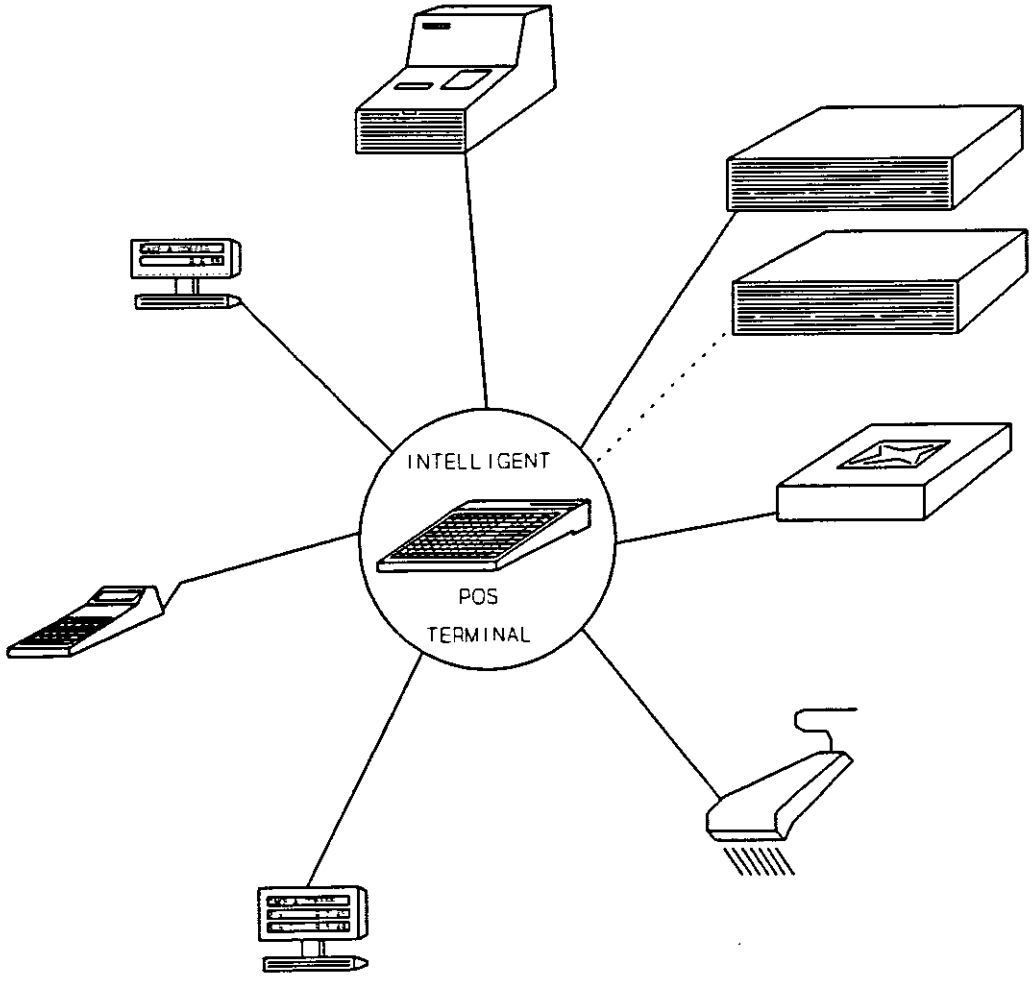
```
;*************************************************************************
; MEMORY I/O EQUATES                                                     *
;*************************************************************************

;*************************************************************************

1_CMD_WR    EQU         8000H       ;INSTRUCTION WRITE
1_STAT      EQU         8001H       ;READ LCD STATUS BIT, D7
1_DAT_WR    EQU         8002H       ;DATA BUFFER WRITE
1_DAT_RD    EQU         8003H       ;DATA BUFFER READ

2_CMD_WR    EQU         9000H       ;INSTRUCTION WRITE
2_STAT      EQU         9001H       ;READ LCD STATUS BIT, D7
2_DAT_WR    EQU         9002H       ;DATA BUFFER WRITE
2_DAT_RD    EQU         9003H       ;DATA BUFFER READ

K_JMP       EQU         0A000H      ;LOCK AND JUMPERS

_DAT        EQU         0B000H
_CMD        EQU         0B001H

_1          EQU         0C000H
_2          EQU         0D000H

ZER         EQU         0E000H
```

```
'SCB        EQU     0F000H      ;COMMS STATUS/CONTROL REG. CH B
!DATB       EQU     0F001H      ;DATA REG. CH B
!SCA        EQU     0F002H      ;COMMS STATUS/CONTROL REG. CH A
!DATA       EQU     0F003H      ;DATA REG. CH A

;*************************************************************************

;*************************************************************************
 CONSTANT DEFINITION EQUATES                                            *
;*************************************************************************
            EQU     09          ;SPACE INDICATOR, ^I
CE          EQU     20H         ;BLANK CHAR = ' '
            EQU     0
            EQU     13H         ;INDICATES LED STRING
            EQU     0           ;END OF STRING
N           EQU     87H         ;POWER CONTROL REGISTER
ON          EQU     0C8H        ;TIMER2 CONTROL REGISTER
P2H         EQU     0CBH        ;TIMER2 CAPTURE REG
P2L         EQU     0CAH

            EQU     70H         ;POLLING TERMINAL
            EQU     71H         ;SELECTING TERMINAL
            EQU     02          ;START OF TEXT
            EQU     03          ;END OF TEXT
            EQU     04          ;
            EQU     05          ;ENQUIRE
            EQU     06
```

```
          EQU       15H
          EQU       10H          ;USED TO PRECEDE CONTROL CODES
0         EQU       30H
1         EQU       31H

_DSP      EQU       01           ;CLEARS LCD DISPLAYS
E         EQU       02           ;CURSOR HOME
E         EQU       06           ;INC
_ON_OFF   EQU       0CH          ;DISPLAY ON/OFF
CTION     EQU       38H          ;8 BIT, 2 LINE, 7X5
E_1       EQU       080H         ;WRITE TO LINE #1
E_2       EQU       0C0H         ;WRITE TO LINE #2

*******************************************************************************
```

INTELLIGENT POINT OF SALE TERMINAL

SCHEMATIC DIAGRAMS



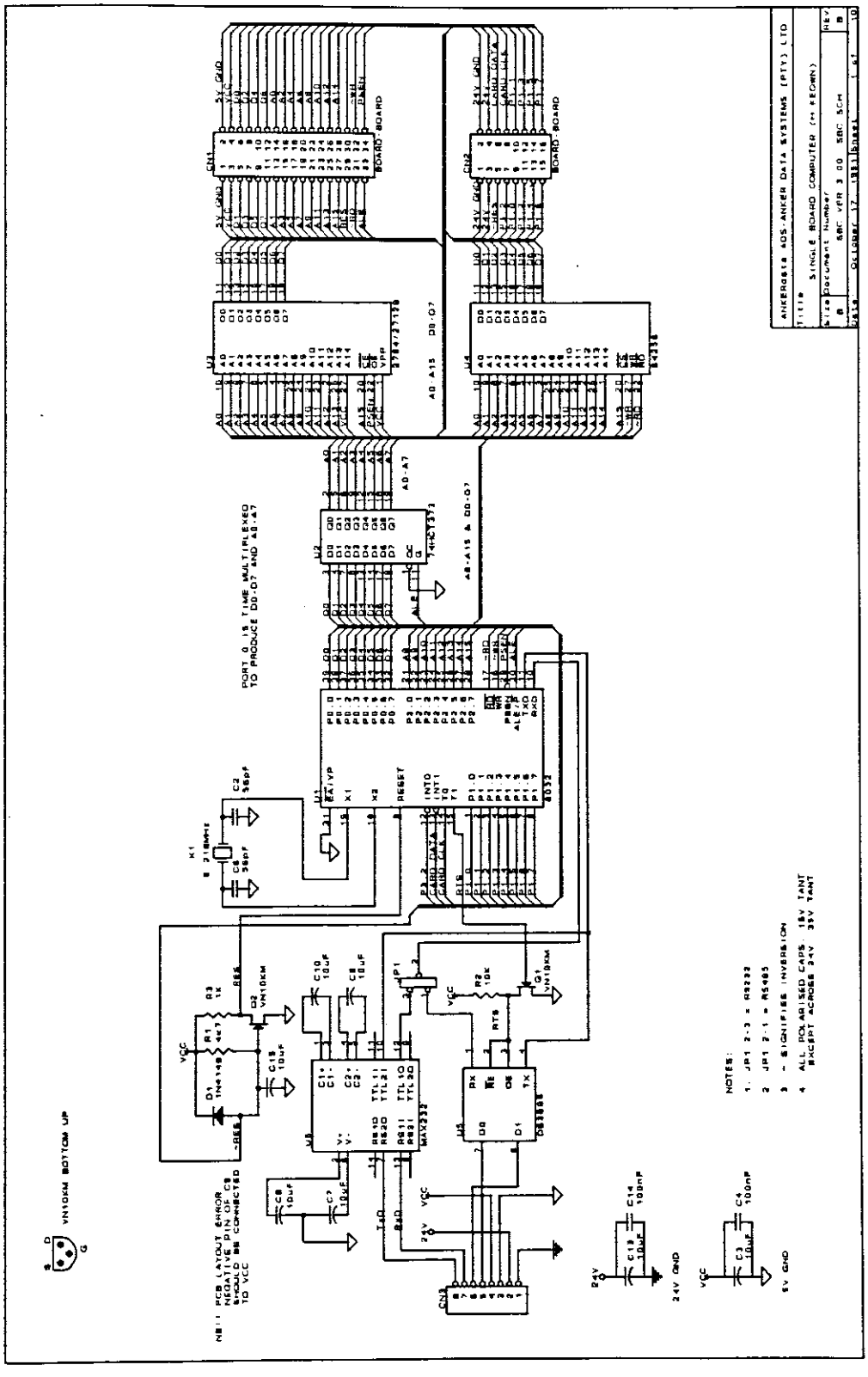


Figure 4-1. Single Board Computer

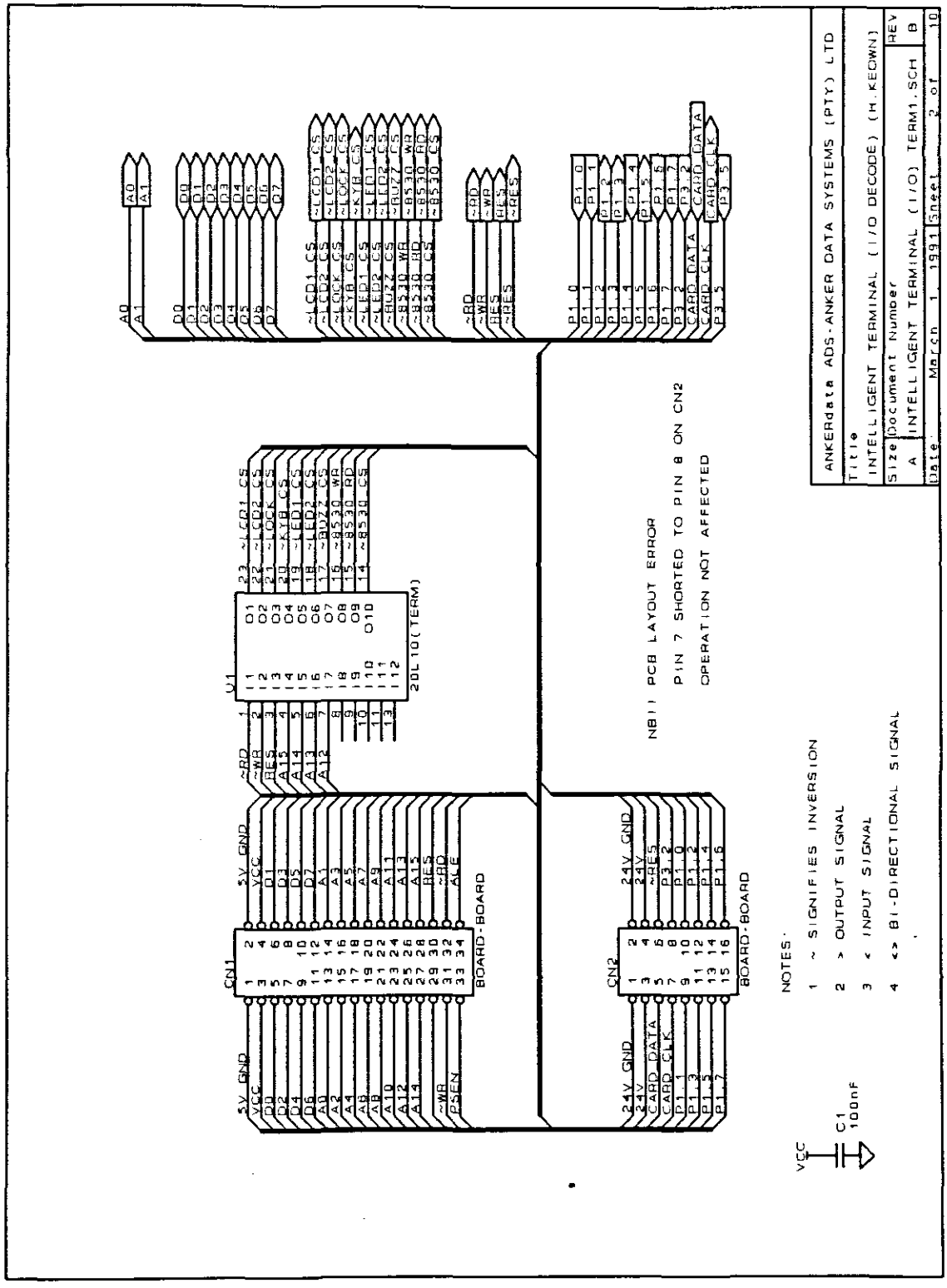


Figure 4-2. I/O Decode

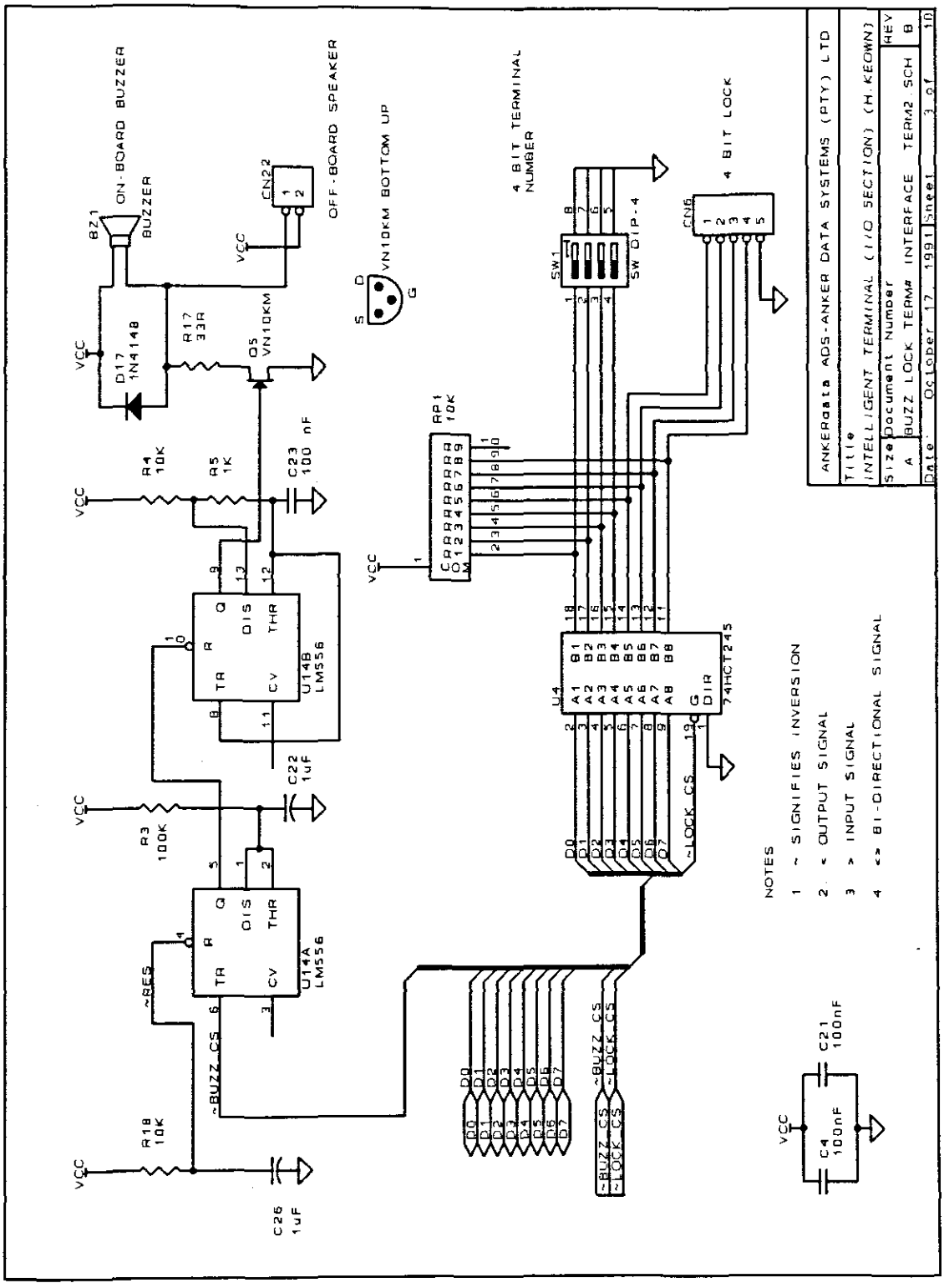


Figure 4-3. Buzzer & Lock Interface

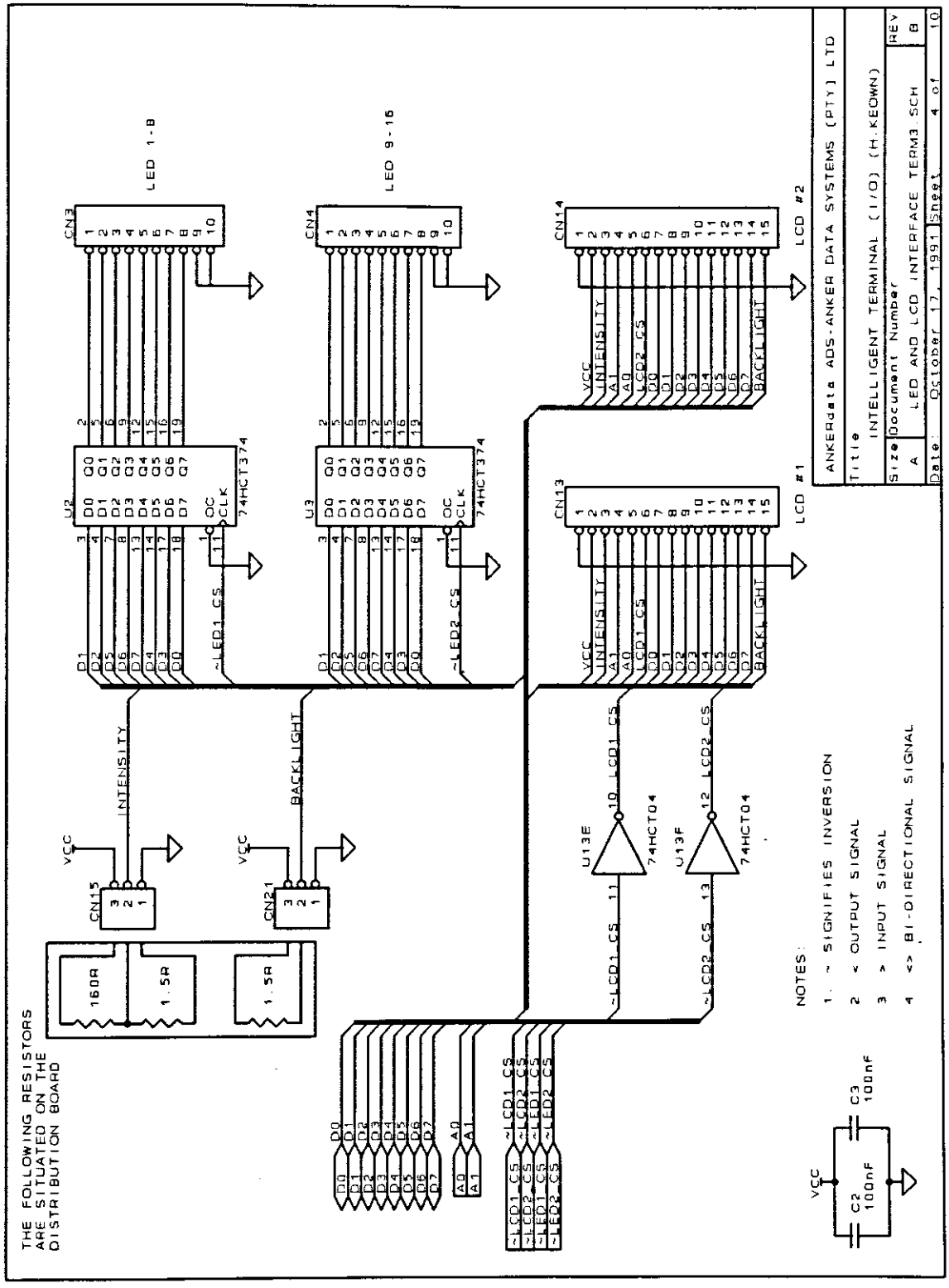


Figure 4-4. LED & LCD Interface

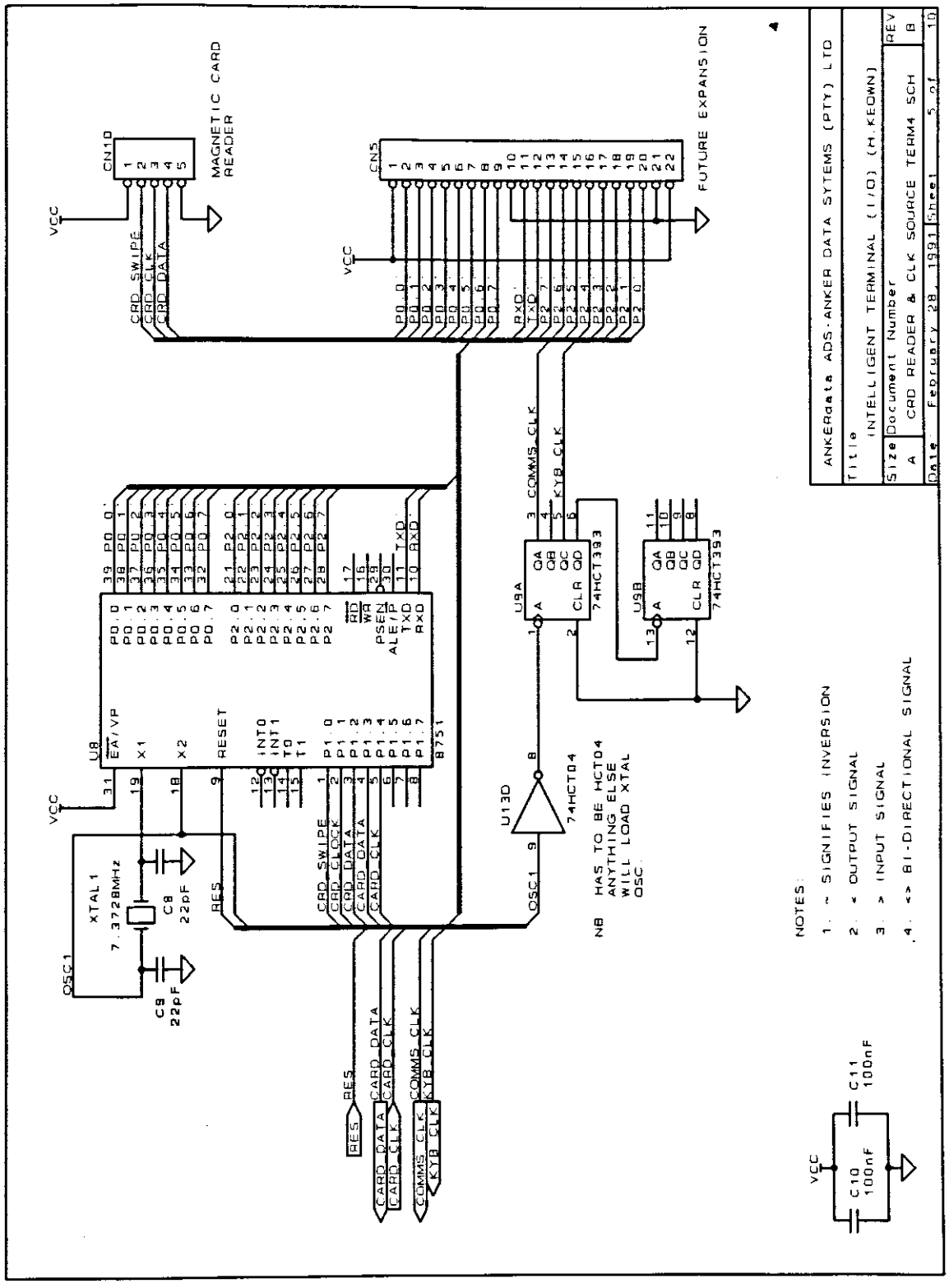


Figure 4-5. Card Reader Interface

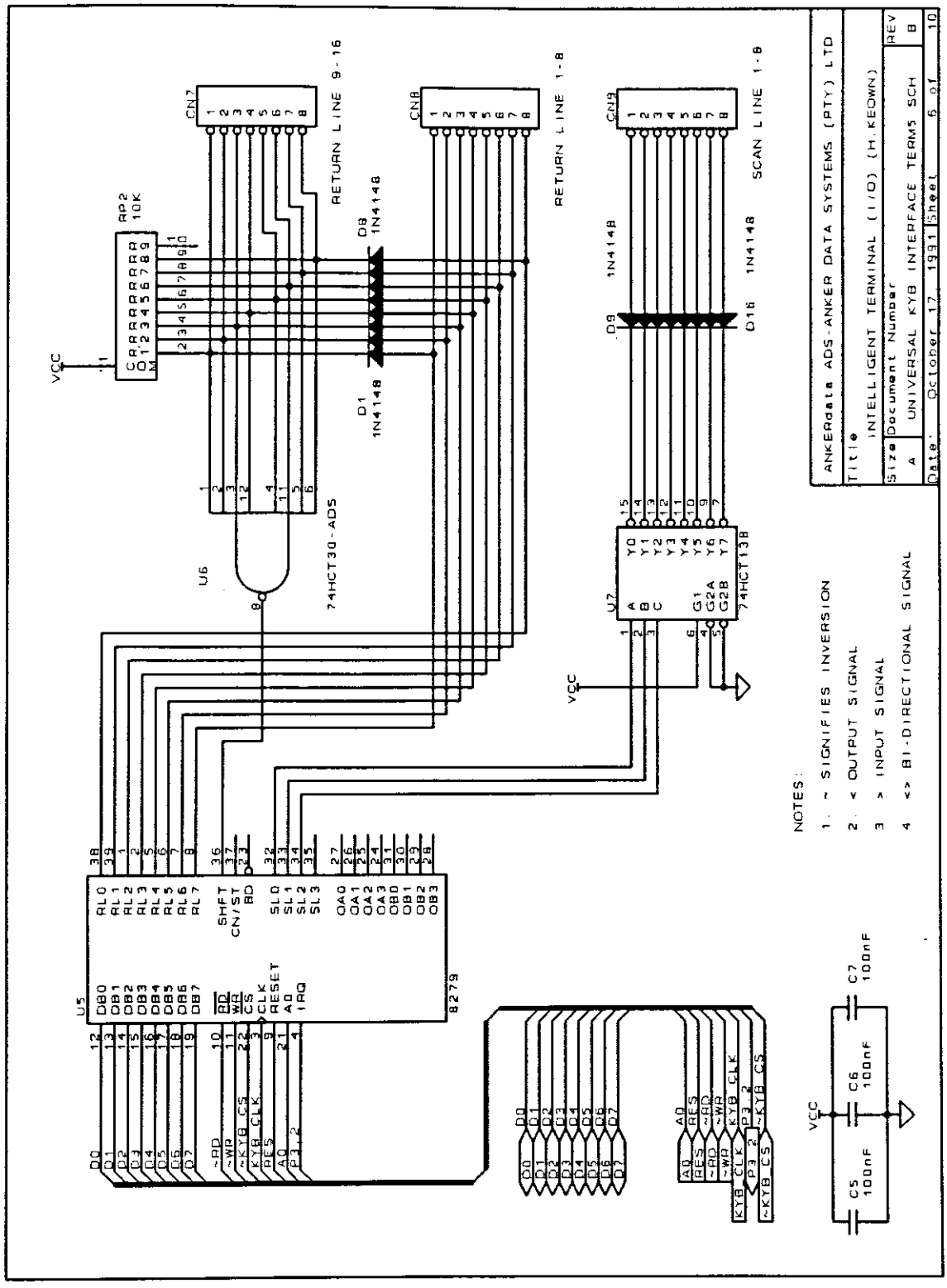


Figure 4-6. Keyboard Interface

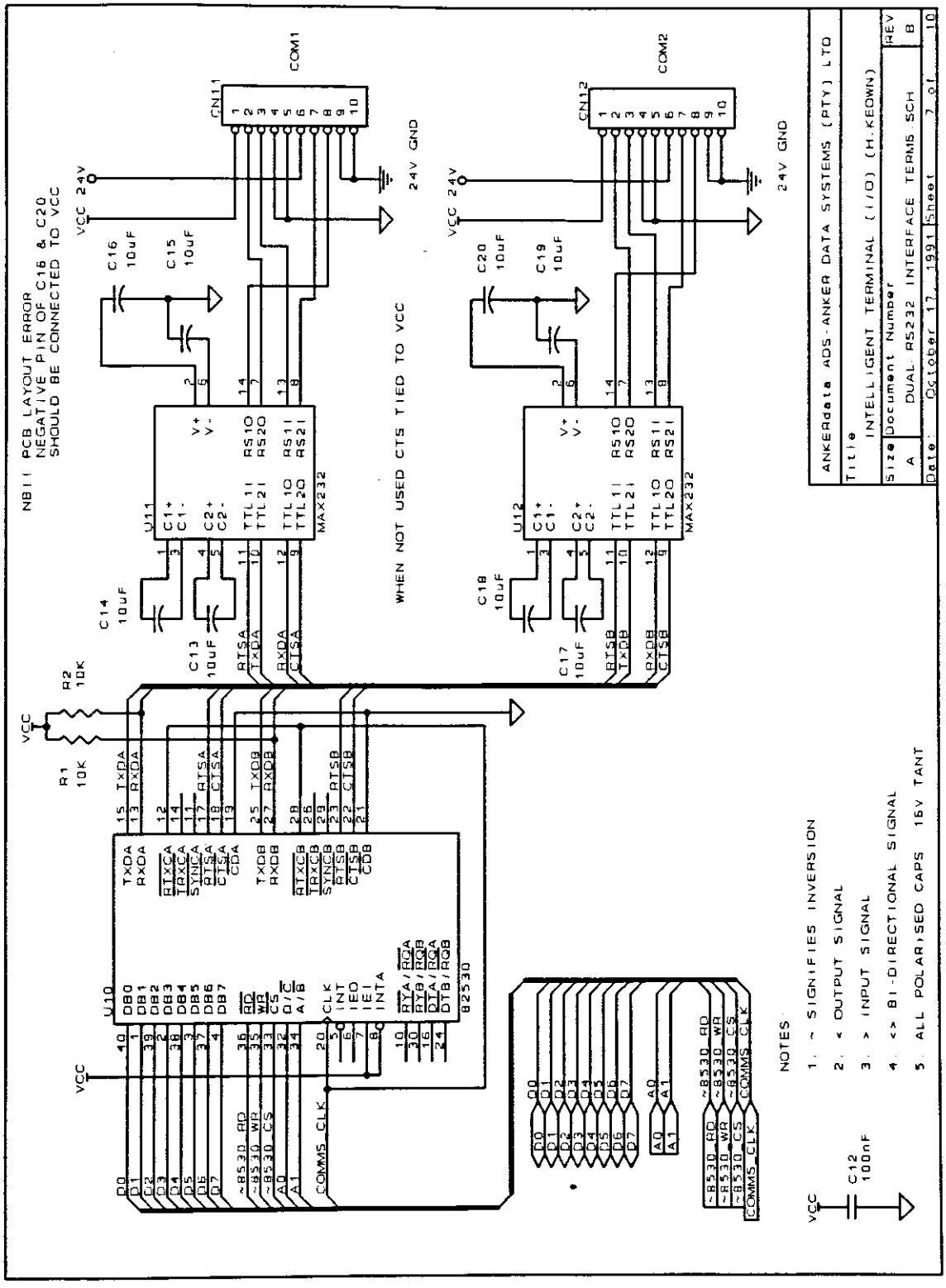


Figure 4-7. RS232 Interface

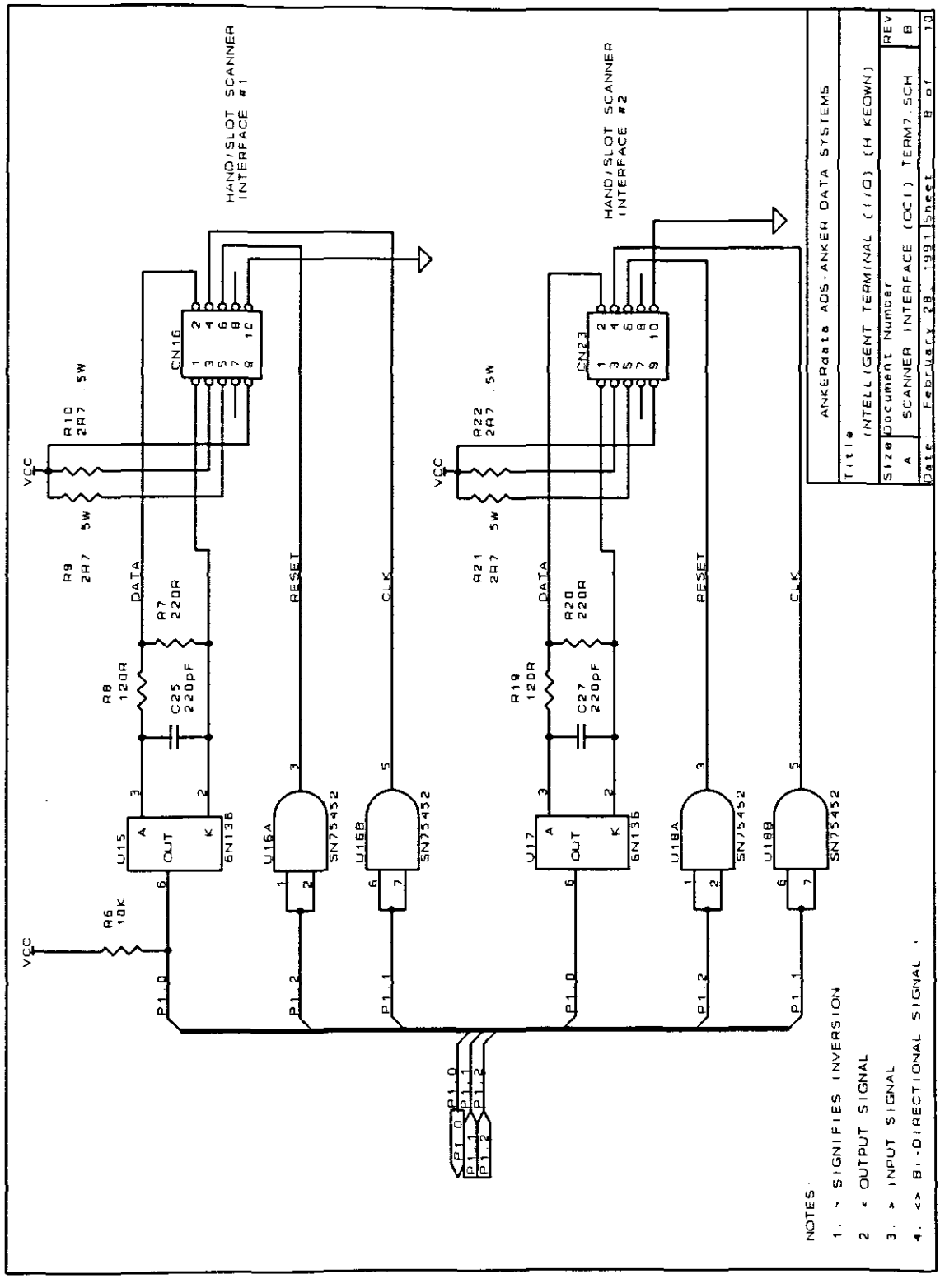


Figure 4-8. Scanner Interface

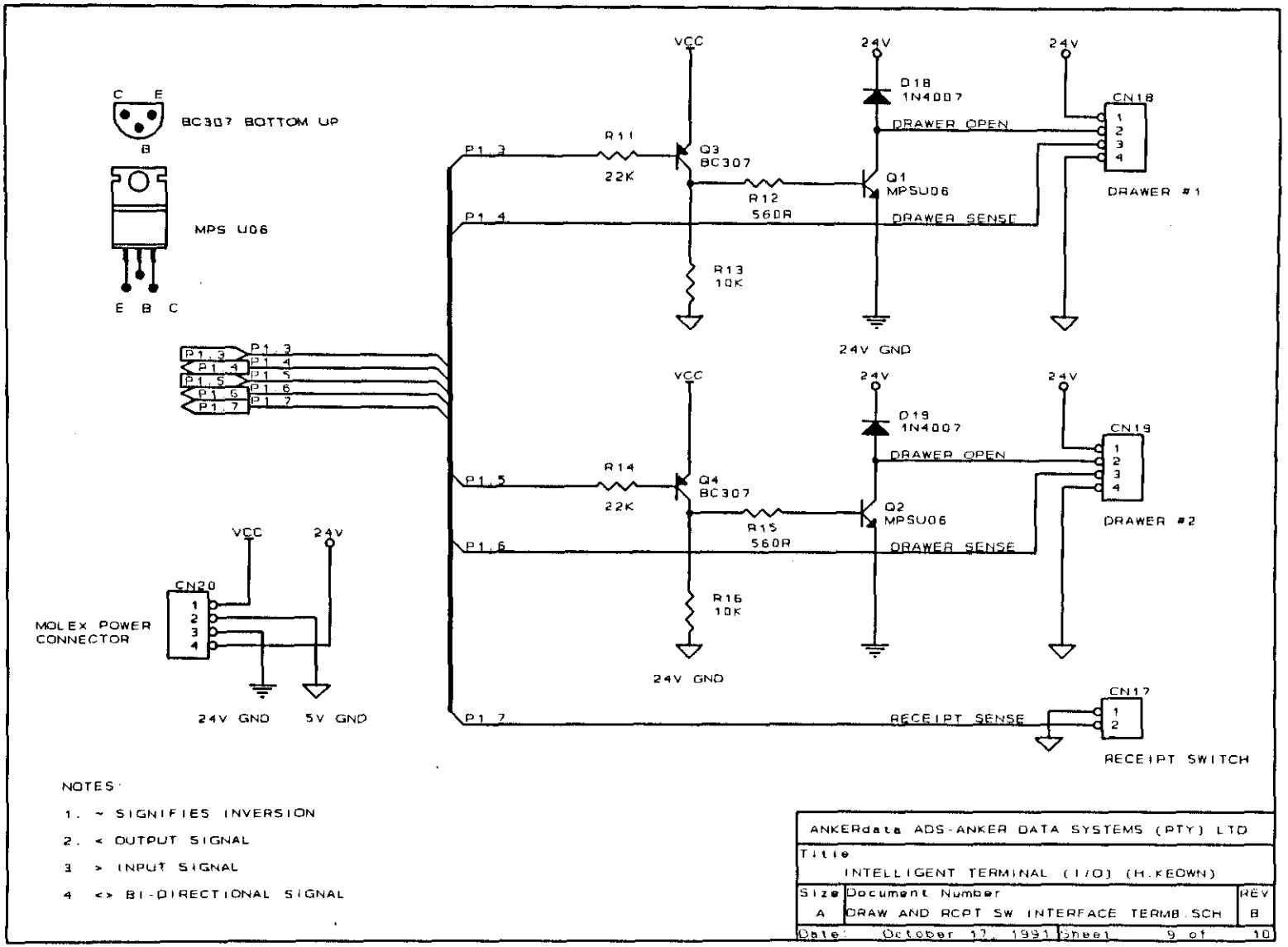


Figure 4-9. Draw & Rec. Interface

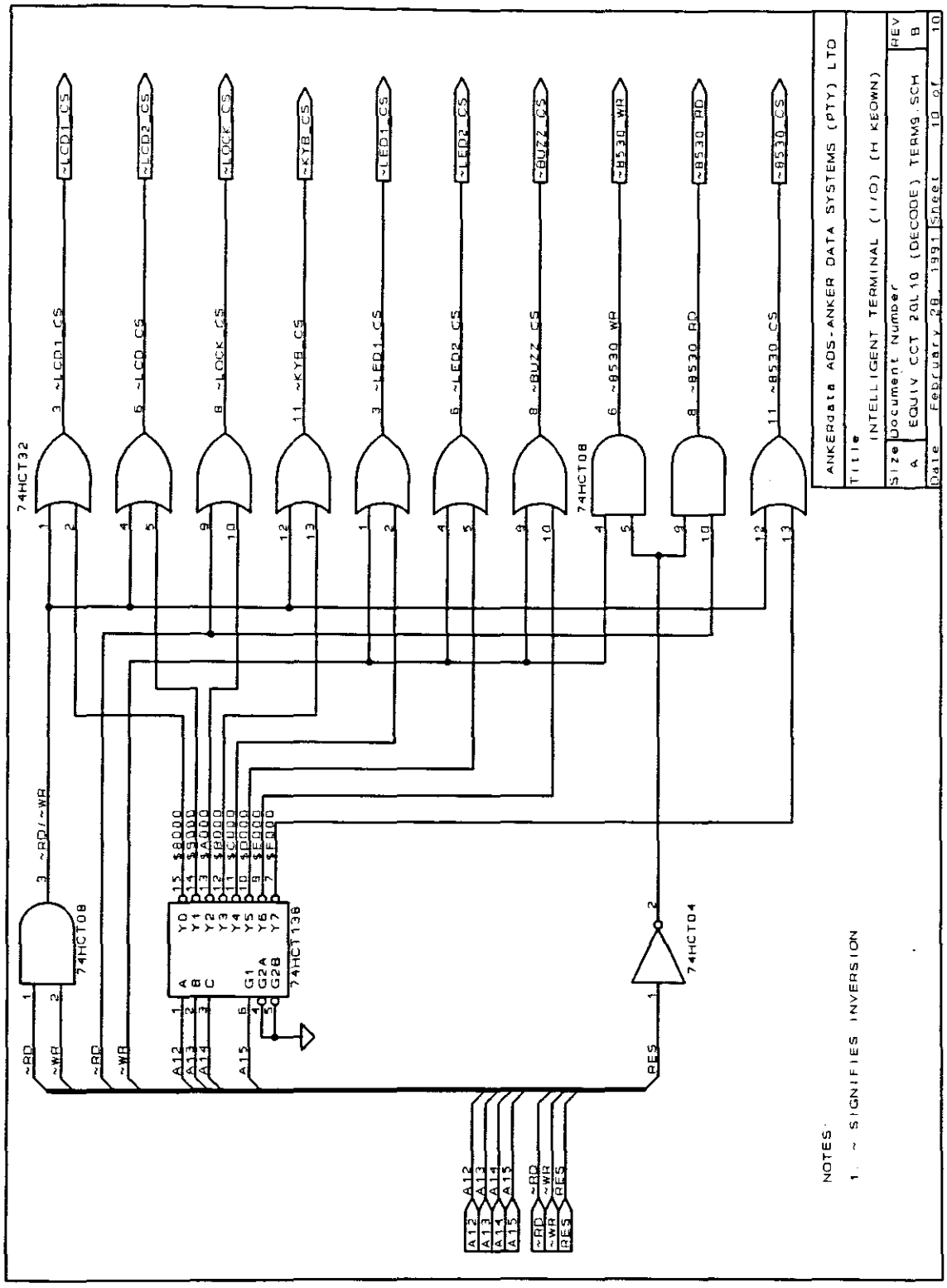


Figure 4-10. 20L10 Equiv. Circuit

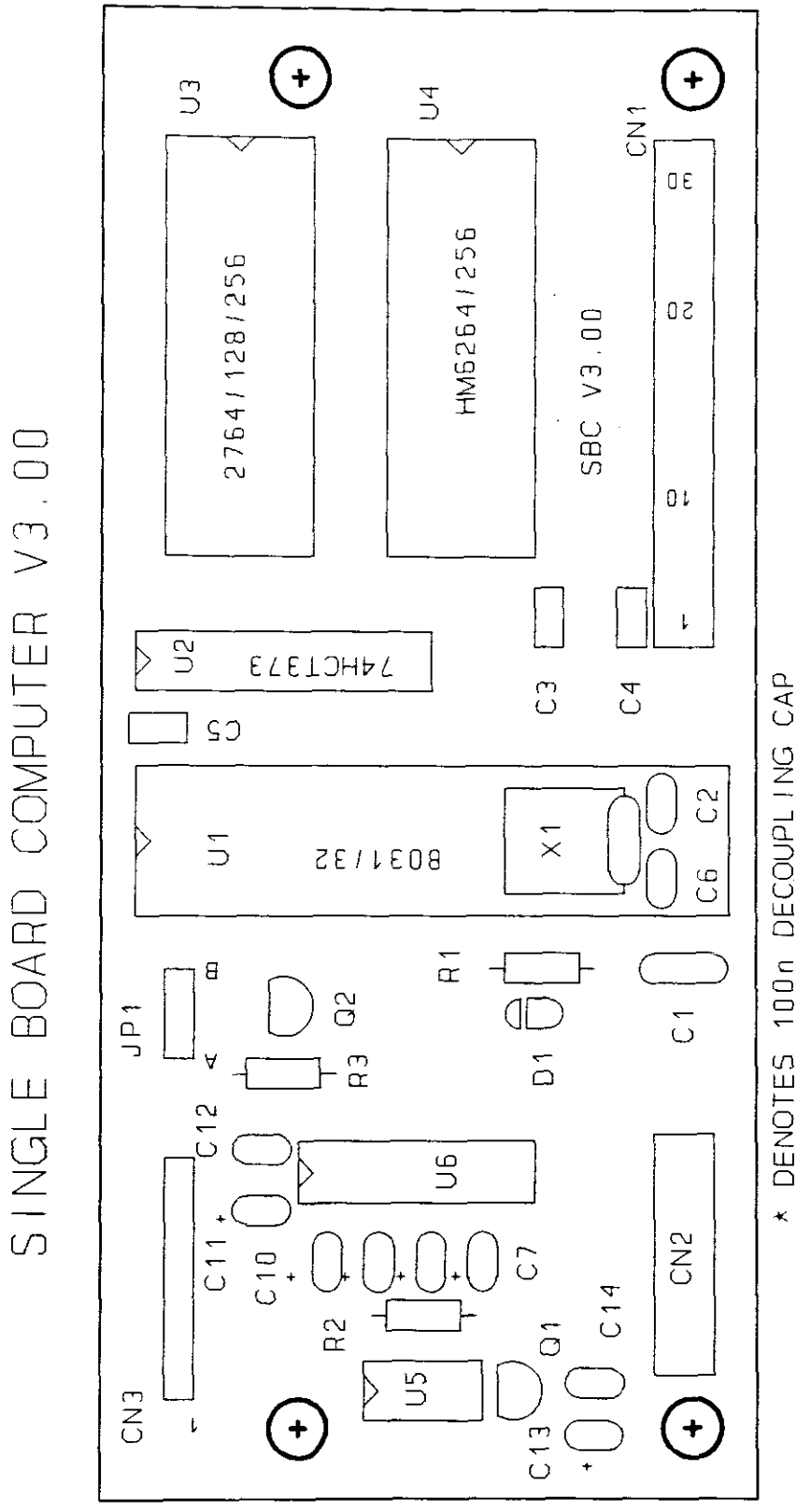


Figure 4-11. SBC Silkscreen

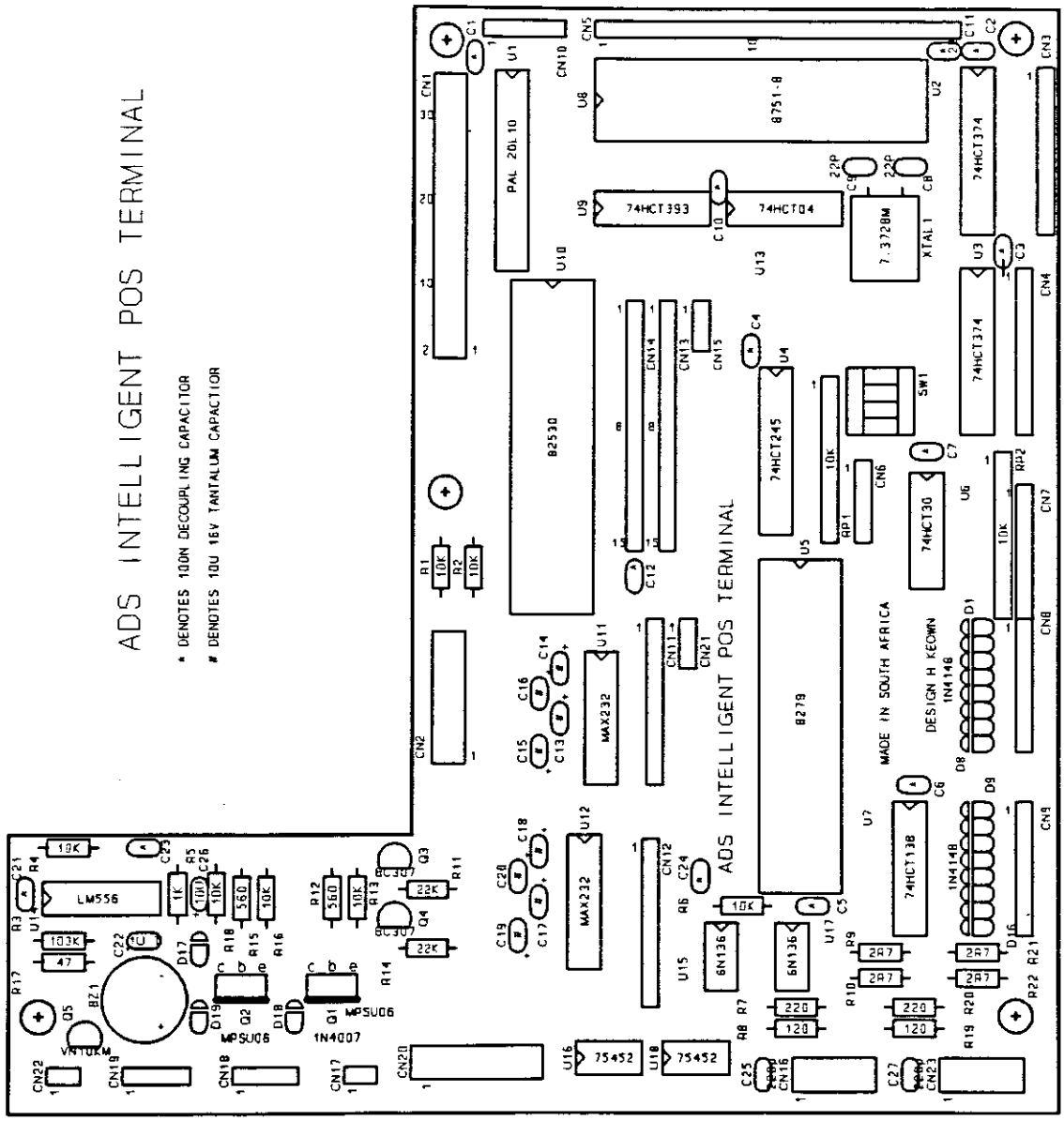
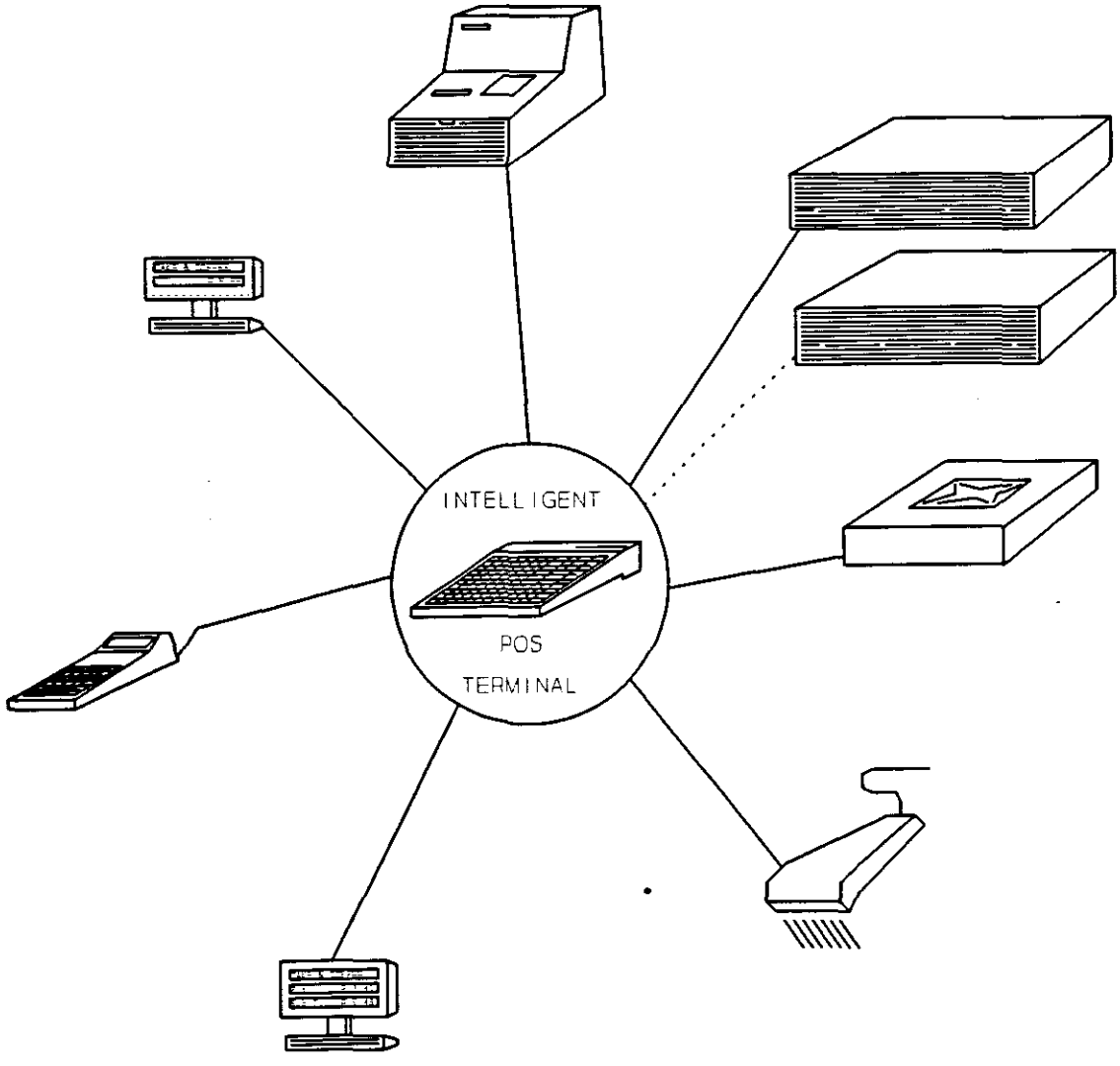


Figure 4-12. Terminal I/O Silkscreen

INTELLIGENT POINT OF SALE TERMINAL

PARTS LIST



CHAPTER 5

PARTS LIST

1. SINGLE BOARD COMPUTER

SBC VER 3.00 SBC.SCH Revision: B

Revised: October 17, 1991

Bill Of Materials October 17, 1991 19:22:29

Item	Quantity	Reference	Part
1	2	C2,C6	56pF
2	7	C3,C7,C8,C9,C10,C13,C15	10uF
3	2	C4,C14	100nF
4	2	CN1,CN2	BOARD-BOARD
5	1	CN3	8 HEADER
6	1	D1	1N4148
7	1	JP1	JP3
8	2	Q1,Q2	VN10KM
9	1	R1	4k7
10	1	R2	10K
11	1	R3	1K
12	1	U1	8032
13	1	U2	74HCT373
14	1	U3	2764/27128
15	1	U4	64256
16	1	U5	DS3695
17	1	U6	MAX232
18	1	X1	9.216MHz

2. INTELLIGENT TERMINAL (I/O)

2.1 INTELLIGENT TERMINAL (I/O) TERM1.SCH

Revised: March 1, 1991 Revision: B

Bill Of Materials October 17, 1991 18:12:53

Item	Quantity	Reference	Part
1	1	C1	100nF
2	2	CN1,CN2	BOARD-BOARD
3	1	U1	20L10(TERM)

2.2 BUZZ LOCK TERM# INTERFACE TERM2.SCH

Bill Of Materials October 17, 1991 18:34:22

Item	Quantity	Reference	Part
1	1	BZ1	BUZZER
2	2	C4,C21	100nF
3	2	C22,C26	1uF
4	1	C23	100 nF
5	1	CN6	HEADER 5
6	1	CN22	HEADER 2
7	1	D17	1N4148
8	1	Q5	VN10KM
9	1	R3	100K
10	3	RP1,R4,R18	10K
11	1	R5	1K
12	1	R17	33R
13	1	SW1	SW DIP-4
14	1	U4	74HCT245
15	1	U14	LM556

2.3 LED AND LCD INTERFACE TERM3.SCH

Bill Of Materials October 17, 1991 18:36:50

Item	Quantity	Reference	Part
1	2	C2,C3	100nF
2	2	CN3,CN4	HEADER 10
3	2	CN13,CN14	HEADER 15
4	2	CN15,CN21	HEADER 3
5	1	RX1	160R
6	2	RX2,RX3	1.5R
7	2	U2,U3	74HCT374
8	1	U13	74HCT04

2.4 CRD READER & CLK SOURCE TERM4.SCH

Bill Of Materials October 17, 1991 18:37:03

Item	Quantity	Reference	Part
1	2	C8,C9	22pF
2	2	C10,C11	100nF
3	1	CN5	HEADER 22
4	1	CN10	HEADER 5
5	1	U8	8751
6	1	U9	74HCT393
7	1	U13	74HCT04
8	1	XTAL1	7.3728MHz

2.5 UNIVERSAL KYB INTERFACE TERM5.SCH

Bill Of Materials October 17, 1991 18:55:45

Item	Quantity	Reference	Part
1	3	C5,C6,C7	100nF
2	3	CN7,CN8,CN9	HEADER 8
3	12	D1,D4,D5,D6,D7,D8,D10, D11,D12,D13,D14,D15	DIODE
4	4	D1,D8,D9,D16	1N4148
5	1	RP2	10K
6	1	U5	8279
7	1	U6	74HCT30-ADS
8	1	U7	74HCT138

2.6 DUAL RS232 INTERFACE TERM6.SCH

Bill Of Materials October 17, 1991 19:15:16

Item	Quantity	Reference	Part
1	1	C12	100nF
2	8	C13,C14,C15,C16,C17,C18, C19,C20	10uF
3	2	CN11,CN12	HEADER 10
4	2	R1,R2	10K
5	1	U10	82530
6	2	U11,U12	MAX232

2.7 SCANNER INTERFACE (OCI) TERM7.SCH

Bill Of Materials October 17, 1991 19:08:16

Item	Quantity	Reference	Part
1	2	C25,C27	220pF
2	2	CN16,CN23	HEADER 5X2
3	1	R6	10K
4	2	R7,R20	220R
5	2	R8,R19	120R
6	4	R9,R10,R21,R22	2R7 .5W
7	2	U15,U17	6N136
8	2	U16,U18	SN75452

2.8 DRAW AND RCPT SW INTERFACE TERM8.SCH

Bill Of Materials October 17, 1991 19:18:38

Item	Quantity	Reference	Part
1	1	CN17	HEADER 2
2	3	CN18,CN19,CN20	HEADER 4
3	2	D18,D19	1N4007
4	2	Q1,Q2	MPSU06
5	2	Q3,Q4	BC307
6	2	R11,R14	22K
7	2	R12,R15	560R
8	2	R13,R16	10K

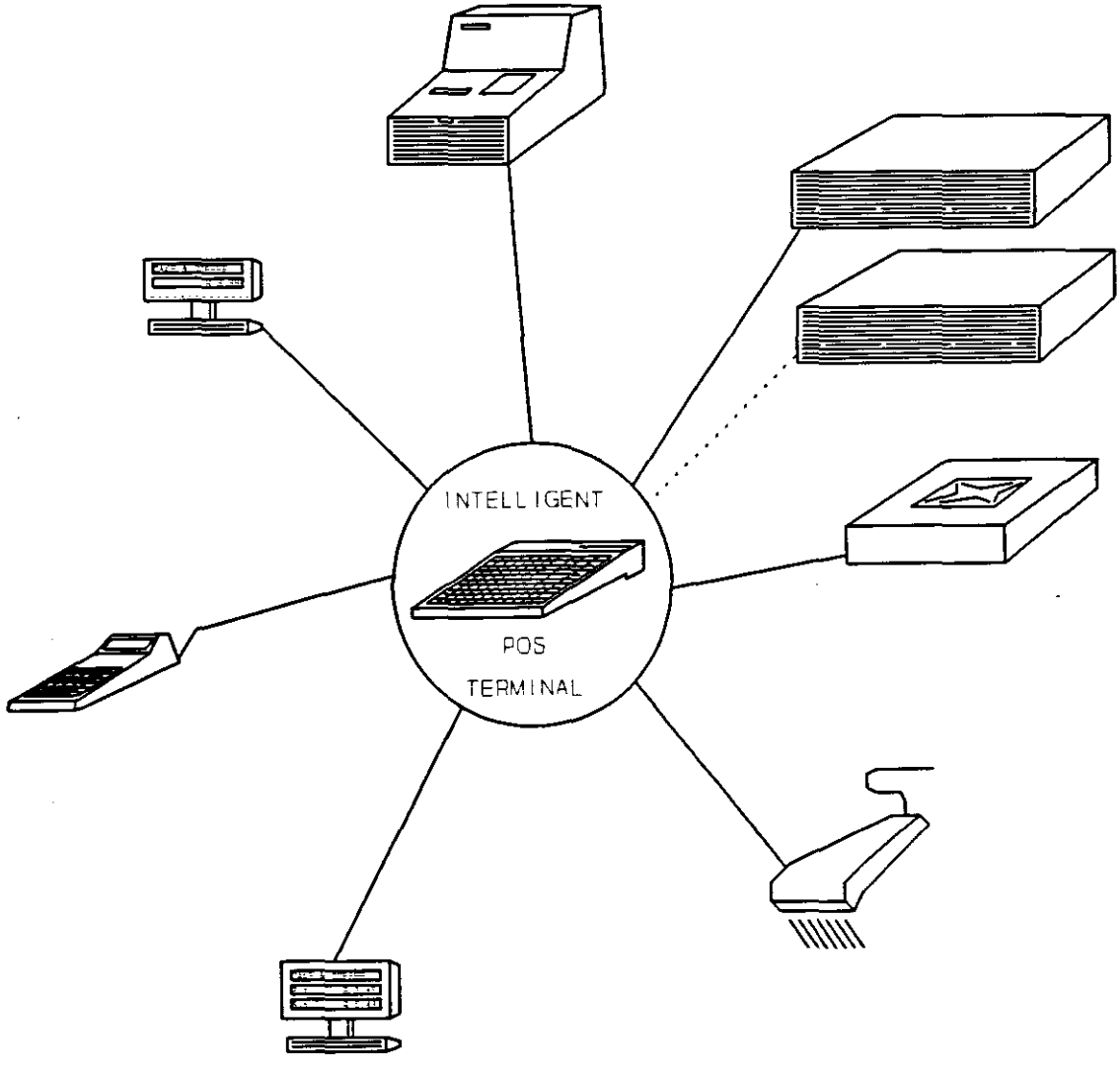
2.9 EQUIV CCT 20L10 (DECODE) TERM9.SCH

Bill Of Materials October 17, 1991 19:18:50

Item	Quantity	Reference	Part
1	1	U1	74HCT138
2	1	U2	74HCT08
3	2	U3,U4	74HCT32
4	1	U5	74HCT04

INTELLIGENT POINT OF SALE TERMINAL

BIBLIOGRAPHY



CHAPTER 6

BIBLIOGRAPHY

Advanced Micro Devices (AMD)

Programmable Logic

Boylestad & Nashelsky

Electronic Devices and Circuit Theory

Bourroughs

Series L Technical Manual

Dallas Semiconductor

Data Book

Floyd

Digital Fundamentals

Floyd

Electronic Devices

Intel

8-Bit Embedded Controllers 1989

Microsystems Components Handbook 1989

Microprocessors and Peripherals 1989

Microcommunications Handbook

MCS-51 Users Guide

MCS-52 Users Guide

Micro/C-51

IBM

XT Technical Reference Manual

IBM

AT Technical Reference Manual

Maxim

CMOS Data Acquisition Products 1988

Monolithic Memories

Large Scale Integration (LSI) Data Book

National Semiconductor

Linear Data Book

Logic Data Book

CMOS Data Book

Data Conversion, Interface and Bipolar LSI Data Book

Programmable Logic Data Book

Norton, P

Programmers Guide to the IBM PC

Optrex Corporation

Dot Matrix LCD Module - Users Guide

Scantech

MS7100 Symbol Scanner Publication

Sperry

Programming Considerations for Magnetic Stripe Readers

Siliconix

Small Signal FET Design

Tokyo Electronic Company (TEC)

PC I/F Transmission Control Procedures for the MA -1190

Zilog

Z8030 / Z8530 SCC Serial Communications Controller 1983

Linear Technology

1990 Linear Applications Handbook

Matra-Harris Semiconductors (MHS)

Digital CMOS/HMOS

Memories

SOFTWARE PACKAGES

Orcad SDT, VST, PLD

MSWORD 5.0, MSWORD 4.0

Wordperfect 5.1

Drawperfect 1.1

HP's Drawing Gallery

Intel's Macro Assembler and Linker, ASM51 and RL51

IAR C51

Archemedes C51

ABEL

PCAD