INTELLIGENT POINT OF SALE TERMINAL
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: [Signature] 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>
<thead>
<tr>
<th>Section</th>
<th>Pages</th>
</tr>
</thead>
<tbody>
<tr>
<td>SUMMARY</td>
<td>PAGE 12</td>
</tr>
<tr>
<td>INTRODUCTION</td>
<td>PAGE 16</td>
</tr>
<tr>
<td>Figure 0-1. INTELLIGENT TERMINAL</td>
<td>PAGE 13</td>
</tr>
<tr>
<td>Figure 0-2. Integrated Terminal - MTS2002</td>
<td>PAGE 18</td>
</tr>
<tr>
<td>Figure 0-3. MTS2002 Configuration</td>
<td>PAGE 19</td>
</tr>
<tr>
<td>Figure 0-4. POS Terminals and Back Office Computer</td>
<td>PAGE 24</td>
</tr>
</tbody>
</table>

## CHAPTER 1 | PAGE 2

1. COMMUNICATIONS INTRODUCTION | PAGE 2

2. TRANSMISSION SPECIFICATION | PAGE 4

3. TRANSMISSION CONTROL CODES | PAGE 5

4. TRANSMISSION FORMAT | PAGE 7
   4.1 TEXT FORMAT | PAGE 8

5. TRANSMISSION CONTROL PROCEDURES | PAGE 10
   5.1 CONNECTION PHASE (NO PROCESSING) | PAGE 10
   5.2 DATA LINK ESTABLISH PHASE | PAGE 10

INTELLIGENT POS TERMINAL | PAGE 1
# TABLE OF CONTENTS

5.2.1 DATA LINK ESTABLISH PHASE AT HOST  .  .  .  .  .  .  .  .  .  .  .  .  .  .  PAGE 11
  5.2.1.1 SELECTING  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  PAGE 11
  5.2.1.2 POLLING  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  PAGE 12

5.2.2 DATA LINK ESTABLISH PHASE AT TERMINAL  .  .  .  .  .  .  .  .  .  .  PAGE 13

5.3 TEXT TRANSFER PHASE  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  PAGE 15
  5.3.1 TEXT TRANSFER PHASE AT THE HOST  .  .  .  .  .  .  .  .  .  .  .  .  .  PAGE 15
  5.3.2 TEXT TRANSFER PHASE AT THE TERMINAL  .  .  .  .  .  .  .  .  .  .  .  .  PAGE 17

5.4 DATA LINK END PHASE  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  PAGE 19

5.5 DISCONNECTION PHASE  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  PAGE 19

6. ERROR CONTROL  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  PAGE 20

7. RETRY COUNTER FUNCTION  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  PAGE 21
  7.1 HOST RETRY COUNTER FUNCTION  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  PAGE 21
  7.2 TERMINAL RETRY COUNTER FUNCTION  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  PAGE 21

8. INTELLIGENT TERMINAL CONTROL CODES  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  PAGE 22
  8.1 OUTPUT DEVICE CONTROL CODES  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  PAGE 24
    8.1.1 16 X LED's  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  PAGE 24
    8.1.2 TWO X LIQUID CRYSTAL DISPLAYS  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  PAGE 26
    8.1.3 SOUND THE BUZZER  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  PAGE 27
    8.1.4 OPEN CASH DRAWER #1 OR #2  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  PAGE 28
    8.1.5 ENABLE OR DISABLE MAGNETIC CARD READER  .  .  .  .  .  .  .  .  .  PAGE 28
    8.1.6 HOST ENQUIRY  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  PAGE 29
    8.1.7 TWO RS232 SERIAL PORTS  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  PAGE 30

  8.2 INPUT DEVICE CONTROL CODES  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  PAGE 35
    8.2.1 4 BIT CONTROL LOCK  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  PAGE 35

INTELLIGENT POS TERMINAL  PAGE 2
TABLE OF CONTENTS

8.2.2 MAGNETIC CARD READER ................ PAGE 35
8.2.3 DRAWER STATUS ......................... PAGE 36
8.2.4 RECEIPT STATUS ......................... PAGE 37
8.2.5 KEYBOARD OR SCANNER DATA .............. PAGE 37
8.2.6 RESPONSE TO HOST ENQUIRY ............... PAGE 39
8.2.7 TWO X RS232 SERIAL PORTS ............... PAGE 39

9. PROGRAMMING EXAMPLE ..................... PAGE 41

Table 1-1. Transmission Specification ............. PAGE 4
Table 1-2. Transmission Control Codes ............. PAGE 6
Table 1-3. Keyboard Matrix ........................ PAGE 38

Figure 1-1. INTELLIGENT TERMINAL-Back Office Computer . PAGE 2

CHAPTER 2 ........................................ PAGE 2

1. SYSTEM DESCRIPTION .......................... PAGE 2
   1.1 SYSTEM OVERVIEW .......................... PAGE 2
   1.2 TERMINAL OVERVIEW ........................ PAGE 3
   1.3 DETAILED TERMINAL DESCRIPTION ............ PAGE 5
      1.3.1 PROCESSING SECTION .................... PAGE 5
      1.3.2 COMMUNICATIONS SECTION ............... PAGE 6
      1.3.3 DEVICE SECTION ........................ PAGE 7

2. INTELLIGENT TERMINAL SPECIFICATIONS .......... PAGE 9

INTELLIGENT POS TERMINAL PAGE 3
# Table of Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3. MICROCONTROLLER DESCRIPTION</td>
<td>11</td>
</tr>
<tr>
<td>3.1 DATA AND PROGRAM MEMORY ORGANISATION</td>
<td>11</td>
</tr>
<tr>
<td>3.2 INTERNAL DATA MEMORY ORGANISATION</td>
<td>14</td>
</tr>
<tr>
<td>4. DESIGN IMPLEMENTATION</td>
<td>16</td>
</tr>
<tr>
<td>4.1 PROCESSING SECTION</td>
<td>16</td>
</tr>
<tr>
<td>4.2 COMMUNICATIONS INTERFACE SECTION</td>
<td>18</td>
</tr>
<tr>
<td>4.3 PORT PIN ASSIGNMENT</td>
<td>19</td>
</tr>
<tr>
<td>5. INPUT / OUTPUT DECODE CIRCUITRY</td>
<td>22</td>
</tr>
<tr>
<td>6. ADDRESS MAP AND I/O MAP</td>
<td>25</td>
</tr>
<tr>
<td>6.1 MEMORY I/O EQUATES</td>
<td>25</td>
</tr>
<tr>
<td>6.2 MEMORY MAP</td>
<td>26</td>
</tr>
<tr>
<td>7. BUZZER DESCRIPTION</td>
<td>29</td>
</tr>
<tr>
<td>8. CONTROL LOCK AND TERMINAL NUMBER INTERFACE</td>
<td>31</td>
</tr>
<tr>
<td>9. LED INTERFACE</td>
<td>32</td>
</tr>
<tr>
<td>10. LIQUID CRYSTAL DISPLAY INTERFACE (LCD)</td>
<td>33</td>
</tr>
<tr>
<td>10.1 GENERAL DESCRIPTION</td>
<td>33</td>
</tr>
<tr>
<td>10.2 LCD SPECIFICATIONS</td>
<td>33</td>
</tr>
<tr>
<td>10.3 CONNECTION POSITIONS</td>
<td>35</td>
</tr>
<tr>
<td>10.4 PIN DESCRIPTION</td>
<td>36</td>
</tr>
<tr>
<td>10.5 EXPLANATION OF INTERNAL OPERATION</td>
<td>38</td>
</tr>
<tr>
<td>INTELLIGENT POS TERMINAL</td>
<td>4</td>
</tr>
<tr>
<td>Section</td>
<td>Page</td>
</tr>
<tr>
<td>------------------------------------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>10.5.1 REGISTER</td>
<td>38</td>
</tr>
<tr>
<td>10.5.2 BUSY FLAG (BF)</td>
<td>40</td>
</tr>
<tr>
<td>10.5.3 ADDRESS COUNTER (AC)</td>
<td>40</td>
</tr>
<tr>
<td>10.5.4 DISPLAY DATA RAM (DD RAM)</td>
<td>40</td>
</tr>
<tr>
<td>10.5.5 CHARACTER GENERATOR ROM (CG ROM)</td>
<td>41</td>
</tr>
<tr>
<td>10.5.6 CHARACTER GENERATOR RAM (CG RAM)</td>
<td>41</td>
</tr>
<tr>
<td>10.6 DETAILED EXPLANATION OF INSTRUCTIONS</td>
<td>41</td>
</tr>
<tr>
<td>10.6.1 CLEAR DISPLAY</td>
<td>41</td>
</tr>
<tr>
<td>10.6.2 RETURN HOME</td>
<td>42</td>
</tr>
<tr>
<td>10.6.3 ENTRY MODE SET</td>
<td>42</td>
</tr>
<tr>
<td>10.6.4 DISPLAY ON/OFF CONTROL</td>
<td>43</td>
</tr>
<tr>
<td>10.6.5 CURSOR OR DISPLAY SHIFT</td>
<td>44</td>
</tr>
<tr>
<td>10.6.6 FUNCTION SET</td>
<td>45</td>
</tr>
<tr>
<td>10.6.7 INTENSITY CONTROL</td>
<td>46</td>
</tr>
<tr>
<td>10.7 DESIGN IMPLEMENTATION</td>
<td>47</td>
</tr>
<tr>
<td>11. MAGNETIC CARD READER INTERFACE</td>
<td>48</td>
</tr>
<tr>
<td>11.1 DATA FORMAT</td>
<td>48</td>
</tr>
<tr>
<td>11.1.1 ABA-CODED DATA FORMAT</td>
<td>48</td>
</tr>
<tr>
<td>11.1.1.1 CODED CHARACTER SET</td>
<td>48</td>
</tr>
<tr>
<td>11.1.1.2 INFORMATION FORMAT</td>
<td>50</td>
</tr>
<tr>
<td>11.1.1.3 LONGITUDINAL REDUNDANCY CHECK</td>
<td>51</td>
</tr>
<tr>
<td>11.1.2 IATA-CODED DATA FORMAT</td>
<td>52</td>
</tr>
<tr>
<td>11.1.2.1 CODED CHARACTER SET</td>
<td>52</td>
</tr>
<tr>
<td>11.1.2.2 INFORMATION FORMAT</td>
<td>53</td>
</tr>
<tr>
<td>11.1.2.3 LONGITUDINAL REDUNDANCY CHECK</td>
<td>55</td>
</tr>
<tr>
<td>11.2 READER DESCRIPTION</td>
<td>55</td>
</tr>
</tbody>
</table>
# TABLE OF CONTENTS

16. RECEIPT SENSE ........................................ PAGE 80

17. PRESENT AND FUTURE DEVELOPMENT ............... PAGE 81

Table 2-1. Port Pin Assignment .......................... PAGE 21
Table 2-2. Memory I/O Equates .......................... PAGE 26
Table 2-3. Memory Map ..................................... PAGE 28
Table 2-4. Pin Connections ............................... PAGE 35
Table 2-5. Signal Description ............................ PAGE 37
Table 2-6. Register Selection ............................. PAGE 39
Table 2-7. ABA-Coded Character Set .................... PAGE 49
Table 2-8. IATA-Coded Character Set .................... PAGE 52
Table 2-9. IATA Information Format ..................... PAGE 54
Table 2-10. Reader Specification ........................ PAGE 56
Table 2-11. Reader Control Signals ..................... PAGE 58
Table 2-12. Byte Format .................................... PAGE 77

Figure 2-1. System Configuration ......................... PAGE 2
Figure 2-2. Terminal Description ........................ PAGE 4
Figure 2-3. Detailed Terminal Description .............. PAGE 5
Figure 2-4. INTELLIGENT TERMINAL Specifications ... PAGE 9
Figure 2-5. Memory Organisation ........................ PAGE 11
Figure 2-6. Internal Data Memory ........................ PAGE 13
Figure 2-7. Internal RAM Usage .......................... PAGE 15
Figure 2-8. Single Board Computer ....................... PAGE 16
Figure 2-9. Communication Interfaces .................... PAGE 19
Figure 2-10. Equivalent CCT for 20L10 PAL ............ PAGE 22

INTELLIGENT POS TERMINAL ........................ PAGE 7
| Figure 2-11. I/O Decoder 20L10 | PAGE 24 |
| Figure 2-12. 20L10 PAL | PAGE 25 |
| Figure 2-13. Buzzer Circuit | PAGE 29 |
| Figure 2-14. Lock & Terminal No. Interface | PAGE 31 |
| Figure 2-15. LED Interface | PAGE 32 |
| Figure 2-16. LCD Interface | PAGE 35 |
| Figure 2-17. LCD Block Diagram | PAGE 38 |
| Figure 2-18. Intensity Adjust | PAGE 46 |
| Figure 2-19. LCD Interface | PAGE 47 |
| Figure 2-20. Reader Block Diagram | PAGE 55 |
| Figure 2-21. Timing Diagram | PAGE 56 |
| Figure 2-22. Card Data Transfer | PAGE 59 |
| Figure 2-23. I/O Clock Generation | PAGE 60 |
| Figure 2-24. Expansion Bus | PAGE 61 |
| Figure 2-25. Interconnection between 8032 and 8751 | PAGE 62 |
| Figure 2-26. Data Transfer Timing Diagram | PAGE 62 |
| Figure 2-27. 8279 Block Diagram | PAGE 63 |
| Figure 2-28. Keyboard Interface | PAGE 67 |
| Figure 2-29. 8530 Block Diagram | PAGE 68 |
| Figure 2-30. Communications Interface | PAGE 71 |
| Figure 2-31. Scanner Connection | PAGE 74 |
| Figure 2-32. Byte Transfer From Scanner | PAGE 75 |
| Figure 2-33. Byte Transfer To Scanner | PAGE 75 |
| Figure 2-34. Scanner Interface | PAGE 78 |
| Figure 2-35. Drawer Interface | PAGE 79 |
| Figure 2-36. 2006 System (Metro) | PAGE 82 |
| Figure 2-37. 2007 System (Restaurant) | PAGE 83 |
TABLE OF CONTENTS

CHAPTER 3 .................................................. PAGE 2

ASSEMBLER CODE LISTING .................................. PAGE 2

1. KYB.ASM .................................................. PAGE 2

2. IO1.ASM .................................................. PAGE 35

3. IO2.ASM .................................................. PAGE 67

4. COMMS.ASM ............................................. PAGE 82

5. SCAN.ASM ............................................... PAGE 112

6. 82530.ASM ............................................. PAGE 122

7. CRD.ASM ................................................. PAGE 150

8. EQUATES.ASM ......................................... PAGE 158

CHAPTER 4 .................................................. PAGE 2

SCHEMATIC DIAGRAMS ....................................... PAGE 2

Figure 4-1. Single Board Computer ..................... PAGE 2

Figure 4-2. I/O Decode .................................... PAGE 3

Figure 4-3. Buzzer & Lock Interface .................... PAGE 4

INTELLIGENT POS TERMINAL .............................. PAGE 9
<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4-4</td>
<td>LED &amp; LCD Interface</td>
<td>5</td>
</tr>
<tr>
<td>4-5</td>
<td>Card Reader Interface</td>
<td>6</td>
</tr>
<tr>
<td>4-6</td>
<td>Keyboard Interface</td>
<td>7</td>
</tr>
<tr>
<td>4-7</td>
<td>RS232 Interface</td>
<td>8</td>
</tr>
<tr>
<td>4-8</td>
<td>Scanner Interface</td>
<td>9</td>
</tr>
<tr>
<td>4-9</td>
<td>Draw &amp; Rec. Interface</td>
<td>10</td>
</tr>
<tr>
<td>4-10</td>
<td>20L10 Equiv. Circuit</td>
<td>11</td>
</tr>
<tr>
<td>4-11</td>
<td>SBC Silkscreen</td>
<td>12</td>
</tr>
<tr>
<td>4-12</td>
<td>Terminal I/O Silkscreen</td>
<td>13</td>
</tr>
</tbody>
</table>

**CHAPTER 5**

PARTS LIST

1. SINGLE BOARD COMPUTER

2. INTELLIGENT TERMINAL (I/O)
   - 2.1 INTELLIGENT TERMINAL (I/O)
   - 2.2 BUZZ LOCK TERM# INTERFACE
   - 2.3 LED AND LCD INTERFACE
   - 2.4 CRD READER & CLK SOURCE
   - 2.5 UNIVERSAL KYB INTERFACE
   - 2.6 DUAL RS232 INTERFACE
   - 2.7 SCANNER INTERFACE
   - 2.8 DRAW AND RCPT SW INTERFACE
   - 2.9 EQUIV CCT 20L10 (DECODE)

INTELLIGENT POS TERMINAL PAGE 10
TABLE OF CONTENTS

CHAPTER 6 ........................................ PAGE 2

BIBLIOGRAPHY ..................................... PAGE 2

INTELLIGENT POS TERMINAL PAGE 11
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 referring 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.
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.
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 Director 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.
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.

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
INTRODUCTION

Intelligent Point Of Sale Terminal, 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.
INTRODUCTION

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.
INTRODUCTION

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

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

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

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.

<table>
<thead>
<tr>
<th>HEADER</th>
<th>DATA</th>
</tr>
</thead>
<tbody>
<tr>
<td>TEXT (50 bytes max)</td>
<td></td>
</tr>
</tbody>
</table>

The header field is composed of the control codes:

<table>
<thead>
<tr>
<th>BINARY DEVICE NUMBER</th>
<th>I/O</th>
<th>I/O DESIGNATOR</th>
</tr>
</thead>
<tbody>
<tr>
<td>HEADER (3 bytes)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**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:
Represented 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
(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.
SOFTWARE DESCRIPTION

(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.

<table>
<thead>
<tr>
<th>HEADER</th>
<th>DATA</th>
</tr>
</thead>
<tbody>
<tr>
<td>TEXT (50 bytes max)</td>
<td></td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>BINARY DEVICE NUMBER</th>
<th>I/O</th>
<th>I/O DESIGNATOR</th>
</tr>
</thead>
</table>

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).

CHAPTER 1  PAGE 22
The following are the declarations which are used for the programming examples:

```
void 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]; /* message buffer */
```
8.1 OUTPUT DEVICE CONTROL CODES

8.1.1 16 X LED's

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.

<table>
<thead>
<tr>
<th>LED #</th>
<th>LED ON/OFF</th>
<th>LED FLASH</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-4</td>
<td>5-8</td>
<td>9-12</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>5</td>
<td>6</td>
<td>7</td>
</tr>
</tbody>
</table>

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.
SOFTWARE DESCRIPTION

Programming example #1:

This will flash LED #1.

\texttt{\textbf{sprintf (msg, "OL10001000")}};
\texttt{\textbf{txmaster (msg, 1, 10);}}

Programming example #2:

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

\texttt{\textbf{sprintf (msg, "OL18001000")}};
\texttt{\textbf{txmaster (msg, 1, 10);}}
SOFTWARE DESCRIPTION

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

<table>
<thead>
<tr>
<th>X</th>
<th>'O'</th>
<th>'D'</th>
</tr>
</thead>
<tbody>
<tr>
<td>0/1/2</td>
<td>32 BYTE DISPLAY STRING</td>
<td></td>
</tr>
</tbody>
</table>

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 empty display string.
(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.
\[
\text{sprintf}(
\text{msg},
\text{"OD1**INTELLIGENT***POS TERMINAL"}\
\text{txmaster (msg, 1, 35);}\
\]

Programming example #2
This will display a message on LCD #1 and LCD #2.
\[
\text{sprintf}(
\text{msg},
\text{"OD0**INTELLIGENT***POS TERMINAL"}\
\]
SOFTWARE DESCRIPTION

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.

<table>
<thead>
<tr>
<th>X</th>
<th>'O'</th>
<th>'B'</th>
</tr>
</thead>
</table>

CHAPTER 1  PAGE 27
SOFTWARE DESCRIPTION

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.
SOFTWARE DESCRIPTION

8.1.7 TWO RS232 SERIAL PORTS

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.
SOFTWARE DESCRIPTION

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

|   | '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:

- \( P = 'N' \rightarrow \text{no parity.} \)
- \( P = 'O' \rightarrow \text{odd parity.} \)
- \( P = 'E' \rightarrow \text{even parity.} \)

Programming example #1:
The following command will change the bit format to suit the M290 slip printer on COM1.

```c
sprintf(msg,"0S1%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.

```c
sprintf(msg,"0S1%cDTX FROM HOST TO TERM, TO COM1%c%c%c",34,Ox0d,ox0a,Ox11); /* 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.
SOFTWARE DESCRIPTION

```c
sprintf(msg,"OS2%cD%cREMOVE SLIP FROM FLAT 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.
SOFTWARE DESCRIPTION

8.2 INPUT DEVICE CONTROL CODES

8.2.1 4 BIT CONTROL LOCK

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

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

CHAPTER 1  PAGE 37
SOFTWARE DESCRIPTION

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

<table>
<thead>
<tr>
<th>01</th>
<th>02</th>
<th>03</th>
<th>04</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>11</td>
<td>12</td>
<td>13</td>
</tr>
<tr>
<td>19</td>
<td>20</td>
<td>21</td>
<td>22</td>
</tr>
<tr>
<td>28</td>
<td>29</td>
<td>30</td>
<td>31</td>
</tr>
<tr>
<td>37</td>
<td>38</td>
<td>39</td>
<td>40</td>
</tr>
<tr>
<td>46</td>
<td>47</td>
<td>48</td>
<td>49</td>
</tr>
<tr>
<td>55</td>
<td>56</td>
<td>57</td>
<td>58</td>
</tr>
<tr>
<td>64</td>
<td>65</td>
<td>66</td>
<td>67</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>05</th>
<th>06</th>
<th>07</th>
<th>08</th>
<th>09</th>
</tr>
</thead>
<tbody>
<tr>
<td>14</td>
<td>15</td>
<td>16</td>
<td>17</td>
<td>18</td>
</tr>
<tr>
<td>23</td>
<td>24</td>
<td>25</td>
<td>26</td>
<td>27</td>
</tr>
<tr>
<td>32</td>
<td>33</td>
<td>34</td>
<td>35</td>
<td>36</td>
</tr>
<tr>
<td>41</td>
<td>42</td>
<td>43</td>
<td>44</td>
<td>45</td>
</tr>
<tr>
<td>50</td>
<td>51</td>
<td>52</td>
<td>53</td>
<td>54</td>
</tr>
<tr>
<td>59</td>
<td>60</td>
<td>61</td>
<td>62</td>
<td>63</td>
</tr>
<tr>
<td>68</td>
<td>69</td>
<td>70</td>
<td>71</td>
<td>72</td>
</tr>
</tbody>
</table>

Table 1-3. Keyboard Matrix

SCANNER ENTRY FORMAT

<table>
<thead>
<tr>
<th>X</th>
<th>'I'</th>
<th>'K'</th>
<th>FFH</th>
<th>#</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>06H</td>
<td>13 BYTE NUMBER</td>
</tr>
</tbody>
</table>

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
SOFTWARE DESCRIPTION

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 */

#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\n");
  if (!initmcomm (1)) {

  

}
SOFTWARE DESCRIPTION

```
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 */
                break;
            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;
        }
    }
}
```
SOFTWARE DESCRIPTION

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,OxOd,OxOa,Ox11);
          txmaster (msg, 1, 37);

          sprintf(msg,"OS2%cD%COREMOVE SLIP FROM FLAT_BED PRINTER2%c",3B,OX02,Ox03);
          txmaster (msg, 1, 41);
          break;

CHAPTER 1  PAGE 43
SOFTWARE DESCRIPTION

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%E",6,0x96,0x07,0x01);
    /* 9600, 7, 1, E M290 slip printer */
    txmaster (msg, 1, 9);
    break;

}
SOFTWARE DESCRIPTION

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) {
                    fprintf(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);
                }
        }
    }
}
CHAPTER 1

SOFTWARE DESCRIPTION

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

case 'P':
    printf("RECEIPT ");
    if (mod[2] == 0x0)
        printf("OFF");
    else if (mod[2] == 0x1)
        printf("ON");
    break;

case 'R':
    printf("DRAW %c",mod[2]);
    if (mod[3] == 0x0)
        printf("CLOSED");
    else if (mod[3] == 0x1)
        printf("OPEN");
    break;

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

CHAPTER 1  PAGE 46
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
1. SYSTEM DESCRIPTION

1.1 SYSTEM OVERVIEW

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 Bouroughs 7 bit ASCII protocol, which has a POLL - ACKNOWLEDGE structure. Each terminal has a
HARDWARE DESCRIPTION

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
HARDWARE DESCRIPTION

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

128 KEY KEYBOARD
BI - DIRECTIONAL
MAGNETIC CARD READER
16 STATUS LED'S
4 BIT CONTROL LOCK
2 STANDARD (RS232) SERIAL
COMMUNICATION PORTS
HANDSCANNER INTERFACE
DESKTOP SCANNER INTERFACE
BUZZER
2 CASH DRAWERS
2 LCD'S
RS485 COMMUNICATIONS PORT
SELECTABLE DEVICE ADDRESS

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
HARDWARE DESCRIPTION

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 R̅D and W̅R signals are used to address the external data memory.
HARDWARE DESCRIPTION

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 OFH. 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

<table>
<thead>
<tr>
<th>PORT</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>P0</td>
<td>LOWER ADDRESS BUS AND DATA BUS</td>
</tr>
<tr>
<td>P1</td>
<td>GENERAL PURPOSE I/O</td>
</tr>
<tr>
<td>PORT</td>
<td>DESCRIPTION</td>
</tr>
<tr>
<td>------</td>
<td>-----------------------------</td>
</tr>
<tr>
<td>P1.0</td>
<td>SCANNER DATA</td>
</tr>
<tr>
<td>P1.1</td>
<td>SCANNER CLOCK</td>
</tr>
<tr>
<td>P1.2</td>
<td>SCANNER RESET</td>
</tr>
<tr>
<td>P1.3</td>
<td>DRAWER #1 OPEN</td>
</tr>
<tr>
<td>P1.4</td>
<td>DRAWER #1 OPEN / CLOSE SENSE</td>
</tr>
<tr>
<td>P1.5</td>
<td>DRAWER #2 OPEN</td>
</tr>
<tr>
<td>P1.6</td>
<td>DRAWER #2 OPEN / CLOSE SENSE</td>
</tr>
<tr>
<td>P1.7</td>
<td>RECEIPT SWITCH SENSE</td>
</tr>
<tr>
<td>P2</td>
<td>UPPER ADDRESS BUS</td>
</tr>
<tr>
<td>P3</td>
<td>SPECIFIC AND GENERAL PURPOSE</td>
</tr>
<tr>
<td>P3.0</td>
<td>RX DATA</td>
</tr>
<tr>
<td>P3.1</td>
<td>TX DATA</td>
</tr>
<tr>
<td>PORT</td>
<td>DESCRIPTION</td>
</tr>
<tr>
<td>------</td>
<td>-------------</td>
</tr>
<tr>
<td>P3.2</td>
<td>IRQ OF KEYBOARD CONTROLLER</td>
</tr>
<tr>
<td>P3.3</td>
<td>CARD DATA</td>
</tr>
<tr>
<td>P3.4</td>
<td>CARD CLOCK</td>
</tr>
<tr>
<td>P3.5</td>
<td>ENABLE FOR DS3695</td>
</tr>
<tr>
<td>P3.6</td>
<td>(^\text{WR})</td>
</tr>
<tr>
<td>P3.7</td>
<td>(^\text{RD})</td>
</tr>
</tbody>
</table>

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 \textdagger \textasciitilde RD or \textdagger \textasciitilde WR.

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

The LED's and the buzzer are only written to, therefore the \textdagger \textasciitilde 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 \textdagger \textasciitilde RD and \textdagger \textasciitilde WR pins of the 82530. To ensure a valid reset during power-up, the \textdagger \textasciitilde RD and \textdagger \textasciitilde 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 \textdagger \textasciitilde RD and \textdagger \textasciitilde WR signals for the 82530 communications chip are therefore logically AND'ed with the reset signal. This forces the \textdagger \textasciitilde 82530_RD and \textdagger \textasciitilde 82530_WR signals both 'L' at power-up, thus ensuring a valid hardware reset. The \textdagger \textasciitilde 82530_CS is generated at address 0F000H and is logically OR'ed with the \textdagger \textasciitilde RD / \textdagger \textasciitilde WR signals.
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.
HARDWARE DESCRIPTION

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
### Hardware Description

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Equates</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>KYB_DAT</td>
<td>EQU 0B000H</td>
<td></td>
</tr>
<tr>
<td>KYB_CMD</td>
<td>EQU 0B001H</td>
<td></td>
</tr>
<tr>
<td>LED_1</td>
<td>EQU 0C000H</td>
<td></td>
</tr>
<tr>
<td>LED_2</td>
<td>EQU 0D000H</td>
<td></td>
</tr>
<tr>
<td>BUZZER</td>
<td>EQU 0E000H</td>
<td></td>
</tr>
<tr>
<td>COMSCB</td>
<td>EQU 0F000H</td>
<td>;COMMS STATUS/CONTROL REG. CH B</td>
</tr>
<tr>
<td>COMDATB</td>
<td>EQU 0F001H</td>
<td>;DATA REG. CH B</td>
</tr>
<tr>
<td>COMSCA</td>
<td>EQU 0F002H</td>
<td>;COMMS STATUS/CONTROL REG. CH A</td>
</tr>
<tr>
<td>COMDATA</td>
<td>EQU 0F003H</td>
<td>;DATA REG. CH A</td>
</tr>
</tbody>
</table>

Table 2-2. Memory I/O Equates

### 6.2 Memory Map

<table>
<thead>
<tr>
<th>Address</th>
<th>Control</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0F003H</td>
<td>82530_CS</td>
<td>DATA REG. FOR CH A</td>
</tr>
<tr>
<td>0F002H</td>
<td>82530_CS</td>
<td>COMMAND REG. FOR CH A</td>
</tr>
<tr>
<td>0F001H</td>
<td>82530_CS</td>
<td>DATA REG. CH B</td>
</tr>
<tr>
<td>ADDRESS</td>
<td>CONTROL</td>
<td>DESCRIPTION</td>
</tr>
<tr>
<td>----------</td>
<td>----------</td>
<td>----------------------------------</td>
</tr>
<tr>
<td>0F000H</td>
<td>82530_CS</td>
<td>COMMAND REG. CH B</td>
</tr>
<tr>
<td>0E000H</td>
<td>BUZZ_CS</td>
<td>BUZZER</td>
</tr>
<tr>
<td>0D000H</td>
<td>LED2_CS</td>
<td>SECOND BANK OF LED's</td>
</tr>
<tr>
<td>0C000H</td>
<td>LED1_CS</td>
<td>FIRST BANK OF LED's</td>
</tr>
<tr>
<td>0B000H</td>
<td>KYB_CS</td>
<td>8279 DATA REGISTER</td>
</tr>
<tr>
<td>0B001H</td>
<td>KYB_CS</td>
<td>8279 COMMAND REGISTER</td>
</tr>
<tr>
<td>0A000H</td>
<td>LOCK_CS</td>
<td>CONTROL LOCK</td>
</tr>
<tr>
<td>9003H</td>
<td>LCD2_CS</td>
<td>READ LCD DATA BUFFER</td>
</tr>
<tr>
<td>9002H</td>
<td>LCD2_CS</td>
<td>WRITE LCD DATA BUFFER</td>
</tr>
<tr>
<td>9001H</td>
<td>LCD2_CS</td>
<td>LCD STATUS REGISTER</td>
</tr>
<tr>
<td>9000H</td>
<td>LCD2_CS</td>
<td>LCD INSTRUCTION REGISTER</td>
</tr>
<tr>
<td>ADDRESS</td>
<td>CONTROL</td>
<td>DESCRIPTION</td>
</tr>
<tr>
<td>-----------</td>
<td>-------------</td>
<td>----------------------------------</td>
</tr>
<tr>
<td>8003H</td>
<td>~LCD1_CS</td>
<td>READ LCD DATA BUFFER</td>
</tr>
<tr>
<td>8002H</td>
<td>~LCD1_CS</td>
<td>WRITE LCD DATA BUFFER</td>
</tr>
<tr>
<td>8001H</td>
<td>~LCD1_CS</td>
<td>LCD STATUS REGISTER</td>
</tr>
<tr>
<td>8000H</td>
<td>~LCD1_CS</td>
<td>LCD INSTRUCTION REGISTER</td>
</tr>
<tr>
<td>7FFFH</td>
<td>~A15</td>
<td>32K BYTES STATIC RAM</td>
</tr>
<tr>
<td>00000H</td>
<td></td>
<td>64256 SRAM</td>
</tr>
<tr>
<td>0FFH</td>
<td></td>
<td>256 BYTES OF INTERNAL RAM</td>
</tr>
<tr>
<td>00H</td>
<td></td>
<td>RAM</td>
</tr>
<tr>
<td>1FFFH</td>
<td>~A15</td>
<td>EXTERNAL ROM (8K X 8)</td>
</tr>
<tr>
<td>00000H</td>
<td></td>
<td>2764 EPROM</td>
</tr>
</tbody>
</table>

Table 2-3. Memory Map
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

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

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

Table 2-4. Pin Connections
### 10.4 PIN DESCRIPTION

<table>
<thead>
<tr>
<th>SIGNAL NAME</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>DB4 - DB7</td>
<td>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.</td>
</tr>
<tr>
<td>DB0 - DB3</td>
<td>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.</td>
</tr>
<tr>
<td>E</td>
<td>OPERATION START SIGNAL FOR DATA READ / WRITE.</td>
</tr>
<tr>
<td>R/\bar{W}</td>
<td>SIGNAL TO SELECT READ (R) OR WRITE (W). '0' : WRITE '1' : READ</td>
</tr>
</tbody>
</table>
### HARDWARE DESCRIPTION

<table>
<thead>
<tr>
<th>SIGNAL NAME</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>RS</strong></td>
<td>SIGNAL TO SELECT REGISTER.</td>
</tr>
<tr>
<td></td>
<td>'0': INSTRUCTION REGISTER (WRITE)</td>
</tr>
<tr>
<td></td>
<td>'0': BUSY FLAG; ADDRESS COUNTER (READ)</td>
</tr>
<tr>
<td></td>
<td>'1': DATA REGISTER (WRITE READ)</td>
</tr>
<tr>
<td><strong>Vee</strong></td>
<td>TERMINAL FOR LCD INTENSITY</td>
</tr>
<tr>
<td><strong>Vcc</strong></td>
<td>+5V</td>
</tr>
<tr>
<td><strong>Vss</strong></td>
<td>0V (GND)</td>
</tr>
</tbody>
</table>

Table 2-5. Signal Description
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.

<table>
<thead>
<tr>
<th>RS</th>
<th>R/W</th>
<th>OPERATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>IR WRITE, INTERNAL OPERATION (DISPLAY CLEAR etc)</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>BUSY FLAG (DB7) AND ADDRESS COUNTER. (DB0 - DB6) READ</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>DR WRITE, INTERNAL OPERATION (DR &gt; DD RAM OR CG RAM)</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>DR READ, INTERNAL OPERATION (DD RAM OR CG RAM)</td>
</tr>
</tbody>
</table>

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 \( RS = 0 \) and \( R/W = 1 \) 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

\[
\begin{array}{cccccccccc}
R\bar{S} & R/\bar{W} & D7 & D6 & D5 & D4 & D3 & D2 & D1 & D0 \\
\hline
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0
\end{array}
\]
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

\[
\begin{array}{cccccccccc}
  & RS & R/W & D7 & D6 & D5 & D4 & D3 & D2 & D1 & DO \\
 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & X \\
\end{array}
\]

\(X = \text{don't care}\)

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

\[
\begin{array}{cccccccccc}
  & RS & R/W & D7 & D6 & D5 & D4 & D3 & D2 & D1 & DO \\
 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & I/D & S \\
\end{array}
\]

\(I/D:\)

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

<table>
<thead>
<tr>
<th>RS</th>
<th>R/W</th>
<th>D7</th>
<th>D6</th>
<th>D5</th>
<th>D4</th>
<th>D3</th>
<th>D2</th>
<th>D1</th>
<th>D0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>D</td>
<td>C</td>
</tr>
</tbody>
</table>

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

\[
\begin{array}{cccccccc}
RS & R/W & D7 & D6 & D5 & D4 & D3 & D2 & D1 & D0 \\
0 & 0 & 0 & 0 & 0 & 1 & S/C & R/L & X & X \\
\end{array}
\]

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.
<table>
<thead>
<tr>
<th>S/C</th>
<th>R/L</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>SHIFTS CURSOR POSITION LEFT (AC DEC. BY 1)</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>SHIFTS CURSOR POSITION RIGHT (AC INC. BY 1)</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>SHIFTS THE ENTIRE DISPLAY TO THE LEFT.</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>SHIFTS THE ENTIRE DISPLAY TO THE RIGHT</td>
</tr>
</tbody>
</table>

X = don't care

### 10.6.6 FUNCTION SET

#### 1-LINE DISPLAY

<table>
<thead>
<tr>
<th>RS</th>
<th>R/W</th>
<th>D7</th>
<th>D6</th>
<th>D5</th>
<th>D4</th>
<th>D3</th>
<th>D2</th>
<th>D1</th>
<th>D0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>DL</td>
<td>0</td>
<td>F</td>
<td>X</td>
</tr>
</tbody>
</table>

X = don't care

#### 2-LINE DISPLAY

<table>
<thead>
<tr>
<th>RS</th>
<th>R/W</th>
<th>D7</th>
<th>D6</th>
<th>D5</th>
<th>D4</th>
<th>D3</th>
<th>D2</th>
<th>D1</th>
<th>D0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>DL</td>
<td>1</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

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

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

<table>
<thead>
<tr>
<th>BITS</th>
<th>ROW</th>
<th>CHARACTER</th>
</tr>
</thead>
<tbody>
<tr>
<td>P</td>
<td>B4</td>
<td>B3</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
HARDWARE DESCRIPTION

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

<table>
<thead>
<tr>
<th>B6</th>
<th>B5</th>
<th>B4 B3 B2 B1</th>
<th>COL</th>
<th>ROW</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0 1 1</td>
<td>0 1 0 1</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>B4 B3 B2 B1</th>
<th>ROW</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0 0 0</td>
<td>0</td>
</tr>
<tr>
<td>0 0 0 1</td>
<td>1</td>
</tr>
<tr>
<td>0 0 1 0</td>
<td>2</td>
</tr>
<tr>
<td>0 0 1 1</td>
<td>3</td>
</tr>
<tr>
<td>0 1 0 0</td>
<td>4</td>
</tr>
<tr>
<td>0 1 0 1</td>
<td>5</td>
</tr>
<tr>
<td>0 1 1 0</td>
<td>6</td>
</tr>
<tr>
<td>0 1 1 1</td>
<td>7</td>
</tr>
<tr>
<td>1 0 0 0</td>
<td>8</td>
</tr>
<tr>
<td>1 0 0 1</td>
<td>9</td>
</tr>
<tr>
<td>1 0 1 0</td>
<td>10</td>
</tr>
<tr>
<td>1 0 1 1</td>
<td>11</td>
</tr>
<tr>
<td>1 1 0 0</td>
<td>12</td>
</tr>
<tr>
<td>1 1 0 1</td>
<td>13</td>
</tr>
<tr>
<td>1 1 1 0</td>
<td>14</td>
</tr>
<tr>
<td>1 1 1 1</td>
<td>15</td>
</tr>
</tbody>
</table>

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:

<table>
<thead>
<tr>
<th>FORMAT A</th>
<th>FORMAT B</th>
</tr>
</thead>
<tbody>
<tr>
<td>Start sentinel 1 character</td>
<td>Start sentinel 1 character</td>
</tr>
<tr>
<td>Format code &quot;A&quot; 1 character</td>
<td>Format code &quot;B&quot; 1 character</td>
</tr>
<tr>
<td>Surname</td>
<td>Account number Up to 19 chars</td>
</tr>
<tr>
<td>Surname separator &quot;/&quot;</td>
<td>Separator 1 character</td>
</tr>
<tr>
<td>Initials or first name</td>
<td>Surname</td>
</tr>
<tr>
<td>Separators (when required)</td>
<td>Surname separator &quot;/&quot;</td>
</tr>
<tr>
<td>= &quot;space&quot;</td>
<td>Initials or first name</td>
</tr>
<tr>
<td>FORMAT A</td>
<td>FORMAT B</td>
</tr>
<tr>
<td>------------------------------------------------------------------------</td>
<td>------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Title (when used)</td>
<td>Separator (when required) = &quot;space&quot;</td>
</tr>
<tr>
<td>Separator (when required) = &quot;space&quot;</td>
<td>Title (when used)</td>
</tr>
<tr>
<td>Separator 1 character</td>
<td>Separator (when required) = &quot;space&quot;</td>
</tr>
<tr>
<td>Discretionary data</td>
<td>Separator 1 character</td>
</tr>
<tr>
<td>The balance up to the maximum record length (79 characters)</td>
<td>Discretionary data</td>
</tr>
<tr>
<td>Stop sentinel 1 character</td>
<td>The balance up to the maximum record length (79 characters)</td>
</tr>
<tr>
<td>Longitudinal redundancy check, 1 character</td>
<td>Stop sentinel 1 character</td>
</tr>
<tr>
<td>Total 79 characters (max)</td>
<td>Longitudinal redundancy check</td>
</tr>
<tr>
<td></td>
<td>1 character</td>
</tr>
<tr>
<td></td>
<td>Total 79 characters (max)</td>
</tr>
</tbody>
</table>

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
HARDWARE DESCRIPTION

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

Table 2-10. Reader Specification

![Timing Diagram](image)

Figure 2-21. Timing Diagram

CHAPTER 2 PAGE 56
### HARDWARE DESCRIPTION

#### CHAPTER 2

<table>
<thead>
<tr>
<th>PIN #</th>
<th>SIGNAL</th>
<th>REMARKS</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td><strong>GND - GROUND</strong></td>
<td>GROUND FOR +5V AND SIGNAL GROUND</td>
</tr>
<tr>
<td>2</td>
<td><strong>RDP - READ DATA PULSE</strong></td>
<td>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'</td>
</tr>
<tr>
<td>3</td>
<td><strong>RCP - READ CLOCK PULSE</strong></td>
<td>THIS SIGNAL IS USED FOR READ DATA SAMPLING. (NEGATIVE LOGIC) THIS SIGNAL PERFORMS SAMPLING FOR SIGNAL RDP AT THE TRAILING EDGE.</td>
</tr>
</tbody>
</table>
### Table 2-11. Reader Control Signals

<table>
<thead>
<tr>
<th>PIN #</th>
<th>SIGNAL</th>
<th>REMARKS</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>CLS - CARD LOADING</td>
<td>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.</td>
</tr>
<tr>
<td>5</td>
<td>+5V</td>
<td>POWER SUPPLY</td>
</tr>
</tbody>
</table>
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

CHAPTER 2 PAGE 59
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.

![Diagram of clock generation]

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.
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.

**BI - DIRECTIONAL MAGNETIC CARD READER**

Figure 2-25. Interconnection between 8032 and 8751

Figure 2-26. Data Transfer Timing Diagram
12. KEYBOARD INTERFACE

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
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.
12.4 DESIGN IMPLEMENTATION

Figure 2-28. Keyboard Interface
13. SERIAL COMMUNICATIONS INTERFACE (ZILOG 8530)

13.1 8530 FUNCTIONAL DESCRIPTION

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.

Figure 2-29. 8530 Block Diagram
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:

<table>
<thead>
<tr>
<th>Address</th>
<th>EQU</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>COMSCB</td>
<td>0F000H</td>
<td>COMMS STATUS/CONTROL REG. CH B</td>
</tr>
<tr>
<td>COMDATB</td>
<td>0F001H</td>
<td>DATA REG. CH B</td>
</tr>
<tr>
<td>COMSCA</td>
<td>0F002H</td>
<td>COMMS STATUS/CONTROL REG. CH A</td>
</tr>
<tr>
<td>COMDATA</td>
<td>0F003H</td>
<td>DATA REG. CH A</td>
</tr>
</tbody>
</table>
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 (RTS) and Clear to Send (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
HARDWARE DESCRIPTION

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
HARDWARE DESCRIPTION

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 (DO). 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.

| VOUT | 1 | RDATA | OUTPUT |
| VSOURCE | 2 | RDATA RTN |
| - VIN | 3 | CLKin |
| + VIN | 4 | CLKin RTN |
| - VIN | 5 | RESETIN |
| + VIN | 6 | RESETIN RTN |

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.

<table>
<thead>
<tr>
<th>D7</th>
<th>D6</th>
<th>D5</th>
<th>D4</th>
<th>D3</th>
<th>D2</th>
<th>D1</th>
<th>D0</th>
<th>INFORMATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>P</td>
<td>L</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>P</td>
<td>L</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>P</td>
<td>L</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>P</td>
<td>L</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>P</td>
<td>L</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>P</td>
<td>L</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>P</td>
<td>L</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>6</td>
</tr>
<tr>
<td>P</td>
<td>L</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>7</td>
</tr>
<tr>
<td>P</td>
<td>L</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>8</td>
</tr>
<tr>
<td>P</td>
<td>L</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>9</td>
</tr>
<tr>
<td>P</td>
<td>L</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>A (NORMAL UPC ID)</td>
</tr>
<tr>
<td>P</td>
<td>L</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>B (RESERVED)</td>
</tr>
<tr>
<td>P</td>
<td>L</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>C (RESERVED)</td>
</tr>
<tr>
<td>P</td>
<td>L</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>D (RESERVED)</td>
</tr>
<tr>
<td>P</td>
<td>L</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>E (0 SUPPRESS ID)</td>
</tr>
<tr>
<td>P</td>
<td>L</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>F (EAN LABEL ID)</td>
</tr>
</tbody>
</table>
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.

<table>
<thead>
<tr>
<th>D7</th>
<th>D6</th>
<th>D5</th>
<th>D4</th>
<th>D3</th>
<th>D2</th>
<th>D1</th>
<th>D0</th>
<th>INFORMATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>P</td>
<td>0</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>NOT LAST BYTE</td>
</tr>
<tr>
<td>P</td>
<td>1</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>LAST BYTE</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>D7</th>
<th>D6</th>
<th>D5</th>
<th>D4</th>
<th>D3</th>
<th>D2</th>
<th>D1</th>
<th>D0</th>
<th>INFORMATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>L</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>ODD PARITY</td>
</tr>
<tr>
<td>1</td>
<td>L</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>EVEN PARITY</td>
</tr>
</tbody>
</table>

X = don't care

Table 2-12. Byte Format
14.6 DESIGN IMPLEMENTATION

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 damage 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
HARDWARE DESCRIPTION

Figure 2-36. 2006 System (Metro)
HARDWARE DESCRIPTION

2007 RESTAURANT SYSTEM

POWER
FAILURE
DETECTION

STANDARD XT
MOTHERBOARD

DUAL CHANNEL
COMMUNICATIONS
(RS232, RS485)

64K BYTES BATTERY
BACKED CMOS SRAM

REAL TIME CLOCK

PNET (RS485)

HARD DISK DRIVE
FLOPPY DISK DRIVE

ADS PROPRIETY
BOARDS

DEVICE #1
SCH30.DP

DEVICE #2

DEVICE #X

Figure 2-37. 2007 System (Restaurant)
INTELLIGENT POINT OF SALE TERMINAL

ASSEMBLER CODE LISTING

[Diagram of intelligent POS terminal with various connected devices]
ASSEMBLER CODE LISTING

ER 3

ASSEMBLER CODE LISTING

8.ASM

MAIN_MODULE

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

; 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, ENQBUF, ENQ_FULL

******************************************************************************
******************************************************************************
DATA SEGMENT
******************************************************************************
******************************************************************************
SEG SEGMENT DATA
******************************************************************************
******************************************************************************
SEGMENT DATA
******************************************************************************
******************************************************************************

RSEG DATA_SEG ;RELOCATABLE INTERNAL DATA

X: DS 60 ;60 BYTES FOR THE STACK
X_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
ASSEMBLER CODE LISTING

COUNT2: DS 01 ; REQUIRE A 16 BIT COUNTER
COUNT: DS 01 ; 0 = BOTH LCD'S, 1 = LCD_1, 2 = LCD_2
COUNT: DS 01 ; COUNT # OF BYTES WRITTEN TO LCD_BUF
COUNT: DS 01 ; COUNT # OF BYTES IN KEY_BUF
COUNT: DS 01 ; NUMBER OF BYTES IN TX_BUF
COUNT: DS 01 ; NUMBER OF BYTES IN RX_BUF
MODE: DS 01 ; COMMS MODE
BAR: DS 01 ; RECEIVED CHAR
ADDR: DS 01 ; BINARY VALUE OF TERMINAL ADDR
: DS 01 ; DEVICE ADDR IS A 2 BYTE ASCII VALUE
: DS 01
: DS 01 ; RX_BUF BCC COUNTER
: DS 01 ; TX_BUF BCC COUNTER
_COUNT: 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

TABLES RELATING TO 82530 COMMS CHIP
: DS 01 ; TO TOGGLE RTS LINE, HAVE TO WR TO
: DS 01 ; BUT THIS REG CAN'T BE READ,
: DFH: ; KEEP IMAGE OF THE VALUE FOR THIS REG
_STAT: DS 01 ; TRANSMISSION SUCCESSFUL
:OFFH=BAD
_COUNT: DS 01 ; # OF BYTES TO WRITE TO 82530 COULD
_COUNT: DS 01 ; # OF BYTES IN COM1_RXBUF

PAGE 4
ASSEMBLER CODE LISTING

CNT: 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

***********************************************************************
***********************************************************************
SEGMENT

************************************************************
******************
EG SEGMENT

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 KEY DATA IN KEY_BUF
LAG: DBIT 01 ; INDICATES LED DATA AVAILABLE
FLAG: DBIT 01 ; LOCK STATUS HAS CHANGED
_FLAGS: DBIT 01 ; THIS INDICATES THAT DRAW1 STATUS
          ; HAS CHANGED
_FLAGS: DBIT 01 ; THIS INDICATES THAT DRAW2 STATUS
          ; HAS CHANGED
FLAG: DBIT 01 ; THIS INDICATES THAT RECEIPT ON/OFF

ER 3 PAGE 5
ASSEMBLER CODE LISTING

;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 ;:O = 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
FLAG: 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 ;:O=DON'T TX CRD DATA. 1 = TX CRD DATA
PARITY: DBIT 01 ;:USED FOR PARITY CHECK
F_FULL: DBIT 01 ;:TX BUFFER IS FULL
R_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

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

ER 3 PAGE 6
ASSEMBLER CODE LISTING

INTERNAL DATA SEGMENT

SEGMENT RSEG EXT_SEG ;RELOCATABLE XTERNAL DATA SEGMENT
ORG 0

IF: DS 04 ;LED BUFFER
STAT: 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
IF: DS 08 ;8279 INTERNAL BUFFER = 8 BYTES
A: DS 50 ;COMMS BUFFER
A: DS 50 ;TRANSMIT DATA BUFFER
F: DS 40 ;40 CHARs FOR LCD
F: DS 41 ;STATUS+40 DATA BYTES. 01=OK,
AD
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

GER 3 PAGE 7
TABLES RELATING TO 82530 COMMS CHIP

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

CODE SEGMENT

CSEG AT 0 ;ABSOLUTE SEGMENT FOR MAIN MODULE
ORG 0
USING 1 ;USING RB 0,1
LJMP MAIN

; EQUATE (EQUATES.ASM)

Interrupt Vectors

ORG 03H ;VECTOR ADDR OF EXTERNAL INTO
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

************************************************************************
INTERRUPT ROUTINES
************************************************************************

;VECTOR ADDR OF EXTERNAL INT1
CLR TR0 ;STOP TIMER_0
SETB TMOUT ;TIMER HAS OVERFLOWED

RET
LONG INTERRUPT VECTOR ADDR

要不然, INT:

INT:

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 INTERRUPT 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

ANSFER CAN NOW TAKE PLACE. AFTER BUFFER TRANSFER COMPLETE, CAN START

0 RX MORE CHAR. 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 CHAR
IS USED TO COUNT TX CHAR
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 BOUNDARY

X_MODE 0 = WAIT FOR STX
1 = WAIT FOR DEVICE ADDRESS
2 = RECEIVE CHAR 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 CHAR 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

TER 3
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 MIDDLE OF STRING
ADD  A,#0 ;DIRECT ADD DOESN'T AFFECT ANY FLAGS
JZ  EXIT ;BUFFER EMPTY, EXIT
ASSEMBLER CODE LISTING

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

EOT1:
JNE      A,#EOT,CHK_MODE0 ;EOT, CHECK OTHER MODES
JNE      R4,#9,SET_MODE1 ;IF MODE 0, CHK FOR 'EOT'
JMP      CHK_MODE9

MODE1:
MOV      RX_MODE,#01   ;MODE = 1, WAIT FOR DEV. ADDR1
LJMP     COMMS_EXIT

MODE0:
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
ASSEMBLER CODE LISTING

;COMMS EXIT

LJMP COMMS_EXIT

;DE1:

CJNE R4,#01,CHK_MODE2;IF MODE 1, CHK FOR 'ADDR1'
CJNE A,ADDR1,INTERMEDIATE_EXIT_O ;!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_O ;!ADDR2, RESET RX_MODE
MOV RX_MODE,#03 ;MODE = 3, WAIT FOR POL/SEL
LJMP COMMS_EXIT

;DE3:

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_O ;!SEL, RESET RX_MODE
SETB POL_SEL ;0 = POL, 1 = SEL
MOV RX_MODE,#04 ;MODE = 4, WAIT FOR ENQ
LJMP COMMS_EXIT

;DE4:

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

PAGE 14
JB POL_SEL, SELECT ; 0 = POL, 1 = SEL

;TX_BUF_FULL, 0 = EMPTY, 1 = FULL
JB RE_TX_FLAG, RE_TX ; 1 = RE_TRASMIT 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

RE_TX_FINAL:
MOV RX_MODE, #05 ; WAIT FOR ACK/NAK
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
MO VX 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

;;;EDIATE_EXIT_0:
LJMP EXIT_0

;READY TO RECEIVE DATA
;MODE = 6, WAIT FOR DLE,STX
LJMP COMMS_EXIT

MO VX A,#ACK ;REASY TO RECEIVE DATA
LC ALL TX_BYTE
MO VX RX_MODE,#06 ;MODE = 6, WAIT FOR DLE,STX
LJMP COMMS_EXIT

MO VX A,#NAK ;TX_BUF IS FULL
LC ALL TX_BYTE
LJMP EXIT_0 ;RESET RX_MODE, WAIT FOR NEXT POLL

*s*****RE 3 PAGE 16
ASSEMBLER CODE LISTING

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

MODE5:

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 CONTENTS OF BUFFER
;WAIT FOR NEXT POLL TO TX MESSAGE
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

MODE6:

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

MODE7:

CJNE R4,#07,CHK_MODE8 ;IF MODE 7, CHK IF STX
CJNE A,#STX,EXIT_0 ;!STX, RESET RX_MODE

PAGE 17
ASSEMBLER CODE LISTING

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
ASSEMBLER CODE LISTING

RX_BCC:

MO VX @DPTR, A ; WRITE TO RX_BUF
INC DPTR
INC RX_COUNT
X RL A, RX_BCC
X CH A, RX_BCC ; SAVE BCC INTO RX_BCC
CLR RX_DLE
SJ MP 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
SJ MP EXIT_0

0:

MOV RX_MODE, #0 ; SET RX_MODE = 0, WAIT FOR EOT

EXIT:

POP ACC
MOV P2, A ; RESTORE MSB OF ADDR
MOV R5, DPL
MOV R6, DPH
POP DPH
POP DPL
POP PSW
POP B
ASSEMBLER CODE LISTING

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
**ASSEMBLER CODE LISTING**

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

CLR ;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

IALIZATION
MOV DPTR,#COM1_TXBUF ;INIT CHANNEL A OF 82530
MOV R7,#03
MOV COUNT82,#22
LCALL INIT_82530

TER 3 PAGE 21
ASSEMBLER CODE LISTING

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

;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

;TR:
MOV     DPTR,#COM2_TXBUF ;TRANSMIT DATA ON COM2
MOV     R7,#02 ;TX ON CH B
MOV     COUNT82,#35
LCALL   TX_COMX

82530:
LCALL   SETUP_TX ;IF DATA, THEN TX
LCALL   SETUP_RX ;IF DATA, THEN RX

TER 3
ASSEMBLER CODE LISTING

ENQUIRE:
    LCALL ENQUIRE          ; HOST REQUEST DRAW STATUS, LOCK

EC:
    LCALL REC_READ         ; STATUS 0=CLOSED, 1=OPEN

    MOVA, #"S"
    LCALL CHK_QUE
    LCALL SCAN_READ        ; IF QUE_FULL, WON'T READ SCANNER
    JNB SCAN_FLAG, LCD

    MOVA, #"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
    MOVA, LCD_NUM
    CJNE A, #30H, CHK_LCD1
    LCALL CLS1
    MOV DPTR, #LCD1_DAT_WR
    LCALL LCD_WR
ASSEMBLER CODE LISTING

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_QUEUE
LCALL KYB_READ
JNB KEY_FLAG,CHK_FLASH
MOV A,'#K'
LCALL ADD_QUEUE
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
ASSEMBLER CODE LISTING

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
ASSEMBLER CODE LISTING

RET

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

NSURE THAT CMOS ABOVE ALLOCATED MEMORY STILL CONTAINS 55H, AND IS NOT *
ORRUPTED *
************************************************************************

\AM:

MOV DPTR,#300H ;START ADDRESS ABOVE ALLOCATED

\LOOP2:

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
ASSEMBLER CODE LISTING

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

******************************************************************************
TRANSMIT A BYTE

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

ENTRY:

DPTR POINTS TO SOURCE ADDRESS

MOV R0,#50
_LP1:

MOVX A,@DPTR
LCALL SEND_BYTE
ASSEMBLER CODE LISTING

INC        DPTR
DJNZ       R0, SEND_LPI

RET

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

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

AFT FOR A CERTAIN PERIOD OF TIME

ENTRY:

NOTHING

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

PUSH       ACC
MOV        A, R0
PUSH       ACC
MOV        A, R1
PUSH       ACC
MOV        R0, #0FFH
MOV        R1, #0FH

l:

DJNZ       R0, WAIT1
MOV        R0, #0FFH
DJNZ       R1, WAIT1
POP         ACC
MOV        R1, A
POP         ACC
WRITE A PARTICULAR VALUE ONTO THE STACK WHICH IS CHECKED EACH TIME THROUGH THE MAIN LOOP, IF THIS VALUE DOES CHANGE THE STACK IS GROWING TOO HIGH AND A STACK OVERFLOW IS IMINENT

ON 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

; CK:
MOV       R0,#STACK_END-10 ; 60 - 10 = STACK+50
MOV  A,@RO
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
TRANSFER DATA FROM EXTERNAL DATA INTO LCD_BUF

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
Z2: DB 2,'0 82530 IS FULLY OPERATIONAL!!! ',3,0
*: DB 2,'0 THE POWER AT POINT OF SALE',3,0

TER 3 PAGE 33
ASSEMBLER CODE LISTING

DB 'STACK OVERFLOW @ STACK + 56 !!!',0
DB 'MEMORY ERROR!!!',0
DB 'PASSED MEM FILL',0
DB 'INTERRUPT ERROR',0

END
ASSEMBLER CODE LISTING

l.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)

N 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

TER 3 PAGE 35
INITIALIZE HARDWARE

INITIALIZE VARIABLES IN EXTERNAL DATA MEMORY AFTER THE RAM TEST

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,#OFBH ;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 =
ASSEMBLER CODE LISTING

; 12 CLOCK CYCLES, = 1/12 CLK FREQ
; XTAL = 9.216MHz, TICK = 1.3uSEC
; XTAL = 1.8MHz, TICK = 6.7uSEC

ORL TMOD,#01 ; CONTROL BY TRx, 16 BIT, TIMER

INT:

SETB EA ; ENABLE ALL INTERRUPTS
SETB ETO ; ENABLE TIMER 0 INT
CLR ES ; DISABLE SERIAL PORT INT
ORL IP,#18H ; INT PRIORITY, SERIAL & TO
CLR TR0 ; STOP TIMER 0

RAM:

; TEST LOWER 256 BYTES OF XDATA
; AS THE LCD_BUF RESIDES IN THIS AREA
; AS OFFH IN LCD_BUFFER, NEVER FINDS
; EOS
; AND SITS IN LCD_WR PERMANENTLY

MOV DPTR,#0 ; START ADDR
MOV R1,#OFFH
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

TEST_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 RO,#1FH ;AT POWER UP, LCD IS INTERNALLY BUSY
MOV DPTR,#LCD1_STAT ;WAIT UNTIL THEY ARE READY

: l:

CLR TRO ;STARTunset Timer_0
MOV TH0,#0 ;USED FOR TIMEOUT LOOP
MOV TLO,#0
SETB TRO ;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 TRO ;STOP Timer_0
SJMP WAIT_LCD2 ;LCD OPERATING NORMALLY

: 3:

CLR TMOUT
DJNZ R0,WAIT1
ASSEMBLER CODE LISTING

MOV COUNT,#02 ;LCD ERROR HAS OCCURRED
LCALL ERR_BEEP ;WAIT IN PERMANENT LOOP

_LCD2:

MOV R0,#1FH ;AT POWER UP, LCD IS INTERNALLY BUSY
MOV DPTR,#LCD2_STAT ;WAIT UNTIL THEY ARE READY

;:
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

TER 3

PAGE 39
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
ASSEMBLER CODE LISTING

MOV   DPTR,#0         ;START ADDRESS
MOV   R0,#0FFH
MOV   R1,#7FH
MOV   A,#33H         ;WRITE PATTERN TO RAM

OOP1:

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
ASSEMBLER CODE LISTING

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

KYB:

MOV    DPTR,#KYB_CMD    ; ENCODED SCAN, 2 KEY LOCKOUT
MOV    A,#24H        ; F = 333KHz. DIVIDE 1.8MHz CLK BY 4
ASSEMBLER CODE LISTING

MOV A,#2EH ;F = 1.6MHz. DIV 9.216MHz CLK BY
MOVX @DPTR,A ;15

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

4_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

\texttt{STAT1:}

\begin{verbatim}
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
\end{verbatim}

\texttt{STAT2:}

\begin{verbatim}
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
\end{verbatim}

T_SCAN:

\begin{verbatim}
CLR SCAN_FLAG
LCALL INIT_SCAN
\end{verbatim}

M_NUM:

\begin{verbatim}
LCALL CLS1
MOV DPTR,#MSG5
LCALL CODE_TO_LCD
\end{verbatim}

\begin{verbatim}
MOV DPTR,#LCD_BUF+22
MOV A,ADDR1
MOVX @DPTR,A
INC DPTR
MOV A,ADDR2
MOVX @DPTR,A
\end{verbatim}
ASSEMBLER CODE LISTING

MOVDPTR,#LCD1_DAT_WR
LCALL LCD_WR
LCALLBEEP

EXIT:

SETBES; ENABLE SERIAL INTERRUPT

RET

READ:

PUSHDPL
PUSHDPH

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

READ 8279 FOR KEYBOARD DEPRESSIONS

~ IT IS IMPORTANT TO USE THE CORRECT REGISTERS AND THEIR CORRECT

FOR CASES WHERE A 1 BYTE REGISTER IS USED INSTEAD OF THE DPTR

FOR EXTERNAL DATA TRANSFERS, THE ADDRESS AREA IS ONLY 256 BYTES

00 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

00 IS USED TO ACCESS KEY_BUF IN EXTERNAL 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

***********************************************************************
ASSEMBLER CODE LISTING

;OP1:
    JB    QUE_FULL,DUMMY_READ
    MOV   A,KEY_COUNT
    CJNE  A,#08,CHK_IRQ
    SJMP  DUMMY_READ

:IQR:
    JNB   INTO,KYB_EXIT
    MOV   DPTR,#KYB_CMD
    MOVX  A,@DPTR
    ANL   A,#0FH
    JZ    KYB_EXIT
    MOV   DPTR,#KYB_CMD
    MOVX  A,@DPTR
    ANL   A,#30H
    JNZ   KYB_ERR
    MOV   A,#50H
    MOVX  @DPTR,A
    MOV   DPTR,#KYB_DAT
    MOVX  A,@DPTR
    MOV   DPTR,#KEY_TAB
    MOVX  A,@A+DPTR

    MOV   DPTR,#KEY_BUF
    MOV   B,A
    MOV   A,DPL
    ADD   A,KEY_COUNT

    CHK_IRQ:       iCHECK IF KEY_BUF IS FULL
    iIF 8279 IRQ 'L', NO KEY DEPRESSED
    iREAD 8279 STATUS
    iNUMBER OF KEYS IN FIFO, BUFFER FULL
    iIF ACC = 0, THEN NO KEYS IN FIFO
    iREAD STATUS
    iCHECK OVER/UNDERRUN FLAG
    i1 OF THE 2 BITS WAS SET
    iWR TO 8279, ALLOWING US TO READ FIFO
    iREAD FIFO
    iREADING FIFO
    iCONVERT RAW KEY VALUE
    iSTORE CONTENTS OF ACC
    iEVERY KEY, INCREASE OFFSET TO
ASSEMBLER CODE LISTING

;UF

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

;KEY_READ:

JNB INTO,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,@OPTR ;READING FIFO

;EXIT:

POP DPH
POP DPL

RET
**ASSEMBLER CODE LISTING**

*************************************************************************
CONVERT HEX VALUE INTO ITS DECIMAL ASCII EQUIVALENT.

**ON ENTRY:**

ACC = BYTE TO BE CONVERTED

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

**TO_DEC:**

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>MOV R1,A</td>
<td>;SAVE ACC</td>
</tr>
<tr>
<td>MOV R0,#LCD_BUF+3</td>
<td></td>
</tr>
<tr>
<td>MOV A,#SPACE</td>
<td></td>
</tr>
<tr>
<td>MOV P2,#0</td>
<td>;IN CONJUNCTION WITH MOVX R0/R1</td>
</tr>
<tr>
<td>MOVX @R0,A</td>
<td></td>
</tr>
<tr>
<td>DEC R0</td>
<td></td>
</tr>
<tr>
<td>MOV P2,#0</td>
<td>;IN CONJUNCTION WITH MOVX R0/R1</td>
</tr>
<tr>
<td>MOVX @R0,A</td>
<td></td>
</tr>
<tr>
<td>DEC R0</td>
<td></td>
</tr>
<tr>
<td>MOV P2,#0</td>
<td>;IN CONJUNCTION WITH MOVX R0/R1</td>
</tr>
<tr>
<td>MOVX @R0,A</td>
<td></td>
</tr>
<tr>
<td>DEC R0</td>
<td></td>
</tr>
<tr>
<td>MOV P2,#0</td>
<td>;IN CONJUNCTION WITH MOVX R0/R1</td>
</tr>
<tr>
<td>MOVX @R0,A</td>
<td>;CLEAR OLD DATA FROM LCD_BUF</td>
</tr>
<tr>
<td>MOV R0,#LCD_BUF+4</td>
<td></td>
</tr>
<tr>
<td>MOV A,#EOS</td>
<td></td>
</tr>
<tr>
<td>MOV P2,#0</td>
<td>;IN CONJUNCTION WITH MOVX R0/R1</td>
</tr>
<tr>
<td>MOVX @R0,A</td>
<td></td>
</tr>
<tr>
<td>DEC R0</td>
<td></td>
</tr>
</tbody>
</table>
;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:

TER 3  PAGE 51
ASSEMBLER CODE LISTING

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
ASSEMBLER CODE LISTING

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_ADDR 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 X001H
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,@RO ;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 CHAR, WRITE TO LINE #2
CJNE A,COUNT,LCD_LOOP4

MOV R7,#0FFH
DJNZ R7,$
MOV R7,#0FFH
DJNZ R7,$
ASSEMBLER CODE LISTING

MOV A,#LINE_2
PUSH DPL
MOV DPL,#0 ;WRITE LCD X000H CMD_WR
MOVX @DPTR,A
POP DPL ;RESTORE ORIGINAL ADDRESS
MOV R7,#OFFH
DJNZ R7,$
MOV R7,#OFFH
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
ASSEMBLER CODE LISTING

CD_WR:

MOV COUNT,#02
LCALL ERR_BEEP

RET

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

;LEAR LCD #1

;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

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

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

LEAR LCD #2

*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

ER 3
ASSEMBLER CODE LISTING

```
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
```

*END OF ASSEMBLER CODE LISTING*
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
MOV 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

************
TRANSFER DATA FROM INTERNAL DATA AREA TO INTERNAL DATA INTO TEMP_BUF

ENTRY:
R1 POINTS TO SOURCE ADDR

TO_TEMP:

_LOOP1:

RET

BEEP THE BUZZER

:  

PUSH DPL  
PUSH DPH  
MOV DPTR,#BUZZER  
MOvx @DPTR,A  
POP DPH
ASSEMBLER CODE LISTING

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
BEEP THE BUZZER TO INDICATE A PARTICULAR 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

IN ENTRY:
COUNT = # OF BEEPS

***************************************************** *******************
IN: OOP1:
LCALL BEEP
MOV R0, #0FFH
MOV R1, #0FFH

OOP2:
DJNZ R0, ERR_LOOP2
MOV R0, #0FFH
DJNZ R1, ERR_LOOP2
DJNZ COUNT, ERR_LOOP1
JMP $ ;*******SIT IN PERMANENT LOOP
ASSEMBLER CODE LISTING

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**

<table>
<thead>
<tr>
<th>Tabulations</th>
<th>Values</th>
<th>Decimals</th>
</tr>
</thead>
<tbody>
<tr>
<td>DB</td>
<td>00,04,13,22,31,54,45,27,00,03</td>
<td>00 - 09</td>
</tr>
<tr>
<td>DB</td>
<td>12,21,30,53,44,26,00,19,46,65</td>
<td>10 - 19</td>
</tr>
<tr>
<td>DB</td>
<td>56,38,47,39,00,00,00,00,00,00</td>
<td>20 - 29</td>
</tr>
<tr>
<td>DB</td>
<td>00,00,00,00,00,00,00,00,00,00</td>
<td>30 - 39</td>
</tr>
<tr>
<td>DB</td>
<td>00,02,11,20,29,50,41,23,00,10</td>
<td>40 - 49</td>
</tr>
<tr>
<td>DB</td>
<td>37,55,66,51,42,24,00,01,28,64</td>
<td>50 - 59</td>
</tr>
<tr>
<td>DB</td>
<td>67,52,43,25,36,18,09,63,72,00</td>
<td>60 - 69</td>
</tr>
<tr>
<td>DB</td>
<td>00,00,35,17,08,62,71,00,00,00</td>
<td>70 - 79</td>
</tr>
<tr>
<td>DB</td>
<td>40,48,49,58,57,00,00,00,00,00</td>
<td>80 - 89</td>
</tr>
<tr>
<td>DB</td>
<td>00,00,00,00,00,00,00,00,00,00</td>
<td>90 - 99</td>
</tr>
<tr>
<td>DB</td>
<td>00,00,00,00,32,14,05,59,68,00</td>
<td>100 - 109</td>
</tr>
<tr>
<td>DB</td>
<td>00,00,33,15,06,60,69,00,00,00</td>
<td>110 - 119</td>
</tr>
<tr>
<td>DB</td>
<td>34,16,07,61,70,00,00,00,00,00</td>
<td>120 - 129</td>
</tr>
</tbody>
</table>

**TEC 84 KEY KEYBOARD, ITALIAN KEYBOARD**

<table>
<thead>
<tr>
<th>Key Tabulations</th>
<th>Values</th>
<th>Decimals</th>
</tr>
</thead>
<tbody>
<tr>
<td>DB</td>
<td>00,00,00,00,00,00,00,00,00,84,83</td>
<td>00 - 09</td>
</tr>
<tr>
<td>DB</td>
<td>82,81,79,80,78,76,36,35,34,33</td>
<td>10 - 19</td>
</tr>
<tr>
<td>DB</td>
<td>31,32,30,28,24,23,22,21,19,20</td>
<td>20 - 29</td>
</tr>
</tbody>
</table>
ASSEMBLER CODE LISTING

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, 17, 15 ; 80 - 89
DB 14, 13, 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
ASSEMBLER CODE LISTING

; 120 - 129

DB 00, 00, 00, 00, 00, 00, 00

MSG1: DB '*ADS* THE POWER AT POINT OF SALE', 0
MSG1: DB ' THE POWER AT POINT OF SALE', 0
MSG2: DB '******RAM ERROR AT POWER-UP*****', 0
MSG3: DB ' PERFORMING RAM TEST ', 0
MSG4: DB 'PASSED RAM TEST!', 0
MSG5: DB 'PASSED RAM TEST! TERM # V1.2', 0

INCLUDE (EQUATES.ASM)

END
INPUT_OUTPUT_MODULE_2

ALL I/O ROUTINES IN THIS MODULE

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

EXTERN DATA (COUNT, TEMP_BUF, LED_COUNT1, LED_COUNT2, CRD_COUNT)
EXTERN BIT (LED_FLAG, LOCK_FLAG, DRAW1_FLAG, TMOUT, CRD_FLAG, CRD_ON_OFF)
EXTERN BIT (REC_FLAG, DRAW2_FLAG)

EXTERN XDATA (LOCK_STAT, DRAW1_STAT, KEY_BUF, RX_BUF, TX_BUF, LED_BUF,
            MD_BUF)
EXTERN XDATA (REC_STAT, DRAW2_STAT, LCD_BUF)

EXTERN CODE (INIT_HARDWARE, LCD_WR, CODE_TO_LCD, BEEP, TEST, KYB_READ)
EXTERN CODE (IDATA_TO_TEMP, HEX_TO_ASCII, HEX_TO_DEC, CLS1)

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

O_2_SEG SEGMENT CODE ;RELOCATABLE CODE SEGMENT
RSEG I_O_2_SEG

CHAPTER 3 PAGE 67
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 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 AVAILABLE

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
ASSEMBLER CODE LISTING

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
* TRANSMISSION

* ON ENTRY:
* NOTHING

* ON EXIT:
ASSEMBLER CODE LISTING

ACC = BYTE READ

;***************************************************** ***********************
;R DATA:

MOV R2,#08 ;8 BITS
CLR A

;***************************************************** ***********************
;R_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

;***************************************************** ***********************
ASSEMBLER CODE LISTING

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
ASSEMBLER CODE LISTING

I 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.

I 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 ASSOCIATED 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 CONFUSE THIS WITH A CONTROL CHAR. AFTER RECEIVING 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 REQUIRE 4 BYTES, 2 = LED ON/OFF AND 2 FOR LED FLASH
`ASSEMBLER CODE LISTING

LED_ON_OFF:

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLR</td>
<td>CLEAR LED_FLAG</td>
</tr>
<tr>
<td>MOV</td>
<td>D PTR, #RX_BUF+2; READ RX DATA, EXCLUDE I/O, TYPE #</td>
</tr>
<tr>
<td>MOV</td>
<td>R1, #LED_BUF; BUFFER ADDR</td>
</tr>
<tr>
<td>MOV</td>
<td>COUNT, #8; 8 BYTE LED BUFFER COUNTER</td>
</tr>
</tbody>
</table>

LED_LOOP:

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MOVX</td>
<td>A, @DPTR; READ RX_BUF</td>
</tr>
<tr>
<td>ANL</td>
<td>A, #0FH; MASK OFF MS NIBBLE</td>
</tr>
<tr>
<td>MOV</td>
<td>P2, #0; IN CONJUNCTION WITH MOVX R0/R1</td>
</tr>
<tr>
<td>MOVX</td>
<td>@R1, A; LOWER 4 BITS INTO LED_BUF</td>
</tr>
<tr>
<td>DEC</td>
<td>COUNT; BYTE COUNTER</td>
</tr>
<tr>
<td>INC</td>
<td>DPTR; NEXT BYTE IN COMBUF</td>
</tr>
<tr>
<td>MOVX</td>
<td>A, @DPTR; READ RX_BUF</td>
</tr>
<tr>
<td>ANL</td>
<td>A, #0FH; MASK OFF MS NIBBLE</td>
</tr>
<tr>
<td>SWAP</td>
<td>A; SHIFT LEFT BY 4 BITS</td>
</tr>
<tr>
<td>MOV</td>
<td>B, A; SAVE ACC</td>
</tr>
<tr>
<td>MOV</td>
<td>P2, #0; IN CONJUNCTION WITH MOVX R0/R1</td>
</tr>
<tr>
<td>MOVX</td>
<td>A, @R1; READ LED_BUF</td>
</tr>
<tr>
<td>ORL</td>
<td>A, B; OR NEXT NIBBLE</td>
</tr>
<tr>
<td>MOV</td>
<td>P2, #0; IN CONJUNCTION WITH MOVX R0/R1</td>
</tr>
<tr>
<td>MOVX</td>
<td>@R1, A; RESULT INTO LED_BUF</td>
</tr>
<tr>
<td>INC</td>
<td>DPTR</td>
</tr>
<tr>
<td>INC</td>
<td>R1; WRITE TO NEXT ADDR</td>
</tr>
<tr>
<td>DJNZ</td>
<td>COUNT, LED_LOOP; NEXT BYTE</td>
</tr>
</tbody>
</table>

; TRANSFER FROM 8 BYTE RX_BUF
; INTO 4 BYTE LED_BUF COMPLETE
ASSEMBLER CODE LISTING

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

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
ASSEMBLER CODE LISTING

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

**K1:**

MOV A,B
MOV DPTR,#LOCK_STAT
MOVX @DPTR,A ;SAVE NEW STATUS
SETB LOCK_FLAG ;LOCK STATUS HAS CHANGED
LCALL BEEP

**K_EXIT:**

RET
ASSEMBLER CODE LISTING

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.

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

REC_STAT:

MOV A,B ;SAVE CURRENT REC STATUS
MO VX @DPTR,A
SETB REC_FLAG ;INDICATES STATUS HAS CHANGED

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.
ASSEMBLER CODE LISTING

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

DRAW1_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

NEW1_STAT:

MOV A,B ;SAVE CURRENT DRAW STATUS
MOVX @DPTR,A
SETB DRAW1_FLAG ;INDICATES STATUS HAS CHANGED

DRAW1_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.

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

PETER 3 PAGE 79
ASSEMBLER CODE LISTING

DRAW2_READ:
    MOV    A,#0            ;INIT ACC
    MOV    C,#1.6          ;READ CURRENT DRAW STATUS
    RLC
    MOV    B,A
    MOV    DPTR,#DRAW2_STAT ;READ PREVIOUS STATUS
    MOVX   A,@DPTR
    CJNE   A,B,NEW2_STAT   ;DRAW HAS CHANGED
    SJMP   DRAW2_EXIT

NEW2_STAT:
    MOV    A,B            ;SAVE CURRENT DRAW STATUS
    MOVC   @DPTR,A
    SETB   DRAW2_FLAG    ;INDICATES STATUS HAS CHANGED

DRAW2_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

PTER 3         PAGE 80
ASSEMBLER CODE LISTING

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

RET

***************************************************************************
***************************************************************************
INCLUDE (EQUATES.ASM)
END
COMMS.ASM

; COMMS_MODULE

LOW LEVEL AS WELL AS UPPER LEVEL COMMS ROUTINES

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

BLIC TEST_COMMS, CHK_TX_BUFFS, CHK_RX_BUFFS, TX_STRING

TRN DATA (COUNT, TEMP_BUF, KEY_COUNT, TX_COUNT, RX_COUNT, LCD_NUM)

TRN DATA (LCD_COUNT, TX_BCC, RX_BCC, ADDR1, ADDR2, CRD_COUNT, QUE_COUNT)

TRN DATA (BIN_ADDR)

TRN 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)

TRN BIT (CRD_ON_OFF, REC_FLAG, DRAW2_FLAG)

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

TRN XDATA (TX_DATA, CRD_BUF, KEY_QWE, REC_STAT, DRAW2_STAT)

TRN CODE (INIT_HARDWARE, LCD_WR, CODE_TO_LCD, BEEP, TEST, KYB_READ)

TRN CODE (IDATA_TO_TEMP, HEX_TO_ASCII, HEX_TO_DEC, LED_ON_OFF, LED_FLASH)

TRN CODE (LOCK_READ, DRAW1_OPEN, DRAW2_OPEN, CLS1, CLS2)

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

"TER 3"
ASSEMBLER CODE LISTING

ASSEMBLER CODE LISTING

I TRN BIT (ENQ_FLAG, ENQ_FULL)

TRN XDATA (ENQ_BUF)

CALL ENQUIRE

***** 82530 COMMS CHIP

TRN DATA (TX82_STAT, COUNT82, RX1_COUNT, RX2_COUNT)

TRN BIT (TXBUF1_FULL, TXBUF2_FULL, RXBUF1_FULL, RXBUF2_FULL)

TRN XDATA (COM1_TXBUF, COM1_RXBUF, COM2_TXBUF, COM2_RXBUF)

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

RBSEG 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

PAGE 83
ASSEMBLER CODE LISTING

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:
   RO 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
ASSEMBLER CODE LISTING

 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

 PERMEDIATE_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

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

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

: MOVX A, @R0 ; READ TX_BUF
ASSEMBLER CODE LISTING

JB BCC_FLAG, CHK_ETX
CJNE A, #DLE, CALC_BCC
SETB BCC_FLAG
INC R0
SJMP BCC ;READ NEXT BYTE

rETX:

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

PTER 3

PAGE 87
ASSEMBLER CODE LISTING

MOVL SBUF,A ;START TX PROCESS
MOVL R0,#TX_BUF+1 ;SET R0 = START ADDR +1 OF TX_BUF

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
VARIABLE 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
ASSEMBLER CODE LISTING

'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 TEMPORARY 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
ASSEMBLER CODE LISTING

MO VX @ 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 ADDR

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

TER 3
ASSEMBLER CODE LISTING

INC     R7
DJNZ    KEY_COUNT, CHK_LOOP1 ;NEXT BYTE TO TRANSFER
MOV     KEY_COUNT, #0        ;CLEAR COUNTER
LJMP    CHK_EXIT

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

ERR_CHK_EXIT1:
    LJMP    CHK_EXIT1          ;INTERMEDIATE JMP

CRD:

PTER 3
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

;CRD10:

;CRD11:

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

* 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
ASSEMBLER CODE LISTING

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

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

;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
ASSEMBLER CODE LISTING

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

PAGE 97
ASSEMBLER CODE LISTING

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
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

IS_DATA:

MOVX A,@DPTR ;READ FROM COM1/2_RXBUF
INC DPTR
MOVX @RD,A ;WRITE TO TX_DATA BUFFER
INC R1
INC R7
DJNZ R6,TRANS_DATA

RET
TEST COMMS

r_ COMMS :

MOV R0, #TX_BUF
MOV R1, #20H
MOV A, #30H
MOV P2, #0

JOP1:

MOVX @R0, A
INC R0
INC A
DJNZ R1, T_LOOP1

; IN CONJUNCTION WITH MOVX R0/R1

; TX_BUF NOW CONTAINS STRING

SETB PSW.3
MOV TX_COUNT, #20H
MOV R3, TX_COUNT
MOV R0, #TX_BUF
CLR PSW.3
SETB TX_BUF_FULL
CLR P1.7
CLR P3.5
MOV SBUF, #0FH

; SELECT REGISTER BANK 1, 8H - FH
; USED IN COMMS INT ROUTINE
; DON'T CHANGE TX_COUNT, IN CASE RE-TX
; SET R0 = START ADDR OF TX_BUF
; USED IN COMMS INT ROUTINE
; SELECT REGISTER BANK 0
; CLEARED IN INT ROUTINE
; ENABLE DS3695 TO TX, GOES THRU 7404
; ENABLE DS3695 TO TX, GOES THRU 7404
; START TX PROCESS
RETI

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

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 DATA ie DON'T READ

TER 3
3 BYTE
IN THE CASE OF A HOST ENQUIRY
'\texttt{R}' = DRAWER STATUS
'\texttt{L}' = LOCK STATUS
'\texttt{P}' = RECEIPT STATUS

3 BYTE
IN THE CASE OF TX STRING FOR 82530
BININARY \# 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
'\texttt{1}' = DRAWER #1
'\texttt{2}' = DRAWER #2

4 BYTE
IN THE CASE OF TX STRING FOR 82530
'\texttt{1}' = COM1
'\texttt{2}' = COM2

5 BYTE
IN THE CASE OF TX STRING FOR 82530
'\texttt{C}' = COMMAND STRING
'\texttt{D}' = DATA STRING

\texttt{R0} USED FOR READING
\texttt{R1} USED FOR WRITING TO BUFFER
\texttt{COUNT} USED TO COUNT \# OF BYTES TO TRANSFER FROM RX\_BUF TO OUTPUT\_BUF

\texttt{RX\_BUFFS:}
\begin{verbatim}
  JB RX_BUF_FULL, CHK_BUFFS ; DATA AVAILABLE IN RX_BUF
\end{verbatim}

\texttt{\textasciitilde 3 PAGE 102}
ASSEMBLER CODE LISTING

LJMP CHK_RX_EXIT1

BUFS:
MOV DPTR,#RX_BUF
MO VX A,@DPTR ;READ RX_BUF
INC DPTR
CJNE A,#'O',INTERMEDIATE_RX_EXIT ;NOT FOR AN OUTPUT DEVICE
MO VX 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
MO VX 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
ASSEMBLER CODE LISTING

SJMP ADD_EOS

MOV A, LCD_COUNT
CJNE A, #32, CHK1 ; MAX OF 32 BYTES IN LCD_BUF
SJMP ADD_EOS

MOVX A, @DPTR ; READ RX_BUF
CJNE A, #32, CHK1
SJMP ADD_EOS

MOVX A, @DPTR ; READ RX_BUF
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, #' ' ; IN CONJUNCTION WITH MOVX R0/R1
MOV P2, #0
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
ASSEMBLER CODE LISTING

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
ASSEMBLER CODE LISTING

LCALL DRAW1_OPEN
SJMP CHK_RX_EXIT

DRAW2:
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

CARD_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

TER 3

PAGE 106
TRANSFER FROM RX_BUF(8032) TO COMX_BUF(82530)

R6 AND R7 IS USED FOR TEMPORARY 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)

`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
ASSEMBLER CODE LISTING

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
ASSEMBLER CODE LISTING

SJMP RXBUF_EXIT

TXBUF2:
CJNE R4,#02,RXBUF_EXIT
SETB TXBUF2_FULL

IF_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,@RO
MOVX @DPTR,A
INC DPTR

MOV R0,#DRAW2_STAT
MOV P2,#0 ;IN CONJUNCTION WITH MOVX R0/R1
MOVX A,@RO
MOVX @DPTR,A
ASSEMBLER CODE LISTING

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

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

LUDGE (EQUATES.ASM)

END
ASSEMBLER CODE LISTING

SCAN.ASM

; SCANNER_MODULE

MODULE FOR SLOT SCANNER INTERFACE (OCIA)

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

~ CODE (WAIT, TEST, BEEP)  **********************************************************

~***************************************************** ********************

\SEG SEGMENT

RSEG

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 KNOW 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

TER 3

PAGE 112
CHECK IF THE KEY QUE CAN ACCOMODATE 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

---

THE USER.

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

---

ASSEMBLER CODE LISTING
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:
ASSEMBLER CODE LISTING

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 ADDR
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 ADDR
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:
MOVP2,#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
**
**************************************************************************

_SCAN:

MOV SCAN_COUNT,#0
CLR SCAN_FLAG
CLR P1.1 ;CLK 'L'
MOV R7,#08

_SCAN_LP1:
ASSEMBLER CODE LISTING

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

RET

~***************************************************** *********************
~***************************************************** *********************

READ SLOT SCANNER

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

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

PAGE 117
ASSEMBLER CODE LISTING

;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
MO VX @R0,A
INC R0
JB B.6,SCAN_EXIT1 ;LAST BYTE WAS READ
DJNZ R6,SCAN_READ_LP1

EXIT1:

SETB SCAN_FLAG ;SCAN DATA AVAILABLE
SJMP SCAN_EXIT

_ERR:

MOV SCAN_COUNT,#0

_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

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.

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

OUT:
CLR TR0 ; STOP TIMER_0
MOV TH0, #0 ; USED FOR TIMEOUT LOOP
MOV TLO, #0
SETB TR0 ; START TIMER_0
ASSEMBLER CODE LISTING

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

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

INCLUDE (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 ASSOCIATED '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

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

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

FN DATA (TX82_STAT, COUNT82, PARITY, ST_BITS, TX_DATA_BITS)
FN DATA (RX_DATA_BITS, BAUD, RTS, RTSB, RX1_COUNT, RX2_COUNT)
FN XDATA (COM1_TXBUF, COM2_TXBUF, COM1_RXBUF, COM2_RXBUF)
FN 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 RECEIVE SOFTWARE

RECEIVE 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 BUFFER (COM1_RXBUF/COM2_RXBUF)
R7 = 2 COMDATB ; DATA REG. CH B
4 COMDATA ; DATA REG. CH A

REG 6 = # OF BYTES CURRENTLY 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
ASSEMBLER CODE LISTING

SUBB A,R6
JNC RX_COMX_LP1
MOV R6,#40
SJMP RX_EXIT

RX_COMX_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

RX_EXIT:
PUSH DPL
PUSH DPH
LCALL CHK_CHAN
MOVX A,@DPTR ;READ CHAR

PAGE 125
ASSEMBLER CODE LISTING

 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

 JMPX_LP2:
 MOV   A,B   ;RESTORE RX CHAR
 MO VX   @DPTR,A   ;WR RX CHAR TO COM1/2_RXBUF
 INC   R6
 SJMP  RX_COMX   ;CHK IF ANOTHER BYTE AVAILABLE

 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 (DEFAULT 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 BUFFER (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
ASSEMBLER CODE LISTING

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

RTS 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

;INITIALIZATION 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_RTC

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
ASSEMBLER CODE LISTING

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

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
ASSEMBLER CODE LISTING

; INITIALIZATION 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
TOGGLE THE RTS LINE

ON ENTRY:

DPTR POINTS TO SOURCE BUFFER (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 BUFFER (COM1_TXBUF/COM2_TXBUF)
ASSEMBLER CODE LISTING

MOVX @DPTR, A
POP DPH
POP DPL

RET

SETUP INITIALIZATION STRING TO INITIALIZE 82530

ON ENTRY:

DPTR POINTS TO SOURCE BUFFER (COM1_TXBUF/COM2_TXBUF)

TUP_CMD_STR:

MOV RO,#RX_DATA_BITS ;READ RX BIT RATE
MOV A,@RO
INC DPH
INC DPH
INC DPH
MOVX @DPTR, A ;SET NEW BIT RATE

MOV RO,#PARITY ;READ PARITY BYTE, INCLUDES ST BITS
MOV A,@RO
INC DPH
INC DPH
INC DPH
MOVX @DPTR, A ;SET NEW PARITY AND STOP BITS
MOV RO,#TX_DATA_BITS ;READ TX BIT RATE
MOV A, @R0
INC DPTR
INC DPTR
INC DPTR
INC DPTR
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
MOVX @DPTR, A
; SET NEW BAUD RATE

RET

***************************************************************************
***************************************************************************
SETUP COMMS OPTIONS TO INITIALIZE 82530 FROM HOST

ON ENTRY:

DPTR POINTS TO SOURCE BUFFER (COM1_TXBUF/COM2_TXBUF)
OFFSET @ BAUD RATE

***************************************************************************
***************************************************************************
MS_OPTIONS:

MOVX A, @DPTR
; SET BAUD RATE

PTER 3
INC DPTR

;SET BAUD RATE

CJNE A,#48H,CHK96 ;4.8K BAUD
MOV A,#16H
SJMP SET_BAUD

CJNE A,#96H,CHK19 ;9.6K BAUD
MOV A,#0AH
SJMP SET_BAUD

CJNE A,#19H,CHK38 ;19.2K BAUD
MOV A,#04
SJMP SET_BAUD

CJNE A,#38H,CHK_DATA_BITS ;38.4K BAUD
MOV A,#01

BAUD:
MOV R0,#BAUD
MOV @R0,A

DATA_BITS:
MO VX A,@DPTR ;SET # OF DATA BITS
INC DPTR

CJNE A,#7H,CHK8
MOV A,#41H ;# OF RX BITS
MOV B,#28H ;# OF TX BITS
ASSEMBLER CODE LISTING

SET_DATA_BITS:

RO,#RX_DATA_BITS
@RO,A
RO,#TX_DATA_BITS
@RO,B

T_DATA_BITS:

MOV R0,#RX_DATA_BITS
MOV @R0,A
MOV R0,#TX_DATA_BITS
MOV @R0,B

T_STOP_BITS:

MOVX A,@DPTR ;SET # OF STOP BITS
INC DPTR

K1:

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
ASSEMBLER CODE LISTING

\_E:
  CJNE A, #'E', CHK_N
  MOV A, #03H ; EVEN PARITY
  SJMP SET_PARITY

\_N:
  CJNE A, #'N', CHK_O
  MOV A, #00H ; NO PARITY
  SJMP SET_PARITY

\_O:
  CJNE A, #'O', CO_EXIT
  MOV A, #01H ; ODD PARITY

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

**************************************************************************
ASSEMBLER CODE LISTING

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 WHETHER TRANSMISSION WAS SUCCESSFUL OR NOT.

THIS STATUS BYTE APPLIES TO BOTH COM1 & COM2.

ON ENTRY:

DPTR POINTS TO SOURCE BUFFER (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

PAGE 138
ASSEMBLER CODE LISTING

LCALL CHK_CTS ;CHECK IF DEVICE READY TO ACCEPT DATA
MOV A,TX82_STAT ;CHECK IF CTS ROUTINE TIMED OUT
CJNE A,#OFFH,TX_LP1
SJMP TX_ERR

; D PTR WILL POINT TO COMSCA / COMSCB
TX82_STAT = OFFH CTS ERROR

; LP1:
CLR TR0 ;STOP TIMER_0
MOV TH0,#0 ;USED FOR TIMEOUT LOOP
MOV TL0,#0 ;80 msec DELAY
SETB TR0 ;START TIMER_0

; LP2:
MOVX A,@DPTR ;WAIT FOR TX REG TO BE EMPTY
JNB ACC.2,CHK_TMOUT1
SJMP TX_LP3

; _TMOUT1:
JB TMOUT,TX_ERR
SJMP TX_LP2

; LP3:
CLR TR0 ;STOP TIMER_0
LCALL CHK_CHAN ;LOAD D PTR WITH ADDR OF 82530
MOV A,B ;RESTORE ACC
MOVX @DPTR,A ;WRITE TO 82530
POP DPH ;RESTORE D PTR
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

IT_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

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

CTS LP1:

MOV     TX82_STAT,#0
CJNE    R7,#02,CTS_LP1 ;CH B
MOV     DPTR,#COMSCB ;RO FOR CTS STATUS
SJMP    CTS_LP2

CTS_LP1:

CJNE    R7,#04,EXIT_CHK_CTS ;CH A
MOV     DPTR,#COMSCA

CTS_LP2:

CLR     TR0     ;STOP TIMER_0
MOV     TH0,#0     ;USED FOR TIMEOUT LOOP
MOV     TL0,#0     ; 80 msec DELAY
SETB    TR0     ;START TIMER_0

CTS_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

CHK_CTS_TMOUT:

JB      TMOUT,CTS_ERR
SJMP    CTS_LP3

ERR:

CLR     TMOUT     ;CLEAR TMOUT FLAG
MOV     TX82_STAT,#0FFH ;CT ERROR

CHK_CTS:

CLR     TR0     ;STOP TIMER_0
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

K_CHAN:

CJNE R7,#01,CHK_LP1 ;CH B, COMMAND
MOV DPTR,#COMSCB
SJMP EXIT_CHK_CHAN

K_LP1:

CJNE R7,#02,CHK_LP2 ;CH B, DATA
MOV DPTR,#COMDATB
SJMP EXIT_CHK_CHAN

K_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

EXIT_CHK_CHAN:

RET

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

INITIALIZE VARIABLES FOR 82530 MODULE
***************************************************************************

**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
ASSEMBLER CODE LISTING

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 BUFFER (COM1_TXBUF/COM2_TXBUF)

* R7 = 1 COMSCB ;COMMS STATUS/CONTROL REG. CH B
* 2 COMDATA ;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

PAGE 144
ASSEMBLER CODE LISTING

_82530:

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

LT_82530_LP1:

CLR A
MOVX A,@DPTR ; READ INITIALIZATION CODE
INC DPH
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

PAGE 145
ASSEMBLER CODE LISTING

        MOVX   A,@DPTR
        MOVX   A,@DPTR
        SJMP   INIT82_EXIT

;INIT82_ERR:
        LCALL   CLS1
        MOV     DPTR,#MSG1
        LCALL   CODE_TO_LCD
        MOV     DPTR,#LCD1_DAT_WR
        LCALL   LCD_WR

;INIT82_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

CHAPTER 3
ASSEMBLER CODE LISTING

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

82530:
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

APTER 3

PAGE 147
ASSEMBLER CODE LISTING

```
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
```

PAGE 148
ASSEMBLER CODE LISTING

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

INCLUDE (EQUATES.ASM)

END

~***************************************************** ********************

%1: DB '* 82530 IS NOT * OPERATIONAL!!! ',0
%2: DB ' 82530 IS FULLY OPERATIONAL!!! ',0
%3: DB 'TX REG NOT EMPTY OR CTS NEVER LOW',0
ASSEMBLER CODE LISTING

CRD.ASM

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

RC: DS 01 ;RESULT OF XOR OF ALL BYTES
BUF: DS 45 ;CARD BUFFER 45 BYTES
ACK: 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

INCLUDE (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 
TRANSSION

ON ENTRY:

CRD_BUF CONTAINS THE DATA READ FROM THE MAGNETIC CARD
**ASSEMBLER CODE LISTING**

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

**_DATA_:**

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SETB P1.4</td>
<td>Set bit as an input for clock</td>
</tr>
<tr>
<td>MOV R0,#CRD_BUF</td>
<td></td>
</tr>
<tr>
<td>MOV A,@R0</td>
<td>Check status byte</td>
</tr>
<tr>
<td>CLR P1.3</td>
<td>Inform host data available</td>
</tr>
<tr>
<td>CJNE A,#01,TRANS_LP3</td>
<td></td>
</tr>
<tr>
<td>MOV R3,#38</td>
<td>38 bytes to transfer</td>
</tr>
<tr>
<td>SJMP TRANS_LP1</td>
<td></td>
</tr>
</tbody>
</table>

**_TRANS_LP3_:**

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MOV R3,#01</td>
<td>An error in reading card, trans 1 byte</td>
</tr>
</tbody>
</table>

**_TRANS_LP1_:**

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MOV A,@R0</td>
<td></td>
</tr>
<tr>
<td>MOV R2,#08</td>
<td>8 bits</td>
</tr>
</tbody>
</table>

**_TRANS_LP2_:**

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>JNB P1.4,$</td>
<td>Wait for CLK 'H'</td>
</tr>
<tr>
<td>JB P1.4,$</td>
<td>Wait for CLK 'L'</td>
</tr>
<tr>
<td>RRC A</td>
<td>Shift data into carry flag</td>
</tr>
<tr>
<td>MOV P1.3,C</td>
<td></td>
</tr>
<tr>
<td>DJNZ R2,TRANS_LP2</td>
<td>Serialize data</td>
</tr>
<tr>
<td>INC R0</td>
<td></td>
</tr>
<tr>
<td>DJNZ R3,TRANS_LP1</td>
<td>Next byte to transfer</td>
</tr>
<tr>
<td>SETB P1.3</td>
<td></td>
</tr>
</tbody>
</table>

**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 = OFFH
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

**_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

**_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 NIBBLE

**PAGE 153**
ASSEMBLER CODE LISTING

LOOP1:
CJNE A,#58H,CRD_LOOP9 ;START SENTINEL
MOV LRC,#11 ;START SENTINEL

LOOP2:
MOV R2,#5 ;BITS / BYTE
MOV A,#OFFH
SETB C ;INITIALIZE REGS

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

LOOP5:
JNB ACC.1,CRD_LOOP4 ;IF SET, INC R4
INC R4 ;'1' FOUND
ASSEMBLER CODE LISTING

_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 ;DISPLAYABLE CHAR
    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
ASSEMBLER CODE LISTING

RRC A ;MOV DATA BIT INTO A
DJNZ R2,CRD_LOOP10 ;NEXT BIT
CLR C
RRC A
CLR C
RRC A
CLR C
RRC A
ORL A,#30H ;DISPLAYABLE CHARs
MOV @RO,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 @RO,#E0T ;FOR LCD STRING, END OF STRING
MOV RO,#CRD_BUF
MOV @RO,#01 ;STATUS BYTE 01 = GOOD, 0FFH = BAD
SJMP CRD_EXIT

_ERR:
MOV RO,#CRD_BUF
MOV @RO,#0FFH
SJMP CRD_EXIT

_LOOP3:
INC RO ;NEXT BYTE IN CRD_BUFFER
DJNZ R3,CRD_LOOP1

(EXIT:

CHAPTER 3 PAGE 156
ASSEMBLER CODE LISTING

RET

;END CARD READING ROUTINE

END
ASSEMBLER CODE LISTING

equate.asm

**************************************************************************
MEMORY I/O EQUATES *
**************************************************************************

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

COMMAND_WRITE EQU 8000H ;INSTRUCTION WRITE
STAT EQU 8001H ;READ LCD STATUS BIT, D7
DATA_WRITE EQU 8002H ;DATA BUFFER WRITE
DATA_READ EQU 8003H ;DATA BUFFER READ

INSTRUCTION_WRITE EQU 9000H
READ_LCD_STATUS EQU 9001H ;READ LCD STATUS BIT, D7
DATA_WRITE EQU 9002H ;DATA BUFFER WRITE
DATA_READ EQU 9003H ;DATA BUFFER READ

LOCK_AND_JUMPERS EQU 0A000H

DATA EQU 0B000H
CMD EQU 0B001H

1 EQU 0C000H
2 EQU 0D000H

ZERO EQU 0E000H

PTER 3 PAGE 158
ASSEMBLER CODE LISTING

`Sdb` EQU 0F000H ;COMMS STATUS/CONTROL REG. CH B

`Dtb` EQU 0F001H ;DATA REG. CH B

`Sca` EQU 0F002H ;COMMS STATUS/CONTROL REG. CH A

`Dat` EQU 0F003H ;DATA REG. CH A

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

**CONSTANT DEFINITION EQUATES**

*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
ASSEMBLER CODE LISTING

EQU 15H
EQU 10H ; USED TO PRECEDE CONTROL CODES
EQU 30H
EQU 31H

_DSP EQU 01 ; CLEAR 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
F_1 EQU 080H ; WRITE TO LINE #1
F_2 EQU 0C0H ; WRITE TO LINE #2
INTELLIGENT POINT OF SALE TERMINAL

SCHEMATIC DIAGRAMS
Figure 4-1. Single Board Computer
Figure 4-4. LED & LCD Interface

THE FOLLOWING RESISTORS ARE SITUATED ON THE DISTRIBUTION BOARD

NOTES:
1. - SIGNIFIES INVERSION
2. - OUTPUT SIGNAL
3. - INPUT SIGNAL
4. - BI-DIRECTIONAL SIGNAL

 veto:

 ANKERdata ADS-ANKER DATA SYSTEMS (PTY) LTD

Title: INTELLIGENT TERMINAL (I/O) (H. KEMHN)
Doc. No.: LED AND LCD INTERFACE TERMS SCH

Size:

A: LED AND LCD INTERFACE TERMS SCH

Date: OCTOBER 17, 1991

Sheet: 4 of 10

REV.
NOTES:
1. = SIGNIFIES INVERSION
2. = OUTPUT SIGNAL
3. = INPUT SIGNAL
4. => BI-DIRECTIONAL SIGNAL

NB HAS TO BE HCT04
ANYTHING ELSE
WILL LOAD XTAL
OSC

ANKERDATA ADS-ANKER DATA SYSTEMS (PTY) LTD
Title: INTELLIGENT TERMINAL (IT) (M. KEGAN)
Size: CARD READER & CLK SOURCE TERM4 SCH
Rev: 0
Date: February 28, 1991 Page: 5 of 5
Figure 4-8: Scanner Interface
Figure 4-9: Draw & Rec. Interface

NOTES:
1. ~ SIGNIFIES INVERSION
2. < OUTPUT SIGNAL
3. > INPUT SIGNAL
4. <> BI-DIRECTIONAL SIGNAL

ANKERDATA ADS-ANKER DATA SYSTEMS (PTY) LTD

Title: INTELLIGENT TERMINAL (I/O) (M KEDDN)

Sheet: DRAW AND RCPT SW INTERFACE TERUB SCHEM

Date: OCTOBER 17, 1995

Page 10

SCHEMATIC DIAGRAMS
SINGLE BOARD COMPUTER V3.00

* DENOTES 100n DECOUPLING CAP
INTELLIGENT POINT OF SALE TERMINAL

PARTS LIST
## CHAPTER 5

### PARTS LIST

#### 1. SINGLE BOARD COMPUTER

**SBC VER 3.00 SBC.SCH**

Revised: October 17, 1991

Bill Of Materials

<table>
<thead>
<tr>
<th>Item</th>
<th>Quantity</th>
<th>Reference</th>
<th>Part</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>C2,C6</td>
<td>56pF</td>
</tr>
<tr>
<td>2</td>
<td>7</td>
<td>C3,C7,C8,C9,C10,C13,C15</td>
<td>10uF</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>C4,C14</td>
<td>100nF</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>CN1,CN2</td>
<td>BOARD-BOARD</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>CN3</td>
<td>8 HEADER</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td>D1</td>
<td>1N4148</td>
</tr>
<tr>
<td>7</td>
<td>1</td>
<td>JP1</td>
<td>JP3</td>
</tr>
<tr>
<td>8</td>
<td>2</td>
<td>Q1,Q2</td>
<td>VN1.0KM</td>
</tr>
<tr>
<td>9</td>
<td>1</td>
<td>R1</td>
<td>4K7</td>
</tr>
<tr>
<td>10</td>
<td>1</td>
<td>R2</td>
<td>10K</td>
</tr>
<tr>
<td>11</td>
<td>1</td>
<td>R3</td>
<td>1K</td>
</tr>
<tr>
<td>12</td>
<td>1</td>
<td>U1</td>
<td>8032</td>
</tr>
<tr>
<td>13</td>
<td>1</td>
<td>U2</td>
<td>74HCT373</td>
</tr>
<tr>
<td>14</td>
<td>1</td>
<td>U3</td>
<td>2764/27128</td>
</tr>
<tr>
<td>15</td>
<td>1</td>
<td>U4</td>
<td>64256</td>
</tr>
<tr>
<td>16</td>
<td>1</td>
<td>U5</td>
<td>DS3695</td>
</tr>
<tr>
<td>17</td>
<td>1</td>
<td>U6</td>
<td>MAX232</td>
</tr>
<tr>
<td>18</td>
<td>1</td>
<td>X1</td>
<td>9.216MHz</td>
</tr>
</tbody>
</table>
## 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

<table>
<thead>
<tr>
<th>Item</th>
<th>Quantity</th>
<th>Reference</th>
<th>Part</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>C1</td>
<td>100nF</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>CN1,CN2</td>
<td>BOARD-BOARD</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>U1</td>
<td>20L10(TERM)</td>
</tr>
<tr>
<td>Item</td>
<td>Quantity</td>
<td>Reference</td>
<td>Part</td>
</tr>
<tr>
<td>------</td>
<td>----------</td>
<td>-----------</td>
<td>------------</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>BZ1</td>
<td>BUZZER</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>C4,C21</td>
<td>100nF</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>C22,C26</td>
<td>1uF</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>C23</td>
<td>100 nF</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>CN6</td>
<td>HEADER 5</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td>CN22</td>
<td>HEADER 2</td>
</tr>
<tr>
<td>7</td>
<td>1</td>
<td>D17</td>
<td>1N4148</td>
</tr>
<tr>
<td>8</td>
<td>1</td>
<td>Q5</td>
<td>VN10KM</td>
</tr>
<tr>
<td>9</td>
<td>1</td>
<td>R3</td>
<td>100K</td>
</tr>
<tr>
<td>10</td>
<td>3</td>
<td>RP1,R4,R18</td>
<td>10K</td>
</tr>
<tr>
<td>11</td>
<td>1</td>
<td>R5</td>
<td>1K</td>
</tr>
<tr>
<td>12</td>
<td>1</td>
<td>R17</td>
<td>33R</td>
</tr>
<tr>
<td>13</td>
<td>1</td>
<td>SW1</td>
<td>SW DIP-4</td>
</tr>
<tr>
<td>14</td>
<td>1</td>
<td>U4</td>
<td>74HCT245</td>
</tr>
<tr>
<td>15</td>
<td>1</td>
<td>U14</td>
<td>LM556</td>
</tr>
</tbody>
</table>
# 2.3 LED AND LCD INTERFACE TERM3.SCH

## Bill Of Materials

<table>
<thead>
<tr>
<th>Item</th>
<th>Quantity</th>
<th>Reference</th>
<th>Part</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>C2, C3</td>
<td>100nF</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>CN3, CN4</td>
<td>HEADER 10</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>CN13, CN14</td>
<td>HEADER 15</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>CN15, CN21</td>
<td>HEADER 3</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>RX1</td>
<td>160R</td>
</tr>
<tr>
<td>6</td>
<td>2</td>
<td>RX2, RX3</td>
<td>1.5R</td>
</tr>
<tr>
<td>7</td>
<td>2</td>
<td>U2, U3</td>
<td>74HCT374</td>
</tr>
<tr>
<td>8</td>
<td>1</td>
<td>U13</td>
<td>74HCT04</td>
</tr>
</tbody>
</table>
# Parts List

## 2.4 CRD Reader & CLK Source Term4.SCH

### Bill Of Materials

<table>
<thead>
<tr>
<th>Item</th>
<th>Quantity</th>
<th>Reference</th>
<th>Part</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>C8, C9</td>
<td>22pF</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>C10, C11</td>
<td>100nF</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>CN5</td>
<td>HEADER 22</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>CN10</td>
<td>HEADER 5</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>U8</td>
<td>8751</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td>U9</td>
<td>74HCT393</td>
</tr>
<tr>
<td>7</td>
<td>1</td>
<td>U13</td>
<td>74HCT04</td>
</tr>
<tr>
<td>8</td>
<td>1</td>
<td>XTAL1</td>
<td>7.3728MHz</td>
</tr>
</tbody>
</table>
# PARTS LIST

## 2.5 UNIVERSAL KYB INTERFACE TERMS.SCH

### Bill Of Materials

<table>
<thead>
<tr>
<th>Item</th>
<th>Quantity</th>
<th>Reference</th>
<th>Part</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3</td>
<td>C5,C6,C7</td>
<td>100nF</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>CN7,CN8,CN9</td>
<td>HEADER 8</td>
</tr>
<tr>
<td>3</td>
<td>12</td>
<td>D1,D4,D5,D6,D7,D8,D10, D11,D12,D13,D14,D15</td>
<td>DIODE</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>D1,D8,D9,D16</td>
<td>1N4148</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>RP2</td>
<td>10K</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td>U5</td>
<td>8279</td>
</tr>
<tr>
<td>7</td>
<td>1</td>
<td>U6</td>
<td>74HCT30-ADS</td>
</tr>
<tr>
<td>8</td>
<td>1</td>
<td>U7</td>
<td>74HCT138</td>
</tr>
</tbody>
</table>
## 2.6 Dual RS232 Interface

### Bill Of Materials

<table>
<thead>
<tr>
<th>Item</th>
<th>Quantity</th>
<th>Reference</th>
<th>Part</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>C12</td>
<td>100nF</td>
</tr>
<tr>
<td>2</td>
<td>8</td>
<td>C13, C14, C15, C16, C17, C18, C19, C20</td>
<td>10uF</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>CN11, CN12</td>
<td>HEADER 10</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>R1, R2</td>
<td>10K</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>U10</td>
<td>82530</td>
</tr>
<tr>
<td>6</td>
<td>2</td>
<td>U11, U12</td>
<td>MAX232</td>
</tr>
</tbody>
</table>
## Bill Of Materials

<table>
<thead>
<tr>
<th>Item</th>
<th>Quantity</th>
<th>Reference</th>
<th>Part</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>C25,C27</td>
<td>220pF</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>CN16,CN23</td>
<td>HEADER 5X2</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>R6</td>
<td>10K</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>R7,R20</td>
<td>220R</td>
</tr>
<tr>
<td>5</td>
<td>2</td>
<td>R8,R19</td>
<td>120R</td>
</tr>
<tr>
<td>6</td>
<td>4</td>
<td>R9,R10,R21,R22</td>
<td>2R7 .5W</td>
</tr>
<tr>
<td>7</td>
<td>2</td>
<td>U15,U17</td>
<td>6N136</td>
</tr>
<tr>
<td>8</td>
<td>2</td>
<td>U16,U18</td>
<td>SN75452</td>
</tr>
</tbody>
</table>
## PARTS LIST

### 2.8 DRAW AND RCPT SW INTERFACE TERM8.SCH

**Bill Of Materials**

<table>
<thead>
<tr>
<th>Item</th>
<th>Quantity</th>
<th>Reference</th>
<th>Part</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>CN17</td>
<td>HEADER 2</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>CN18,CN19,CN20</td>
<td>HEADER 4</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>D18,D19</td>
<td>1N4007</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>Q1,Q2</td>
<td>MPSU06</td>
</tr>
<tr>
<td>5</td>
<td>2</td>
<td>Q3,Q4</td>
<td>BC307</td>
</tr>
<tr>
<td>6</td>
<td>2</td>
<td>R11,R14</td>
<td>22K</td>
</tr>
<tr>
<td>7</td>
<td>2</td>
<td>R12,R15</td>
<td>560R</td>
</tr>
<tr>
<td>8</td>
<td>2</td>
<td>R13,R16</td>
<td>10K</td>
</tr>
</tbody>
</table>

October 17, 1991 19:18:38
### Bill Of Materials

<table>
<thead>
<tr>
<th>Item</th>
<th>Quantity</th>
<th>Reference</th>
<th>Part</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>U1</td>
<td>74HCT138</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>U2</td>
<td>74HCT08</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>U3,U4</td>
<td>74HCT32</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>U5</td>
<td>74HCT04</td>
</tr>
</tbody>
</table>

October 17, 1991 19:18:50
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

IBM

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 SOT, 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