USING A ONE-CHIP MICROCOMPUTER
TO CONTROL AN
AUTOMATED WAREHOUSE MODEL

A Thesis Presented to
The Faculty of the College of Engineering and Technology
Ohio University

In Partial Fulfillment
of the Requirements for the Degree
Master of Science

by
Steven Ricca
June, 1988
Many thanks to my parents, Vincent and Carole, whose continued support contributed immeasurably. I am especially indebted to my wife, Christina, who typed the manuscript and software, and put up with many revisions.

It was Dr. Harold Klock's classroom instruction, which led to my first "handshake" with a microcomputer. As my thesis advisor, his instructions regarding the manuscript were appreciated.

A special thanks to Jim Mc Knight, for his machining and various shop work, that enabled the model to be a success.
# TABLE of CONTENTS

I. INTRODUCTION ................................................. 1

II. OVERVIEW ....................................................... 4

III. SELECTION of COMPONENTS ...................................... 14
   Electrical Components ........................................ 14
   Mechanical Components ....................................... 21

IV. ELECTRICAL DESIGN ............................................ 26

V. MECHANICAL DESIGN ............................................. 46

VI. SOFTWARE DESIGN ............................................... 51

VII. HOST TERMINAL ................................................. 132

VIII. TESTING of MODEL ........................................... 149

IX. OPERATIONAL INSTRUCTIONS ................................... 153
   Maintenance and Transportation ............................... 160

X. SUMMARY .......................................................... 164
   Conclusion ..................................................... 164
   Enhancements and Modifications ............................. 165

BIBLIOGRAPHY ...................................................... 173

APPENDIXES .......................................................... 176

A Complete Software Listing .................................... 176
B Schematic Diagrams ............................................. 191
C Wiring Reference ................................................ 196
D Suppliers Reference ............................................. 204
## TABLE of ILLUSTRATIONS

<table>
<thead>
<tr>
<th>Title</th>
<th>Figure #</th>
<th>Page #</th>
</tr>
</thead>
<tbody>
<tr>
<td>Column and Row Layout</td>
<td>2-1</td>
<td>6</td>
</tr>
<tr>
<td>X &amp; Y Movements</td>
<td>2-2</td>
<td>7</td>
</tr>
<tr>
<td>Pneumatic Piston Lowered</td>
<td>2-3</td>
<td>9</td>
</tr>
<tr>
<td>Pneumatic Piston Lifted</td>
<td>2-4</td>
<td>9</td>
</tr>
<tr>
<td>Typical Movement Sequence</td>
<td>2-5</td>
<td>10</td>
</tr>
<tr>
<td>Electrical System Diagram</td>
<td>2-6</td>
<td>12</td>
</tr>
<tr>
<td>Electrical System Block Diagram</td>
<td>4-1</td>
<td>27</td>
</tr>
<tr>
<td>Microcomputer Block Diagram</td>
<td>4-2</td>
<td>28</td>
</tr>
<tr>
<td>Stepper Motor Drive Circuit</td>
<td>4-3</td>
<td>35</td>
</tr>
<tr>
<td>Output Interfacing Circuit</td>
<td>4-4</td>
<td>39</td>
</tr>
<tr>
<td>Main Routines Flowchart</td>
<td>6-1</td>
<td>54</td>
</tr>
<tr>
<td>Interrupt Routines Flowchart</td>
<td>6-2</td>
<td>55</td>
</tr>
<tr>
<td>Assembly Initializations Flowchart</td>
<td>6-3</td>
<td>59</td>
</tr>
<tr>
<td>Assembly Initializations Listing</td>
<td>6-4</td>
<td>61</td>
</tr>
<tr>
<td>System Memory Map</td>
<td>6-5</td>
<td>65</td>
</tr>
<tr>
<td>System Initialization Flowchart</td>
<td>6-6</td>
<td>68</td>
</tr>
<tr>
<td>System Initialization Listing</td>
<td>6-7</td>
<td>69</td>
</tr>
<tr>
<td>I/O Port Configurations</td>
<td>6-8</td>
<td>72</td>
</tr>
<tr>
<td>Demonstration Loader Flowchart</td>
<td>6-9</td>
<td>73</td>
</tr>
<tr>
<td>Demonstration Loader Listing</td>
<td>6-10</td>
<td>74</td>
</tr>
<tr>
<td>Calibration Routine Flowchart</td>
<td>6-11</td>
<td>76</td>
</tr>
<tr>
<td>Calibration Routine Listing</td>
<td>6-12</td>
<td>77</td>
</tr>
<tr>
<td>Premove Routine Flowchart &amp; Listing</td>
<td>6-13</td>
<td>79</td>
</tr>
<tr>
<td>Motor Stepping Routine Flowchart</td>
<td>6-14</td>
<td>81</td>
</tr>
<tr>
<td>Motor Stepping Routine Listing</td>
<td>6-15</td>
<td>83</td>
</tr>
<tr>
<td>Limit Switch Routine Flowchart</td>
<td>6-16</td>
<td>88</td>
</tr>
<tr>
<td>Limit Switch Routine Listing</td>
<td>6-17</td>
<td>91</td>
</tr>
<tr>
<td>Handshake Routine</td>
<td>6-18</td>
<td>93</td>
</tr>
<tr>
<td>Z-Axis Routine Flowchart</td>
<td>6-19</td>
<td>95</td>
</tr>
<tr>
<td>Z-Axis Routine Listing</td>
<td>6-20</td>
<td>97</td>
</tr>
<tr>
<td>Snooze Routine Flowchart</td>
<td>6-21</td>
<td>100</td>
</tr>
<tr>
<td>Snooze Routine Listing</td>
<td>6-22</td>
<td>101</td>
</tr>
<tr>
<td>Compressed Command Format</td>
<td>6-23</td>
<td>103</td>
</tr>
<tr>
<td>Pop Queue/Expansion Flowchart</td>
<td>6-24</td>
<td>105</td>
</tr>
<tr>
<td>Pop Queue/Expansion Listing</td>
<td>6-25</td>
<td>107</td>
</tr>
<tr>
<td>Serial Comm. Interrupt Flowchart</td>
<td>6-26</td>
<td>110</td>
</tr>
<tr>
<td>Serial Comm. Interrupt Listing</td>
<td>6-27</td>
<td>116</td>
</tr>
<tr>
<td>Manual Halt Interrupt Flowchart</td>
<td>6-28</td>
<td>122</td>
</tr>
<tr>
<td>Halt &amp; Fatal Error Interrupt Listings</td>
<td>6-29</td>
<td>124</td>
</tr>
<tr>
<td>Fatal Error Interrupt Flowchart</td>
<td>6-30</td>
<td>127</td>
</tr>
<tr>
<td>OU Demo Commands Listing</td>
<td>6-31</td>
<td>129</td>
</tr>
<tr>
<td>Title</td>
<td>Figure #</td>
<td>Page #</td>
</tr>
<tr>
<td>-------------------------------------------</td>
<td>----------</td>
<td>--------</td>
</tr>
<tr>
<td>RS-232C Female Connector</td>
<td>7-1</td>
<td>133</td>
</tr>
<tr>
<td>Self-Handshake Scheme</td>
<td>7-2</td>
<td>135</td>
</tr>
<tr>
<td>Serial Connections in this System</td>
<td>7-3</td>
<td>138</td>
</tr>
<tr>
<td>Flow Control Handshake Flowchart</td>
<td>7-4</td>
<td>141</td>
</tr>
<tr>
<td>Flow Control Handshake Elec. Conn.</td>
<td>7-5</td>
<td>141</td>
</tr>
<tr>
<td>Serial Stop Bit Timing</td>
<td>7-6</td>
<td>143</td>
</tr>
<tr>
<td>User Operating Instructions Flowchart</td>
<td>9-1</td>
<td>154</td>
</tr>
<tr>
<td>Macro-command Summary</td>
<td>9-2</td>
<td>155</td>
</tr>
<tr>
<td>Macro-command Examples</td>
<td>9-3</td>
<td>158</td>
</tr>
</tbody>
</table>
I. INTRODUCTION

Advances in VLSI technology have produced an extremely versatile family of components known as One-Chip Microcomputers. A spin-off from microprocessor technology, these powerful devices contain an entire computer system on one chip. To illustrate many features and capabilities of these devices, a fairly complex control system has been designed - an automated warehouse model.

Automated control systems are fast becoming the standard method of obtaining accurate, efficient, and reliable processes. The speed and consistency of solid-state controls often results in dramatically lowered costs by eliminating many mechanical mechanisms. In addition, design modifications can be easily accommodated by modifying software instead of hardware.

A fully functional model was constructed for testing management strategies and physically illustrating the microcomputer's control functions. The warehouse model is a slave to a user-operated host terminal or computer, and its design exemplifies the mechanical simplicity, flexibility, accuracy, and low cost that can be benefits of a fully computerized system. High level macrocommands are entered at the host terminal, then fully
digested and carried out by the automated warehouse microcomputer.

The system that was designed takes full advantage of the one-chip microcomputer; in fact, the only additional electronic components needed are buffers for interfacing to the model. The custom designed microcomputer software manages all serial communications, computing, and control functions. Open-loop control of two stepper motors is achieved directly by the microcomputer's software, without the need for external stepper motor control chips. Up to one hundred commands may be stored by the microcomputer for sequential execution.

The remainder of this thesis is organized into nine chapters and an Appendix. In Chapter Two, an overview of the problems to solve, and a brief mechanical and electrical description will be given. Chapter Three discusses the selection of electrical, and mechanical components. In Chapters Four and Five, the specific electrical, and mechanical designs are brought forth. Chapter Six contains an explanation of the microcomputer software, along with accompanying flowcharts and the compiled program listing. The serial communications standards, and host terminal or computer requirements are set forth in Chapter Seven. Chapter Eight contains several test procedures performed by the model, and their results. Operational procedures required by the user are
outlined in Chapter Nine. The last chapter summarizes the overall accomplishments of this thesis, and discusses possible enhancements and modifications. Finally, the appendices contain schematics and wiring charts, a suppliers' reference, and additional illustrations.
The system that has been developed is an independent computer operating a mechanical warehouse model. The model represents a warehouse that operates on identical containers. The entire model is controlled and monitored by a one-chip microcomputer, which is a slave to the control computer (host). This section provides an overview of the main functions the model will perform, as well as, a brief mechanical and electrical description.

The movement of a specific container (block) starts by receiving a high level macro-command from a host terminal. The command is then executed with no additional effort from the host. As many as one hundred commands may be stored at any given time, and they will be sequentially executed on a first-in-first-out basis. Each command is decoded and physically performed by driving stepper motors to the desired location (column and row). A block is picked up or put down at that location, and the move is complete. The next command can then be initiated from that location.

The macro-command set is based on single characters that may be grouped together to achieve a particular action. The most commonly used commands are the "pick up" and "put down" variety. The proper usage of these
two commands requires the column and row numbers to follow, which provide the location. An example might be as follows: U 1 2 D 3 4. These two commands would pick Up the block at column 1, row 2, and put it Down at column 3, row 4. Calibration to the nearest corner is possible by sending the single character "C". Several macro-commands also exist to decrease or increase the physical speed (< or >), return to normal speed (=), clear all stored commands (cntrl-x), halt system (%), and resume from halt (^). The U, D, and C commands are placed in the queue for sequential execution, while the others take effect immediately.

Mechanically, the warehouse model demonstrates the control system functions, and some of the problems that can be involved in such systems. The roughly four-foot by three-foot model is basically a frame built above the warehouse area. The frame, with column and row number assignments is shown in figure 2-1. Similar to the first quadrant of a coordinate plane, the lower, left-most corner is column 0, row 0. A total of eight columns and rows, numbered 0-7 follow outward from the origin. Also, note the limit switch positions, which are the extreme physical positions in the warehouse area.

A sled rides on two parallel edges (rails) of this frame. A stepper motor draws a chain, pulling this sled back and forth, thus generating an X-axis motion (figure
There are seven columns (X) and seven rows (Y). The limits define extreme positions. Blocks are shown at (X,Y): (0,0), (1,4), (1,7), (6,6), and (5,2).
**FIGURE 2-2 X and Y Movements**

Inside this "X-sled" is another sled riding on the rails formed by the X-sled. This inner sled is similarly pulled by another chain, connected to a stepper motor that travels with the X-sled. Collectively, the X and Y-sleds are moved into position at the column and row of interest. A pneumatic piston is then lowered from the Y-sled onto the top block (figure 2-3). Next, an electromagnet inside the piston is activated and "locks onto" the block. The piston is retracted and the Y-sled proceeds to execute the next command (figure 2-4). The typical time required to complete a command is under ten seconds. Blocks may be stacked up to three high.

A typical sequence which relates the controlling host terminal and command set to the warehouse model is shown in figure 2-5. The X and Y sleds could be in any arbitrary location before the sequence. Three blocks are to be moved (A, B, and C). Their destinations are shown by the dashed boxes. The movements of the piston unit are numbered for illustration. After the blocks are moved, the "C" macro-command, (move number 7), causes a calibration to the nearest corner. If twenty-five seconds elapse with no additional inputs, then the system will go into a low-power mode, at this location, until the next command arrives.

All completely original assembly language software was written for this specific application. The
FIGURE 2-3 Pneumatic Piston Lowered
Piston is lowered on top of a block beneath it.

FIGURE 2-4 Pneumatic Piston Lifted
Vacuum is supplied to piston causing it to lift. Internal electromagnet energizes, lifting block also.
FIGURE 2-5 Typical Movement Sequence

System starts at arbitrary position (6,4), sequentially executes commands, and ends with a calibration command. Arrows show path of piston assembly and move number. Solid blocks are moved to location shown by dashes.
microcomputer and its software are not only responsible for communications with a host terminal and storage of these commands, but also the control of all motor, piston, and electromagnet actions. The four-phase sequences required to position the X and Y-stepper motors are computed, on-the-fly, between motor steps. These calculations allow both motors to move simultaneously. In addition, limit switches are checked after every motor-step to ensure the accuracy of this open-loop system.

Electrically, the system is very straightforward, as shown in figure 2-6. The microcomputer controls many identically configured, power handling circuits. All microcomputer outputs drive TTL buffers. Each buffer drives an optical isolator, providing a non-electrical connection to the larger current handling power transistors. The microcomputer, buffers, and their voltage regulator are mounted on a 44-pin edge connector card. Two other cards contain nearly identical configurations of optoisolator/power transistor combinations. All three cards and the two, system power supplies are housed in an electrical rack, allowing boards to be easily inter-connected.

In order to construct the warehouse model, and have it function as previously discussed, several decisions were made. The next chapter will describe the selection
FIGURE 2-6 Electrical System Diagram
of methods and components that make up the warehouse model.
III Selection of Electrical Components

One of the key electrical goals of the project was to use a one-chip microcomputer, along with serial communications, to directly control stepper motors without much support hardware, especially stepper control chips. Using a solid-state approach as a guide, the components were selected based on function, reliability, cost, and availability.

First, a one-chip microcomputer had to be chosen [1]. Several preferences were known from the onset. Serial communications interface, internal RAM, and bit-manipulation capabilities were the most obvious. Due to more familiarity with the 6800 family assembly language code, the choice was further narrowed. It was decided that an eight-bit unsigned architecture would suffice, as would, at least two hardware interrupts. It also seemed that two-thousand bytes of EPROM, and sixteen input/output ports would be the minimum system requirements.

Based on availability and cost, this left two candidates. Both the Motorola MC68701 and Hitachi HD63P01M1 microcomputer units met the prescribed minimum requirements [2], [3]. In fact, they are nearly identical units. Both use an identical, enhanced CMOS version 6801 assembly language code. Both units have a
serial communications interface, 16-bit programmable timer, single chip or expanded modes of operation, 128 bytes of internal RAM, and 29 parallel input/output lines. They even have virtually identical pinouts and execution times.

However, the Hitachi unit provided the deciding plus. This microcomputer can accept up to 8K bytes of EPROM directly on top of the chip. There is a socket on top (piggyback) that allows for simple EPROM removal and programming by a standard EPROM programmer. In contrast, the Motorola's internal 2K EPROM is non-removable. To reprogram the internal EPROM, the user is required to make a separate programmer for this specific unit. The Hitachi's convenience dominated, and a standard 8K EPROM with less than 250 nanoseconds access time was selected (2764-150) in accordance with the Hitachi literature [4].

Communicating with a host terminal over serial lines was another "given" in the system. Serial communications are slower, but speed was not really a factor because so little communicating would be required. Serial methods are more noise-immune, and hence longer distances between devices are possible. The "standard" serial interface is the RS-232 connection, and an electrical buffer was needed for the microcomputer unit to be successfully connected. Primarily for noise-rejection reasons, the RS-232 voltage standards call for a -15 to +15 volts
signal transmission range. Usually, an input and output buffer are utilized. Traditionally, these buffers must have voltage supplies of -12 volts and +12 volts each. Accordingly, two voltage supplies with their associated transformer and voltage regulation circuits would have to be developed for this one application. As an alternative, an ingenious chip has been produced that operates from a single +5 volt supply, and internally generates +10 volt and -10 volt supplies [5]. This single supply significantly simplifies the system design while meeting all the EIA RS-232C specifications. Although the chip (Maxim 232) has a higher cost, than simple buffers, overall it is less costly than producing voltages other than +5 volts.

Output buffers for the microcomputer's other output ports were needed since the TTL level, output current was expected to exceed the 1.6 mA maximum imposed by the microcomputer specifications. A "healthy" margin of safety was desired for these buffers since it was uncertain how much current they might need to handle. Knowing that at least 14 buffers would be needed, a concern for layout space was justified. An adequate solution was found in two 74LS245 octal transceiver chips. Each chip can handle eight outputs, with driving and sinking currents exceeding 15 mA for each buffer. An additional bonus was the layout. The buffers were all
in-line, which greatly simplified on-board connections [6].

Noise spikes associated with switching large currents, especially inductive loads, can unpredictably affect logic circuitry. To combat noise problems and allow the use of totally separated voltage supplies, the control signals had to be electrically separated from the larger currents. Optical isolators were the natural choice. A quad optical isolator package (PS2401A-4NEC) was chosen for size and excellent current transfer ratio (80%). This chip affords an isolation voltage of 5000 volts, and its NPN open-collector design is typical [7]. Also, a few single optoisolators (GE 4N28) were used because they were on hand.

Because the NPN output transistor in an optoisolator can not be expected to sink much current, it was clear that a high-gain switching device would be needed to drive large currents. Typically, bipolar transistors exhibit a larger voltage drop across their collector-emitter junction at saturation than a MOS device does across its drain-source junction [8]. This means that to supply five volts to the motors, a voltage supply of 1.5 volts greater, per series power transistor, would have to be allowed. This fact actually worked as an advantage later in setting up the power supply. So, bipolar
transistor technology was selected based on availability and cost.

Once the bipolar decision had been made, the natural choice was for power Darlington transistors. Because current gains of over 1000 are typical, and power demands would be relatively low (5 W average), it was decided that the TO-220 style case would be ideal [9]. Specifications for the TIP106 and TIP101 Power Darlington transistors indicated an average current handling ability of 5A, and their low cost was hard to overlook.

With most of the control circuitry selected, proper stepping motors would need to be selected. Important factors included: accuracy, weight, power demands, low speed torque, and cost. A stepping angle of 1.8 degrees results in 200 steps per revolution and is typical of these motors. Of course, most motors could be half-stepped, if needed, to obtain 0.9 degrees of rotation per step. Because the Y-axis motor would be traveling on the X-sled, its weight and size had to be minimized. It was desirable to find a motor with enough low-speed torque to avoid ramping of the stepping speed. If a motor has enough torque, the job can be done with sufficient speed, and the motor can stay in the "error free stop and start" zone (EFSS) without the need for ramping. After investigating the torque curves for several motors, one
was located that met the requirements, and was also fairly inexpensive [10], [11].

Each phase of this four-phase stepping motor is rated for one ampere at 5 volts. Experimentation proved that its rated torque (35 inch/pounds), and electrical characteristics would suffice. The choice to drive these motors with a unipolar connection was simply based on the fact that it was powerful enough, and drew less current than a bipolar connection, which could always be wired if even more torque was required.

Although excellent accuracy was expected from the motors, there would be a need for limit switches to report if an out-of-bounds condition had occurred. These limit switches could also be used to calibrate the system on start-up. Many switches and styles existed, but it was clear that the proper switch must be sensitive. Such a switch must also be able to recognize a closure with a very small differential travel. In the interest of a solid-state system, the choice was narrowed to an optical device, or Hall Effect sensor.

Several optical type, break-beam devices were experimented with, but they were not complete packages; and hence, required cumbersome alignments. In contrast, a Hall Effect Micro Switch from Honeywell (37L31B-12) was a complete unit built into the housing of a common snap action switch. This unit has a low operating force,
typically two-tenths of an ounce with a lever. Also, its differential travel was rated at fifteen-thousandths of an inch, which would be less than a motor step. Another benefit of Hall effect sensors is the inherent hysteresis effect. This effect ensures bounceless operation since its "operate" point is different from its "release" point [12]. Four of these Hall effect switches were obtained for a very reasonable cost.

After most of the electrical components had been determined, a rough idea of power supply requirements could be assembled. For noise isolation purposes, two separate supplies would be needed. The microcomputer and associated electronics would require a well regulated five-volt supply at a minimum of 300 mA. A one-ampere, 6.3 volt filament transformer was selected along with a full wave bridge rectifier and suitable filter capacitor. It is usually best to produce the voltage regulation as physically near to the devices that will need it. For this reason, a one-chip, positive five-volt regulator (LM7805-T) was selected to be incorporated on-board with the microprocessor.

The other, more power-consuming components would require a heavy-duty power supply. Experimentation showed that the stepper motors each required 1 ampere while running, and 1.5 amperes total holding current. Adding the electromagnet, vacuum solenoid, and both
motors' currents produced a worst case requirement of four amperes. None of these inductive loads required much voltage regulation, so it was decided to use a half-wave bridge and large capacitor approach. A 12.6 volts, center-tapped transformer, rated at better than 6 amperes continuous duty was used to provide an ample margin of safety.

Integrating these electrical components to work with each other is discussed in Chapter Four. More details regarding sources for these components can be found in Appendix D. Next, the choices contributing to the mechanical system development are brought forth.

Selection of mechanical components

The selection of mechanical components was quite often based on three factors: low cost, reliability, and availability. Since the project's focus was on the electrical control system, the mechanical aspects were approached in the simplest, and most straightforward manner that would illustrate a functioning system. The following discussion will explain the selection of certain mechanical processes and individual components, that together make up the mechanical system.
Building a frame above the warehouse area was chosen as the easiest method to access every row and column location. A minimum amount of machining would be required since only two edges (rails) demanded accurate positioning. Because this frame would be stationary, and provide support for most of the machinery, weight was not a consideration. Instead, strength was the primary interest. Angle iron (3/4 x 3/4 x 1/8") was selected since it exhibited no deflection under load, had a low cost, and was readily available. An edge of this angle iron would provide the rail needed for other parts to roll on. Supported near the four corners, this frame could be mounted on a suitable base. External-grade 3/4" plywood, with a reinforcing frame underneath, became that base, providing a stable surface in which to mount devices; as well as, simulating the floor of the warehouse.

Transferring the motors' rotation into a linear pulling motion of the sleds required a drive system decision. Several choices existed; such as, belts and pulleys, worm gear/screw drives, and miniature chains. The need for a positive, non-slip system precluded the belt and pulley combination. After some calculations and pricing, the worm gear and lead-screw pair showed several deficiencies. Among them were the weight, price, and alignment problems. An even bigger problem
was that of speed. The motors would have to exceed 1200 rpm if standard, 8 threads per-inch, threaded rod were to be used. This translates to over 4000 steps per second if a stepping motor would directly drive the screw. The best choice clearly became the miniature chain set-up.

A polyurethane coated drive chain and matching sprocket gears were located and proved to be ideal. Positive drive, no lubrication, silent, zero backlash, lightweight and very strong, are many of the features to be expected with this type of drive. One-quarter inch drive shafts were selected to keep rotational inertia low.

The greatest frictional forces are present just before a system at rest starts to move. To minimize these forces, small ball bearings were selected for all rotating parts. They are all identical and share a one-quarter inch shaft size. Use of the bearings allows friction to be nearly neglected and increases reliability.

A linear actuator had to be developed - capable of dropping onto a block from above, while exhibiting suitable accuracy, speed, power, and weight. No acceptable motorized linear actuators could be located. Most were the lead-screw type, and it was decided that the X- and Y-accelerations would cause misalignments.
To this end, a pneumatic piston type actuator was conceived. Although this meant the need for a separate vacuum source and supply line, there were several advantages. A pneumatic piston could be fairly easily fabricated, and mechanical simplicity was a goal. Another plus was the inherent symmetry of such a device. Since the piston would always be pulled up, inside its cylinder walls when X- and Y-movements are occurring, there would be nothing to misalign. Once the piston is allowed to drop, it will always rest on the highest object, in this case the top block. Positional bookkeeping would not be necessary for this device. The case of not dropping far enough, or worse yet, attempting to drive the actuator beyond its destination could not occur.

To actually "capture" the block of interest and transport it somewhere, required either a mechanical "hand", or a simple electromagnet-ferrous target scheme. The latter was chosen for ease of construction and the simple fact that it would fit so well inside the piston. Because only the top block in a given stack would be operated on, the blocks would all need a sheet iron surface on top to be picked up by the electromagnet.

A vacuum source was selected based primarily on low cost and its ability to supply ample suction to the piston assembly. It was obvious that some sort of a
silencer would be needed to quiet a two-stage blower of this type. It was also clear that excessive vibration would not be acceptable.

Finally, a suitable enclosure for the electronic circuitry was necessary. Such an enclosure would allow for easy installation of component cards, protection from accidental damage, and allow proper cooling. Although metal would make a better RMI shield, a wood enclosure was chosen for ease of construction, and protection from electrical shorting.

Now that the basic processes and materials had been determined, the specific design of the system could begin. More details regarding the mechanical components and their design can be found in Chapter Five and Appendix D. In the next chapter, the specific designs of the electrical system will be discussed.
While proceeding with the electrical design of the system, several important factors were kept in mind. First, the speed of the components was not critical. Because the mechanical system would be the limiting factor, designing for maximum electrical speed was not much of a concern. Instead, a premium was placed on reducing the generation of spurious electrical noise, and on a design that was insensitive to any such noise that would propagate into the system. Also, reliability had to be present in all phases of the system. To ensure this, a generally conservative approach was taken to allow for worst-case situations. The following discussion will show the electrical design considerations involved in producing this system (figure 4-1).

Since the one-chip microcomputer unit has many built in features (figure 4-2), the system design was greatly simplified. However, several important decisions were made in utilizing this chip. First, the microcomputer's internal-clock speed had to be determined. The chip specifications called for a parallel resonant fundamental crystal (AT-cut). The chip internally divides this crystal frequency by four to derive the system clock speed. The maximum clock
FIGURE 4-1 Electrical System Block Diagram
FIGURE 4-2 Microcomputer Block Diagram
speed is one megahertz (M1 version), while the minimum allowed is one hundred kilohertz. These clock requirements are probably due to the CMOS construction of the chip, and more specifically, the use of dynamic RAM internally. More than a dozen crystal frequencies are available in this range (400 KH to 4 MHz). However, to serially communicate successfully with a host terminal, a common baud rate would have to be observed. The serial communication's interface divides the system clock to derive its basis for transfer rates. Consulting Table 6, of the Hitachi literature package [4], showed that a crystal frequency of 2.4576 MHz was the only acceptable choice. At this crystal frequency, the system clock would be 614 KHz, and transfer rates of 38,400, 4,800, 600, or 150 baud would be possible.

According to Hitachi specifications, the internal oscillator requires a minimum of twenty-milliseconds to stabilize itself. After a power-on condition, the power system should also be allowed to stabilize before any larger current demands are required. To ensure the system was indeed stable, a generous time constant was chosen for the reset circuitry. This simple RC network consisted of a 39 mF capacitor and a 510 ohm resistor. The 510 ohm pull-up resistor was selected first. Its rather low ohmic value
was selected to ensure a quick rise time, and reduce the chance of noise triggering the reset.

Directly following a system reset, the microcomputer operation mode must be determined by wiring the 8, 9, and 10 pins externally. The HD63P01 can operated in three basic modes: Single Chip, Expanded Multiplexed, and Expanded Non-Multiplexed mode. To select the mode, a circuit involving a multiplexer and separate mode control switches is recommended in the Hitachi literature [4]. However, since we were only interested in the single chip mode, a simpler circuit was devised. By hardwiring the 8, 9, 10 pins high through 1K resistors, mode seven (single chip) can be selected. This simplification has a drawback: the three, now dedicated, pins can no longer be used as input or output ports. In fact, care must be taken to insure the software will not output to these pins. If such an accident would occur, the pull-up resistors should avert damage to the chip. Because the single chip mode has 29 input/output ports, the loss of these three causes no problem in this application.

With twenty-six input/output pins remaining, it was decided that eight inputs would suffice to check for various conditions about the warehouse. For convenience, all eight were taken from one port of the microcomputer (pins 13 – 20). Since these input lines
were likely to be electrically noisy, a pull-up resistor of 510 ohms was selected for each of these pins. Although these resistors would require input devices to sink more current than, for instance, a 10K ohm pull-up, the maximum of two-milliamperes each required seemed easily achievable. These pull-up resistors not only provide a better noise immunity, and faster rise and fall times; but also, minimize the effects of small leakage currents from the attached devices.

Because serial communications were necessary, two additional pins of the microcomputer had to be dedicated. The onboard serial communications interface is connected such that the receiver and transmitter sections are active through pins 11 and 12 respectively. These TTL-level signals must be buffered to interface with the EIA RS-232C electrical specifications. An RS-232C transmitter generates a voltage between +5 and 15 volts to represent a signal space (low data bit), and a voltage between -5 and -15 for a signal mark (high data bit). An RS-232C receiver recognizes voltages above +3 volts as spaces and voltages below -3 volts as marks. The transition area between these two regions is limited to occur for only 4% of a bit period. The total line capacitance is limited to 2500 picofarads, and any RS-232C interface should be able to survive continuous short circuiting to ground [13].
As previously discussed in Chapter 3, the Maxim, +5 volt powered, dual RS-232 transmitter/receiver chip was selected to provide buffering. External capacitors C1 and C2 are used by the chip's charge pumps to generate the increased voltage range (10V to -10V). As suggested in the Maxim literature [5], the values of C1 and C2 were changed to 47 microfarads instead of 22 microfarads shown in the test circuit. This change lowers the output impedance to about 440 ohms. External capacitors C3 and C4 were also increased from 22 microfarads to 47 microfarads in order to lower the 16 KHz ripple, on the RS-232 outputs, caused by the charge pumps. The only negative result of increasing these four capacitors is a slight increase in power consumption.

Because a regulated +5 volt power supply was needed to drive the logic circuitry, a simple one-chip regulator could be used. For this regulator (7805-T), a TO-220 style package was appropriate since current demand was obvious to be less than one ampere. These regulators typically have a voltage drop of 1.5 volts, meaning the DC input voltage should exceed 6.5 volts to obtain a 5 volt output. Because the logic system was expected to draw only 300 mA, it was sufficient to use a 6.3 volt filament transformer rated for one-ampere with a large filter capacitor.
The one-ampere transformer, connected with a 5,600 microfarad filter capacitor ensured the DC voltage produced, after the full-wave bridge, would be at least +6.5 volts. Assuming fairly light current demands, we can approximate the voltage input to the regulator.

\[
\begin{align*}
6.3 \text{ V RMS from transformer} \\
x 1.41 \text{ V Conversion} \\
8.88 \text{ V Peak} \\
- 1.28 \text{ V Drop across two bridge diodes} \\
7.6 \text{ V DC to regulator}
\end{align*}
\]

To protect against localized noise related problems, a generous number of small bypass capacitors are interspersed among the logic components. Where possible, they are located close to the chips, and at the end of long printed circuit power supply runs - to minimize any stray inductance.

Output buffers were needed to allow the microcomputer unit to send its logic signals to the off-board power drivers. The relatively weak output capabilities of the microcomputer precluded a direct connection. By using two 74LS245 Octal Bus Transceiver chips, a total of sixteen output pins could be buffered. Although the full function of the 74LS245 chip permits a two-way path to be established, only one direction was called for in this application. For this reason, the chip was hardwired to allow only an A to B data transfer. Hardwiring this direction pin (#1) to +5 volts forces the B to A transfer path to go into a high
impedance state, thus effectively disconnecting it. The remaining eight buffers per chip are capable of sourcing 15 milliamperes, or sinking 24 milliamperes of current each [6]. These current capabilities permit driving the off-board components hard enough to mask noise effects.

Controlled by the microcomputer and associated output buffers, the stepper motors' drive circuit are contained on one card. This card interfaces the microcomputer motherboard to the stepper motors. Control signals from the microcomputer board are in the negative logic form, meaning a true condition (on) is physically represented by 0 volts. This negative logic protocol is mainly a precaution, so that when the microcomputer is reset, or not powered, the power driving circuitry will be off. Many systems are also designed in this way, because TTL gates can usually sink much more current than they can source.

In this system both motors are identically controlled (figure 4-3). An optical isolator drives a PNP Darlington power transistor, switching the path to the unregulated power supply's positive rail (+V raw). One leg of a stepper motor's phase coil now has half of its current path complete. The other leg of each phase coil (common) has one of two conditions imposed on it. First, if the NPN power transistor is off, the current flow to ground is a path through a 10 ohm power
FIGURE 4-3  Stepper Motor Drive Circuit
resistor, and the motor is in a low power, holding mode (1/3 power). Secondly, if the NPN power transistor is on, then the path of lower resistance is followed through the NPN Darlington, resulting in the normal, full-power mode. These power modes are controlled by the microcomputer software, and for the moment we will assume each phase coil of the motor has a ground path.

Each of the four, phase coils has an identical power driving circuit. Upon receiving a low output (true) from the microcomputer board, the optoisolator's LED is forward-biased, and current flows through the limiting resistor. Limiting the current to 3.3 mA insures a typical long life for the optoisolator, while still providing a large noise margin [7]. Also important is biasing the optoisolator's output transistor hard enough. This NPN output transistor acts as a pull-down switch, sinking current from its collector. Connected to this collector, a PNP Darlington power transistor (TIP-106) is configured as a collector-follower. The optoisolator sinks the Darlington's base current, along with the 10K pull-up resistor's current. This pull-up resistor compensates for any leakage currents, and ensures a quick turn-off of the Darlington.

Assuming a typical current gain of 3000 for the Darlington, the worst-case current sinking condition for
the optoisolator was calculated to be .77 milliamperes as follows:

\[ \frac{1A}{3000} = .33 \text{ mA} \text{ Darlington base current} \]
\[ (5-.6)V/10K = .44 \text{ mA} \text{ Pull-up resistor} \]
\[ .77 \text{ mA} \text{ Sinking current} \]

Since the optoisolator's typical minimum transfer ratio is specified to be 80%, the Darlington is consequently biased on hard. As previously discussed, this Darlington power transistor's collector is connected to one of the four phase-coils of the stepper motor. In addition to the transistor's internal collector-emitter diode, a protective suppression diode is placed across each coil to dissipate the high-voltage spike produced each time the coil is switched off.

The common connection of all four, motor-phase-coils, leads to the one-third power circuitry. This circuitry occurs only once per motor. In essence, all of a motor's current either passes through a power resistor with a large voltage drop, or through an NPN Power Darlington transistor. The driving circuit is a symmetric copy of an individual phase-coil driver. The microcomputer sends a low (0) control signal to achieve the full power mode. The optoisolator turns on, sourcing current to the NPN Darlington base. The resulting low resistance path conducts current to ground, and the full-power mode is in effect.
In contrast, a high (1) microcomputer control signal turns off the optoisolator, and the 10K pull-down resistor turns off the NPN Darlington. The only current path for the motor is through the power resistor, and the one-third power mode is in effect. Since two out-of-four phase-coils in the stepper motor are engaged at any given time, the NPN power Darlington carries twice the load of its series connected symmetric twin. As illustrated in previous calculations, this circuit has plenty of reserve, and the increased load on this "common" transistor, causes only additional heat dissipation. In the one-third power mode everything runs cooler with the exception of the power resistor. To allow for proper convection cooling, the NPN Darlingtonos are heat sinked and placed at the top of the card. Not only for cooling, but also, to conserve space, the power resistors are located off the card, near the power supply.

In addition to the stepper motors, four other devices are controlled by the microcomputer unit. These devices are the piston electromagnet, vacuum solenoid, an alarm, and a spare output that currently drives an LED. Interfacing these devices to the microcomputer is accomplished on a separate card. The electrical circuitry is similar to the driver circuit of each motor phase-coil (figure 4-4). An optoisolator's collector
**FIGURE 4-4** Output Interfacing Circuit

biases a PNP power Darlington transistor, allowing current to reach the required device.

Located inside the pneumatic piston assembly, the electromagnet design included several interrelated factors: size, weight, field strength, operating current, and heat dissipation. To fit inside the piston, the size and weight were fairly well established. An electromagnet's strength is proportional to the number of windings and the current through them. The heat generated increases with an increase in the current drawn, and is inversely proportional to the internal resistance caused by the wire diameter losses. Since magnetic fields always circulate, it was advantageous to provide a return path for the flux produced. This return reduces the "air gap" in the flux path, greatly increasing efficiency and therefore strength, at a given input power. Another consideration was given to the physical materials that would comprise the flux path. Because the magnet was to be operated with a D.C. source, frequency-related losses such as eddy currents could be neglected; however, permanently magnetizing the core would be unacceptable.

Experimentation led to the development of a satisfactory electromagnet. Approximately 2000 winds of thirty-gage magnet wire was wrapped around a grade-three, hardened steel bolt. The hardened bolt exhibited
no permanent magnetization. A small length of steel pipe forms a shell around the windings, with an iron cap providing an anchor for the electromagnet core. A complete flux path exists when the electromagnet is in contact with a sheet metal topped block.

Several choices existed regarding the stepper motors. As discussed in Chapter Three, adequate stepper motors were located, and experimentation demonstrated their ability to drive the required loads. The motors used were typical, 1.8-degree/step, permanent magnet hybrid types (SLO-SYN M061-FD02). Each phase-coil winding was measured to be 5 ohms, confirming each motor's coil rating of 5 volts at 1 ampere. These motors have six wires: four to energize individual coils and two act as separate "commons" between coil pairs. Since it was determined that a unipolar drive circuit would be used rather than a series bipolar configuration, the two "commons" were tied together.

Because motor heating is the most determining factor of its life and reliability [11], attention focused on balancing the input power safely below the maximum, while still providing the necessary output torque to accurately operate the system. Frequently, a suppression resistance is installed, forming a return path for the inductive decay currents produced when a drive transistor turns off. This suppression resistor
can be used to raise the mid-frequency break point, and allow a given motor to step faster - at the expense of increased overshoot and settling time [11].

Because we were only interested in the low-speed operating range, this expense was not justified. Therefore, no suppression resistor was used; and it was assumed that the diodes across each motor coil could handle the increased stress. To limit current to the stepper motor's windings, a series resistor was necessary. This resistor also reduces the time constant responsible for bringing the current up to full value. In most stepper motor circuits, two equal, series, resistors are connected from each separate common, and then both connect to the same power supply rail. Decreasing these resistors can increase the low speed torque and accuracy, but the price is increased heat dissipation in both the motor and power drivers [11].

Instead of using two equal resistors, essentially in parallel, a single series resistor was used. This simplified configuration presented no low-speed problems. Calculating the resistor's actual value was a matter of applying basic circuit laws. Assuming the power supply's D.C. voltage to be 8.5 volts when loaded, and subtracting two Darlington voltage drops of 0.9 volts each, leaves 6 volts between the motor and the resistor. Due to the "common" connection, the motor's
resistance is 2.5 ohms (Rm/2). Letting the current per phase-coil (Im) be .75 A, we can see the motor would draw a total of 1.5 A - through a resistor whose value is 2 ohms.

\[ \text{Rs} = \left( \frac{V_{ps} - 2V_{ce}}{2\text{Im}} \right) - \frac{\text{Rm}}{2} \]

Calculating the required wattage for this series resistor, we can see that a 10 watt power resistor will be adequate - using "1.75" as a safety factor [11].

\[ \text{Wattage} = (2\text{Im}) \times \text{Rs} \times 1.75 \]

It is interesting to note the following observation about this power resistor. In our single-series-resistor configuration, the required wattage increases at four times the rate of a dual-series-resistor configuration. This may be one reason why no one seems to use the simpler single resistor circuit. At higher voltages and currents, this set-up would definitively be prohibitive.

Hall-Effect limit switches are used to calibrate the system and detect any out-of-bounds conditions. When one of these four switches is depressed, the corresponding input pin of the microcomputer unit is pulled to ground. This signals the microcomputer that an extreme physical position has been reached. Outwardly resembling conventional microswitches, these Hall Effect switches electronically operate more like a transistor. The switches used in this application have
open-collector outputs. When a switch is not depressed (free), the output transistor is off, and the output terminal "floats", providing no current path to ground. However, when depressed, the output transistor is on, and the output terminal can sink current to ground.

As in most open-collector designs, a pull-up resistor is needed to minimize leakage currents and establish a solid quiescent voltage level. Pull-up resistors of 3300 ohms were used on the microcomputer card to provide better noise immunity than the 10K ohms value suggested by Honeywell. Reducing this pull-up value results in a 1.5 mA current sinking demand, which can easily be accommodated according to the switch specifications of 8 mA average output current [12].

Isolating the microcomputer circuitry with a separate power supply allows the use of an unregulated D.C. power supply for the higher current, inductive loads. This power supply had to be able to handle the high inrush currents as well as a fairly hefty 3.5 A average current. Since the power driving circuitry was already designed, it was known that a D.C. level above 5 volts would be needed to drive the stepper motors. This compensates for the two voltage drops across the Darlington's' and the series resistor. To keep the D.C. level as smooth as possible, a large, 17,000 microfarad capacitance was connected in parallel across the supply.
A 6.9 volt center-tapped transformer was connected with two power diodes, in a half-wave rectifier configuration. These diodes are rated at 25 amperes maximum average current and over 500 amperes maximum surge current! Although truly excessive, these diodes ensure the high inrush currents needed to charge the capacitors at start-up are not a problem. With no load, the capacitor charges up to the transformer's peak voltage minus one diode's forward voltage drop.

\[
\begin{align*}
6.9 \text{ V RMS from transformer} \\
\times 1.41 \text{ V Conversion} \\
9.73 \text{ V Peak} \\
-0.7 \text{ V Diode drop} \\
9.03 \text{ V D.C. level (no load)}
\end{align*}
\]

Although the no load voltage exceeds previous assumptions, it obviously would be less under a load.

The previous discussions have illustrated the main design considerations involved in producing a reliable electronic system. More details regarding sources for these electrical components can be found in the appendices. Next, we will discuss the mechanical design aspects of the system.
V. Mechanical Design

Even though the mechanical aspects were not the focal point of this project, nearly every mechanical detail was carefully thought out and designed for this specific purpose. The discussion to follow will show the design and development of the mechanical systems needed in the model. More details can be found in the Appendix D. Because the natural integration of mechanical components started in the center and worked outward, the discussion will follow the same inside to outside progressions where possible.

The pneumatically-powered, vertical actuator assembly had to be designed so that it would have sufficient strength to pick up one block, an internal electromagnet, its own piston weight, and overcome friction. An interest in minimizing weight and overall size was also present, since both X and Y-motors would be moving this assembly. It was estimated that a small vacuum blower motor could lift a forty-inch column of water. This equates to more than one pound per-square-inch of suction. A two-inch diameter piston can lift three pounds, when subjected to one pound per-square-inch of suction. The total weight to be lifted was estimated to be well under two pounds. So, a two inch diameter rod of aluminum was hollowed out to reduce
weight and make room for an electromagnet to be fit inside.

To eliminate any rotational motion of the piston, a key-way notch was cut into the piston wall. This key runs only on the "active" piston length. The upper-piston length is unkeyed to avoid excessive air losses when fully extended. The cylinder walls were machined from a cast acrylic pipe to match the piston bore and leave a slight air gap, to reduce friction. Cast acrylic is light weight, and could easily be cemented. This also made it a natural choice for the Y-sled base. The base dimensions were chosen as the minimum possible to ensure stability when moving in the X and Y-directions.

Next, the wheel dimensions were calculated, allowing for a one-eighth-inch-wide U-groove to keep the Y-sled aligned. In order to reduce rotational inertia and keep friction low, the aluminum wheels were designed as small as possible, allowing just enough clearance for the bearings to be pressed in. To avoid any small misalignment from jamming the sled's rolling motion, the grooved wheels were only installed on one side of the sled. The other side has blank wheels that support their share of the load, and are free to move axially. Finally, chain mounts were fabricated of aluminum and mounted near the Y-sled's center of gravity.
The track width of the Y-sled wheels determined the width of the X-sled. Mounted on the X-sled would be a stepper motor responsible for pulling the Y-sled. A suitable motor mount was fashioned from aluminum, both to reduce weight, and to dissipate heat from the motor. The other end of the X-sled is also aluminum, and contains two bearings that support the freewheeling second gear of the Y-sled drive system. Additionally, these two mounts each have two wheels bolted into one side. Identical to the Y-sled wheels, they ride along the edges of the frame beneath.

In order to evenly advance the X-sled along its rails, two chains were needed. The chains were fastened such that they would evenly pull near the X-sled's vertical center of gravity. Keeping the chains synchronized required an axle at either end of the warehouse. Both axles are one-quarter inch diameter, precision ground, steel rods. Mounting the axles to the frame is done with ball-bearing end mounts. Two gears mount on each axle and pull the chains connected to the X-sled. These four gears are identical, and their diameter was selected to yield about twenty-five thousandths of an inch chain travel per motor step. This is easily confirmed since:

\[(\text{eff. dia.}) \times (\pi) \times (1/200 \text{ steps}) = (\text{dist per step})\]
Therefore, gears with a pitch diameter of 1.78 inches (56 teeth) were selected to complement the miniature chains. To offset the tension imposed by the chains on the small axles, a mid-axle support was added to each side.

Attached to one axle through a similar gear and chain arrangement is the X-axis stepper motor. Its adjustable motor mount was designed to allow for proper chain tightening. The entire aluminum motor mount also serves as a heat sink. To allow for a speed change, the gearing was chosen to be slightly different than one-to-one. This allows a slight (15%) torque increase with a corresponding loss of speed. If desired, these gears can be reversed for the opposite effect.

Because unexpected results often occur in prototypes, each sled has a "stop" at the end of its path. These "stops" prevent damage to the gears should a limit switch, the software be in error. The X-sled "stops" were designed into the X-axis limit switch mounts, while the Y-sled "stops" also double as chain guides.

The vacuum blower is a two-stage series-wound AC/DC motor. Since these are generally loud devices, an enclosure was built to muffle most of the undesirable noise. This vacuum source is channeled to the piston by means of a solenoid vacuum switch. The microcomputer
unit sends a control signal to energize the solenoid. The solenoid lifts a leather faced button, uncovering a large hole and thus venting the vacuum source. The piston, no longer under vacuum, smoothly drops. The vacuum switch was designed in a manner, so that, a failure in either the solenoid or supporting logic would leave the piston in the up position. This is a necessary safeguard to prevent system damage that may occur if the system moves while the piston is down. An added benefit is that when the piston is down for an extended time, as in a rest position, the blower motor receives the maximum airflow for cooling purposes. This vacuum switch is connected to a "boom", positioned over the warehouse. This half-inch conduit "boom" provides support for the flexible supply line, while guiding the airflow.

Integrating the electrical and mechanical designs previously mentioned, produces the necessary system hardware. Controlling and coordinating the hardware components so that specific tasks can be executed is the function of the software. The next chapter will explore the design and implementation of that software.
VI. SOFTWARE

Performing the logic of the automated warehouse is the 6801 assembly language software, written exclusively by the author, for use in a one-chip microcomputer unit. The following chapter will explore the routines — which together control the warehouse. After an overview of the functions, each core routine will be explained and supplemented with both a flowchart and assembled listing. Following the core routines, the secondary and interrupt driven routines will be probed.

A host terminal or computer serially communicates with the automated warehouse computer by issuing recognized macro-commands. Because these commands (from the host to the model) are a significant part of the system, they will be discussed here.

The macro-commands can be broken into two main groups: (1) One and three character sequential commands, (2) One character immediate commands. The first group is used most often, supplying what and where information. The first character is what action is to be performed. The options include R (Rest), U (Pick Up), and D (Put Down). The next two characters are where the action will be performed. The column number (0-7) is followed by the row number (0-7). With the
exception of additional macro-commands, other characters are ignored. The user is free to use spaces, commas, and any lowercase letters to increase readability. These three 3-character macros are placed in the warehouse queue and executed sequentially.

Another sequential macro is the character "C" (calibrate). It is similarly placed in the queue, but unlike the previous commands, this macro is only one character in length. By using the "C" macro, the user is instructing the warehouse to calibrate itself at the nearest corner. The column and row numbers are not needed, and if any are supplied they will be ignored.

The next group of macro-commands are all one character in length, and take effect immediately under most conditions. These macros are system functions and are not placed in the warehouse queue. They are only recognized if they occur as the "first byte" in a move sequence, and ignored elsewhere. The "first byte" simply means that all three bytes of the previous move have been found and the sequence has started over, looking for a first byte. For example, in the sequence:

U 1 2, D 3 4, C, R 0 0U55D66

the first bytes here are U, D, C, R, U, and D. The spaces and commas are ignored.

By invoking the macro "Cntrl-X" (one character), the queue is immediately cleared. The "%" (percent
symbol) causes the warehouse to immediately halt all operations until a "^" (caret) is received. The ">" (greater than symbol) increases the stepper motor speed by about 0.25 steps per second. The "<" (less than symbol) similarly decreases the speed. These last two speed commands are accumulative. Finally, the "=" (equal symbol) negates all speed changes and returns the system to the base (default) speed. A macro-command summary and additional examples may be found on page 137.

The microcomputer software can be broken into two primary groups: the main routines (figure 6-1), and the interrupt driven routines (figure 6-2). Each of these routines performs a unique function. First, the assembler declarations are made, and the system is initialized. Next, a check is made to see if a demonstration sequence is requested by the user. The calibration routine sets the desired destination as the nearest corner, where the limit switches will be activated.

Before the motors are actually moved from their present position, the pneumatic piston is lifted and the system is prepared for motion. Next, the rotational direction is determined for each motor, and the specific phase calculations are performed. After both motors are simultaneously advanced one step, a delay is
FIGURE 6-1  Main Routines Flowchart
FIGURE 6-2  Interrupt Routines Flowchart
encountered to regulate the stepping speed.

Next, the limit switches are checked to see if an out of bounds condition has occurred. If the limit switch routine detects such a condition, a correction is made to the present position counter. After all the limits have been checked for this one motor step, the remaining motor steps are similarly performed until the required destination has been reached.

Next, the routine which permits serial communications with the host computer is encountered. If the microcomputer's queue has sufficient storage space remaining, then a handshake to the host will enable data transmission. Otherwise, data transmission will be inhibited at this time.

The Z-axis action routine executes the pick and place movements by controlling the pneumatic piston and electromagnet. Next, a test is performed to determine if there are any additional commands in the queue. If no commands are present, then the snooze routine will be executed, causing the system to go into a low-power mode until a command arrives. With at least one command known to be in the queue, the next command is extracted, preserving the input sequence, and then expanded into the appropriate action and location. If the command is a calibration, the software will jump to the routine which will prepare the system for movement.
The second group of routines are interrupt driven, that is, they can interrupt the main program whenever a specific need must be fulfilled. When an incoming byte (character) is received, a serial communications interrupt routine processes host requests until the queue is full, or no more data is received. Additional interrupt driven functions include a manual halt routine, a catastrophe routine, and microcomputer error detection.

In order to allow storing a maximum of commands in the queue, several strategies had to be developed. Since the internal RAM is limited to 128 bytes, and 100 bytes is dedicated to the queue, only 28 bytes remain for the system. Sixteen of these bytes are used for local variables, leaving only 12 bytes for the stack. Since an interrupt, or subroutine call, uses 7 bytes to store registers on the stack it was important to limit the stack operations so that only one interrupt, or one subroutine, occurs at any given time. Because a serial communications interrupt can occur at any time, this precluded the use of any subroutines. However, this limitation is not much of a handicap, because the microcomputer can address 8k of EPROM, and the only real subroutine candidate would be the timing loops. In fact, the elimination of subroutines leads to a faster-executing, and more readable program.
It is worthy to mention the instruction execution times for this microcomputer. Since the HD63P01 uses a pipelined instruction pre-fetch method to speed operations, it is not possible to simply calculate the number of cycles with the clock speed to obtain a timing delay. Factors such as branches, and short (1-cycle) instructions make it difficult to predict exact timing loops. Through experimentation, it was found that the microcomputer actually executes 10 to 15 percent faster than expected.

In the interest of making a readable program, all operations of a bit-by-bit nature are written in binary, and all counting loops are in decimal. Where possible, all word (2-byte) operations are shown as four hexadecimal digits. Descriptive labels are used for clarity, and similar labels with a number following (eg. STALL2, STALL3) indicate a nearly identical routine exists, and was probably already explained. Comments are located on the right-hand side in lower-case letters.

The first page of the source-code listing establishes the labels, local variables, dimension constants, and interrupt vectors needed by the assembler (figure 6-3). These labels, memory locations, and constants are referred to by name throughout the program, which not only helps to clarify the source
FIGURE 6-3  Assembly Initializations Flowchart
code, but also simplifies modifications and enhances portability. The reader should notice the alphabetical arrangement of these entries; as well as, the format (binary, decimal, hex, octal). Throughout this paper, all labels and instruction mnemonics will be referred to as they appear in the source code - in all capital letters with labels in BOLDFACE type. The assembler used to generate the source code only recognized the 6801 "standard" instruction set, so six instructions particular to the 6801 enhanced instruction set were avoided. The assembler determines the mode in the following manner: # indicates immediate addressing, $ indicates that the number following is in hex, % indicates binary, @ indicates octal, and ' (apostrophe) denotes an ASCII literal character. Comment lines always begin with an asterisk (*).

Before the executable code can begin, the assembly initializations must occur. As outlined in figure 6-3, and the actual listing (figures 6-4A, and B) this initialization is straightforward. Starting with the equates, the AUTOCAL label sets the interval between automatic calibrations. If AUTOCAL moves occur without a "Calibrate" macro-command then one will be inserted. The actual number (50) was chosen to demonstrate its operation. Labels DATDIR1-4 define the processor's data direction registers (1-4). A bit value of "0" sent to
AUTOMATED WAREHOUSE CONTROLLER (Version AWC22)
Developed by Steve Ricca, 1987-88
All rights reserved

Define assembler labels, constants, variables
Source code = 6801 assembly language

begin

0032 AUTOCAL EQU 50
0033 DATDIR1 EQU $0000
0034 DATDIR2 EQU $0001
0035 DATDIR3 EQU $0004
0036 DATDIR4 EQU $0005
0037 GO232 EQU $11101111
0038 PORT1 EQU $0002
0039 PORT2 EQU $0003
003A PORT3 EQU $0006
003B PORT4 EQU $0007
003C QFILLED EQU 85
003D QMAX EQU 100
003E QTOP EQU $00FF
003F QCDATA EQU $0012
0040 RMODECR EQU $0010
0041 STOP232 EQU $11111111
0042 TRCSR EQU $0011

Local Variables - RAM Locations

ORG $0080
0080 DESTX RMB 2
0082 DESTY RMB 2
0084 LASTCAL RMB 1
0085 PRSPOS RMB 2
0087 PRSYPOS RMB 2
0089 QENTRY RMB 1
008A SPEED RMB 1
008B TEMPHI RMB 2
008C TEMPHI EQU TEMPHI+1
008D XPHASE RMB 1
008E YPHASE RMB 1
008F ZACTION RMB 1

FIGURE 6-4A Assembly Initializations Listing
### Table of Phase Bits

- Bits 7-4 = X motor
- Bits 3-0 = Y motor
- These must stay in this order for this system
- CW = $5, 6, A, 9$
- CCW = $9, A, 6, 5$
- Toward minimum
- Toward maximum

<table>
<thead>
<tr>
<th>ORG</th>
<th>$FFD0 phase sequence $</th>
<th>Bits 7-4</th>
<th>Bits 3-0</th>
</tr>
</thead>
<tbody>
<tr>
<td>FFD0</td>
<td>X</td>
<td>Y</td>
<td></td>
</tr>
<tr>
<td>FFD1</td>
<td>FCB 10011001</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>FFD2</td>
<td>FCB 10011010</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>FFD3</td>
<td>FCB 10010110</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>FFD4</td>
<td>FCB 10101010</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>FFD5</td>
<td>FCB 10101010</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>FFD6</td>
<td>FCB 10100110</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>FFD7</td>
<td>FCB 10100101</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>FFD8</td>
<td>FCB 01101001</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>FFD9</td>
<td>FCB 01101010</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>FFDA</td>
<td>FCB 01100110</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>FFDB</td>
<td>FCB 01100101</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>FFDC</td>
<td>FCB 01011001</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>FFDD</td>
<td>FCB 01011010</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>FFDE</td>
<td>FCB 01010110</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>FDFD</td>
<td>FCB 01010101</td>
<td>3</td>
<td>3</td>
</tr>
</tbody>
</table>

### Dimension Constants

<table>
<thead>
<tr>
<th>ORG</th>
<th>$FFE0 (FFFE) $</th>
</tr>
</thead>
<tbody>
<tr>
<td>FFE0</td>
<td>BASESP FDB 63650</td>
</tr>
<tr>
<td>FFE2</td>
<td>DWNTIME FCB 20</td>
</tr>
<tr>
<td>FFE3</td>
<td>UPTIME FCB 25</td>
</tr>
<tr>
<td>FFE4</td>
<td>XMAX FDB 2945</td>
</tr>
<tr>
<td>FFE5</td>
<td>XMN FDB 2000</td>
</tr>
<tr>
<td>FFE6</td>
<td>XDISTCEN FCB 135</td>
</tr>
<tr>
<td>FFE7</td>
<td>YMAX FDB 2688</td>
</tr>
<tr>
<td>FFE8</td>
<td>YM skin FDB 2000</td>
</tr>
<tr>
<td>FFE9</td>
<td>XDSTCEN FCB 84</td>
</tr>
</tbody>
</table>

### Interrupt Vectors

<table>
<thead>
<tr>
<th>ORG</th>
<th>$FFE0 (FFFE) $</th>
</tr>
</thead>
<tbody>
<tr>
<td>FFE0</td>
<td>TRAP FDB FATAL</td>
</tr>
<tr>
<td>FFE2</td>
<td>SCI FDB BYTEL</td>
</tr>
<tr>
<td>FFE3</td>
<td>TOF FDB FATAL</td>
</tr>
<tr>
<td>FFE4</td>
<td>OCF FDB FATAL</td>
</tr>
<tr>
<td>FFE6</td>
<td>ICF FDB FATAL</td>
</tr>
<tr>
<td>FFE8</td>
<td>IRQ1 FDB HALT</td>
</tr>
<tr>
<td>FFE9</td>
<td>SOFW FDB FATAL</td>
</tr>
<tr>
<td>FFEA</td>
<td>NMI FDB FATAL</td>
</tr>
<tr>
<td>FFB0</td>
<td>RES FDB START</td>
</tr>
</tbody>
</table>

---

**FIGURE 6-4B Assembly Initializations Listing**
these system registers configures that particular input/output line as an input, while a bit value of "1" configures the line as an output. Label GO232 provides a "0" bit 4, which is needed for the serial communications handshake protocol. The labels PORT1-4 are system registers through which the input/output data is passed in accordance with the data direction configuration. Each port can be thought of as eight I/O lines or bits, that can be manipulated in parallel just like memory operations. Due to the internal hardware arrangement of these ports [7], it is possible to read an output port with meaningful results - a detail that proved useful. Constants QMAX and QFILLED control the size of the command queue. The queue can accommodate up to QMAX (100) commands, and is considered "filled" if more than QFILLED (85) commands are present. The queue grows downward from QTOP, which is the highest internal RAM location. The next command to be executed is always at this address. RECDATA, RMODECR, and TRCSR are all system registers which control the serial communications interface. The received data byte is held in RECDATA, while the others are the "rate and mode control register", and the "transmit/receive control status register".

Several local variables are maintained in RAM. Sixteen bytes of memory are dedicated to these
frequently changing variables, starting at the lowest available RAM ($80-8F). The system memory map is shown in figure 6-5. The absolute coordinates of the X and Y-destinations are stored in DESTX and DESTY respectively. These are word quantities (2-bytes each) stored in the conventional "high-low", unsigned format. Similarly, PRSXPOS and PRSYPOS hold the present X and Y-positions. To keep track of the number of commands executed since the last calibration cycle, LSTCAL keeps a count that is incremented after each move. QENTRYS counts the number of commands presently in the queue. If QENTRYS is zero, then the queue is empty. RAM variable SPEED holds a number that affects the speed of the stepper motors. Its value can be changed during run-time by the user. TEMPHI and TEMPO together form a temporary word storage. Obviously, these memories are not pushed on the stack during an interrupt, so their use should be carefully examined. In order to keep track of the present phase that each of the four-phase stepper motors is currently operating under, the variables XPHASE and YPHASE hold the presently active phases. Finally, the current Z-axis operation to be performed is stored with the ZACTION variable.

The next group of assembler directives forms a table of "phase bits". A more detailed discussion will follow in the phase calculation section. At this time
FIGURE 6-5  System Memory Map
simply take note that these binary source statements form a look-up table that starts with the label PHBITS. They could be located anywhere in the 8K EPROM address space, but for convenience are currently on the "last page" ($FFD0).

Similarly, the dimension constants are at the high EPROM addresses. These constants are particular to this physical model, and were changed frequently during system testing. The word constant BASESP is used in a timing loop to set the normal, or base speed of the stepper motors at 100 steps per second. UPTIME and DWNTIME furnish delays allowing the piston to be lifted or dropped. The absolute coordinates of the minimum and maximum dimensions inside the warehouse are established with the constants XMIN, XMAX, YMIN, and YMAX. Of these two pairs, one constant in each direction is arbitrary within certain bounds to be discussed later. To this note, the minimums were both chosen to be 2000. Since the linear distance per X-direction step is approximately .025 inches, the model requires 945 steps in the X-direction to traverse the warehouse. Due to a difference in gearing, the Y-direction step is .035 inches, and only 588 steps are needed. The relative distance between the center of adjacent columns (X) or rows (Y) is given by constants XDSTCEN and YDSTCEN.
The last grouping of labels consists of interrupt vectors. Unlike the previous labels, these address vectors are not relocatable. Their locations were predetermined by the microcomputer manufacturer ($FFEE-$FFFF). When an interrupt condition occurs, the microprocessor stores the vital registers (7 bytes) and jumps to the target address provided by the appropriate interrupt vector. Many of these vectors are unused in this system. If an interrupt through these "unused" vectors is attempted, then clearly an error of some sort exists, and the program will jump to a fatal error routine (Fatal) that will be explained later. The SCI interrupt signals a serial communications interface request. This request will be serviced beginning at the address BYTE1. The non-maskable interrupt vector (NMI) will jump to a catastrophe routine (Fatal). Lastly, but highest in priority, is the reset vector (RES), used to start the system from a power-off state. First, the internal chip mode is selected (one chip), then all ports are configured as inputs, and the program begins executing at the starting address (START).

As shown in the flowchart (figure 6-6) and listing (figure 6-7), the program begins by initializing important run-time parameters at the START address (E100). First, the stack pointer is loaded, allowing nine bytes of free stack space. Clearing QENTRYS zeros
FIGURE 6-6 System Initialization Flowchart
**START PROGRAM - Initialize, set-up serial protocol**

---

```
START LDS #00098   * Declare stack area
CLR ZENTRY     * Queue empty
LDAA #00000111  * 150 Baud
STAA RMODECR   * Serial protocol
LDAA #00011000  * RIE, RE
STAA TRCSR     * Enable receiver
LDAA #STOP232   * Stop handshk to host
STAA PORT2     * P24 = output
LDAA #00010000  * Enable interrupts
STAA DATDIR2
CLI
```

**Configure Ports**

```
LDAA #00000000   * Port1 all inputs (8)
STAA DATDIR1
LDAA #11111111  * LED on, others off
STAA PORT3
LDAA #11111111  * Port3 all outputs (8)
STAA PORT4
STAA DATDIR3
STAA DATDIR4
```

**Initialize Stepper Motors**

```
CLR XPHASE     * Motors start at
CLR YPHASE     * phase 0
LDAA PHBITS
STAA PORT4
LDX #35000    * Warm up, low power
BNE STALL1
STAA STALL1   * Wait .2 sec for
STAA SPEED    * motor warm-up
```

**Establish Initial Position**

```
LDD XMAX XMAX   * Initialize X Position
ADDX XMIN +XMIN
LSRD     /2
SUBD $5 
STDX FRXPOS
LDD YMAX YMAX   * YMIN
ADDY YMIN +YMIN
LSRD     /2
SUBD $5 
STDY FRSYPOS
```

---

**FIGURE 6-7 System Initialization Listing**
the queue, then the internal serial communications interface hardware is initialized. Bits 0, and 1 select the transfer rate, while bits 2, and 3 select the NRZ format and internal clock source. Next, the transmit/receive control bits (3 and 4) are set to enable Port 2 bit 3 (P23) to receive serial information, signalling with a SCI interrupt when a byte or error has been received. Because we are not transmitting any serial data we must leave the transmit bits clear (1 and 2). With the receiver initialized, a handshake can be relayed to the host. By storing the STOP232 data at PORT2, bit 4 is latched high, but not output yet. First, we must initialize Port 2 bit 4 (P24) as an output. Once the data direction register is loaded, the handshake will be continuously sent. It is important to store the data to the port before declaring the I/O direction. Otherwise unknown and potentially hazardous control signals may be initially sent out. This is especially true on this microprocessor's Port 2, since bits 0, 1, and 2 are used for mode programming. Next, the interrupt mask is cleared, enabling interrupt requests.

In a similar sequence, Port 3 is configured as all outputs. All Port 3 outputs are off (1), with the exception of P35. This bit controls an LED indicating the system is active and not in the "Snooze" mode. The
LED is simply a spare control output that could be used for future modifications (see Chapter 10). The input/output ports are now configured as shown in figure 6-8. Notice that each input or output is true when the corresponding bit is low (0 volts).

Next, to initialize the stepper motors, the first entry in the phase-bits table is output to PORT4, and the motors hold at this position, in the low-power mode. This allows a motor warm-up period. The variable SPEED starts with the value 127. As will be seen later, an increase in this number results in a speed increase, while a decrease slows the stepper motors.

Since the initial position at start up is unknown, an assumption is made at this time. The absolute coordinates of the warehouse center are calculated based on the minimum and maximum dimensions. Then an offset is subtracted, and the results stored as the present position (PRSXPOS and PRSYPOS). As will be seen, this assumption will cause a calibration to the column 0, row 0 corner, regardless of the actual position.

Next, the demonstration switch is checked as shown in figures 6-9 and 6-10. For the moment, let us assume the switch is off and the demo routine is jumped over. The program will then fall into the "smart" calibration routine.
PORT 1 - SYSTEM INPUTS
LOW = TRUE

<table>
<thead>
<tr>
<th>PORT 1</th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>RUN</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DEMO</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SWITCH</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Unused</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HALT</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>/ PANIC</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SWITCH</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Unused</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>YMIN</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LIMIT</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SWITCH</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Unused</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>XMIN</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LIMIT</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SWITCH</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Unused</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>YMAX</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LIMIT</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SWITCH</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Unused</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>XMAX</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LIMIT</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SWITCH</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

PORT 2 - SERIAL COMUNICATIONS
AND CHIP MODE SELECTION

<table>
<thead>
<tr>
<th>PORT 2</th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>NOT AVAILABLE TO USER</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RS232</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HNDSHK</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SERIAL INPUT</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MODE INPUT</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MODE INPUT</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MODE INPUT</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

PORT 3 - SYSTEM CONTROLS
LOW = TRUE (ON)

<table>
<thead>
<tr>
<th>PORT 3</th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>UNUSED</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>UNUSED</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AWAKE LED / SPARE</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ALARM</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FULL POWER X MOTOR</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FULL POWER Y MOTOR</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ELECT-MAGNET ON</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DROP PISTON</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

PORT 4 - MOTOR OUTPUTS
LOW = TRUE (ON)

<table>
<thead>
<tr>
<th>PORT 4</th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>COIL</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>#4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>COIL</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>#3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>COIL</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>#2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>COIL</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>#1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>COIL</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>#4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>COIL</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>#3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>COIL</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>#2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>COIL</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>#1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

FIGURE 6-8 I/O Port Configurations
FIGURE 6-9 Demonstration Loader Flowchart
Figure 6-10 Demonstration Loader Listing
SMCALB is the start of the "smart" calibrate routine (figures 6-11, and 6-12). The function of this routine is to cause a calibration to the nearest corner, hence the term "smart". This is achieved by first loading word variables DESTX and DESTY with the maximum dimensions, plus 1234. Adding 1234 assures that the destinations are far past the physical boundaries. As will later be seen, these exaggerated destinations ensure that the X and Y-motors will drive until the limit switches are reached.

With these (maximum) destinations, the X- and Y-motors would always move towards the XMAX, YMAX corner. This is where the "smart" part comes in. First, the X-direction midpoint is calculated, and if the present position (PRSXPOS) is past this midpoint, then the recently set X-destination (XMAX+1234) stands, and the program branches to YCALB. If however PRSXPOS is less than the midpoint, then the X-destination is changed to XMIN minus 1234, causing the X-motor to seek an equally unobtainable coordinate, but this time it is in the minimum direction. At YCALB an identical process is performed in the Y-direction. Based on their present positions, X and Y-destinations past the nearest corner will now be stored in DESTX and DESTY. At ZCALB, the final instructions in this routine clear the last calibration counter (LSTCAL), and store a code that will
SET X & Y DESTS.
PAST MAX DIMS.

XCALB
PRES X POS
PAST MIDPT?

YES

NO

CHANGE X DEST.
TO BEFORE X MIN.

YCALB
PRES Y POS
PAST MIDPT?

YES

NO

CHANGE Y DEST.
TO BEFORE Y MIN.

ZCALB
SET PISTON FLAG
CLR CALB. COUNT

FIGURE 6-11 Calibration Routine Flowchart
SMART CALIBRATE ROUTINE
* Alters destinations to beyond nearest corner *

SMART CALIBRATE ROUTINE

* Change destinations beyond maximum X

SMART CALIBRATE ROUTINE

* Calculate midpoint X

SMART CALIBRATE ROUTINE

* Change destinations beyond maximum Y

SMART CALIBRATE ROUTINE

* Calculate midpoint Y

SMART CALIBRATE ROUTINE

* Past midpoint ?

SMART CALIBRATE ROUTINE

* No, change dest.

SMART CALIBRATE ROUTINE

* No, change dest.

SMART CALIBRATE ROUTINE

* Declare calibration sequence in effect

SMART CALIBRATE ROUTINE

* Calibration counter

SMART CALIBRATE ROUTINE

** FIGURE 6-12 Calibration Routine Listing **
allow this calibration sequence to bypass future Z-axis actions.

Falling through the calibration routine, the PREMOVE routine is encountered next (figure 6-13). It is also the target of a jump from the POPQ routine. In either case, these lines of code prepare the system for movement. The PORT3 outputs are read and manipulated to ensure the motors are in the full power mode (P32, P33). The P30 bit is forced high, controlling the piston to lift. All other PORT3 bits are unchanged. The magnet (P31) is left in the same state, so if it was previously energized, then a block will be lifted with the piston. A two-part loop is entered, affording the piston sufficient time to lift.

This loop structure occurs throughout the program and is worth discussing. The outer loop counts down from the constant UPTIME in this case. Each outer loop executes a 16-bit inner loop as well. This inner loop counts down to produce a 0.1 second delay. Used together, the two loops can provide a delay up to 25 seconds in 0.1 second increments. In this case a 2.5-second delay was desired so UPTIME contains 25 - the outer loop object. Over 1.5 million machine cycles are used executing these two loops. When the loops are completed, the PREMOVE routine is done, and the system is ready to move.
PREMOVE ROUTINE

Get system ready for movement.

Piston up, motors at full power

PREMOVE ROUTINE

PREMOVE LDAA PORT3

ANDA $11110011

ORAA $00000000

STAA PORT3

Piston up

OUTLP1 LDX $16250

INLP1 DEX

BNE INLP1

* Full power

* Mag same

* Piston up

* Wait for piston up

* Inner loop = .1 sec

FIGURE 6-13 Premove Routine Flowchart and Listing
The start of the stepper motor movement routine begins at the label XMOTOR (figures 6-14, and 6-15). First, a comparison is made between the present X-position and the desired X-destination. Branches will be taken if the present position is before or past the destination. If no branch is taken, then the X-destination has been reached and the program falls through to check the Y-axis direction. This time, the present Y-position is compared to the desired Y-destination. If they are equal, then the branch is not taken: both X- and Y-destinations have been reached, and a jump to the HANDSHK routine is performed.

Assuming the X-destination was not reached, a branch will be taken to BEFORX or PASTX. These lines of code start the phase-calculation sequence. In order to produce a single step, several factors must be known. First, the stepper motors follow a four-phase sequence. By representing these phases with the numbers 0, 1, 2, and 3, a modulo-4 counter can be used to cycle through the sequence. By ignoring all but the lower two bits of a byte, a modulo-4 type count can be easily maintained. One modulo-4 counter is needed for each motor. RAM variables XPHASE and YPHASE are used to store each counter. By incrementing or decrementing XPHASE, the next X-phase sequence can be selected.
FIGURE 6-14A Motor Stepping Routine Flowchart
FIGURE 6-14B  Motor Stepping Routine Flowchart
**MOTOR STEPPING ROUTINE**

* Compare positions, calculate next phase sequence, table look up, output, delay *

**Start Phase Calculation**

XMOTOR LDX PRSXPOS
CPX DESTX
BCS BFORX
BHI PASTX
LDX PRSYPOS
CPX DESTY
BNE YMOTOR
JMP HANDSHK

* Check if rotation needed on X-motor

BEFOREX INC XPHASE
LDX PRSXPOS
JMP YMOTOR

* Clockwise rotation calculation for X-motor

PASTX DEC XPHASE
LDX PRSXPOS
SUBD #0001
STD PRSXPOS

* CCW rotation calc for X-motor

YMOTOR LDX PRSYPOS
CPX DESTY
BCS BFORY
BHI PASTY
JMP MERGE

* Check if rotation needed on Y-motor

BEFORY INC YPHASE
LDX PRSYPOS

* CW rotation calc for Y-motor

PASTY DEC YPHASE
LDX PRSYPOS

* CCW rotation calc for Y-motor

MERGE LDX XPHBITS
LDX XPHASE
ANDB #00000011
AS
AS
ABX
LDAX YPHASE
ANDB #00000011

* Base address of bits

ACCB = 0000XX00

* Add to base address

ADD $0001

* 1-of-16 bit patterns output to motor coils

DELAY2 LDAA SPEED
LDAB #3
MUL BASESP

* Variable speed

MULTIPLICATION factor

BASE speed(100 s/sec)

LDEX TEMPHI
LDX TEMPHI

* Delay loop

MERGE LDX PHBITS
LDAX PORT4

**FIGURE 6-15 Motor Stepping Routine Listing**
The actual next phase decision depends on the desired motor rotation. A branch to BEFORX indicates a clockwise rotation is needed to move one step closer to the desired destination. XPHASE is incremented, and the present X-position counter is also updated. PRSXPOS now indicates the new position the system will be at when the motors actually move. Conversely, a branch to PASTX indicates a counter-clockwise rotation is needed, decrementing both XPHASE and PRSXPOS.

Next, at YMOTOR a comparison in the Y-direction is performed to determine the Y-motor rotation, if any. At first glance, this may seem redundant, but by checking the flowchart (figure 6-14A) we can see that it is necessary. If the X-position did not match the X-destination a branch would be executed and the Y-position goes untested. By testing the Y-position, the necessary rotation is assessed. Just as in the X-direction, YPHASE and PRSYPOS are updated.

Reiterating for a moment, we are now at the address MERGE. The next phase sequences (0-3) for the X and Y-motor steps are held by XPHASE and YPHASE respectively. The present X and Y-position counters have already been updated to where the next step will put us. No changes have been sent to the motors yet. They remain at the last position, held under the previous phase sequence.
At MERGE the 16-bit index register is loaded with the base address of the PHBITS table (see figure 6-5). This table represents the physical hardware configuration of the motors in this system. The high four bits \((7,6,5,4)\) of an entry in this table correspond to the X-motor, while the low four bits \((3,2,1,0)\) correspond to the Y-motor. A given entry in the table holds the actual control signals for both motors. Any wiring changes to the motors can be accommodated by changing the appropriate bits. A pointer into the table is achieved by first stripping all but the two lowest bits \((1,0)\) from XPHASE. Then these two bits are shifted left twice, and added to the index register which holds the table's base address. Next, the Y-phase sequence counter (YPHASE) is stripped of all but its two lower bits and added to the index register. The index register now holds a completed address that points to one of the sixteen table entries. That particular entry is output to PORT4, and the motors will simultaneously move one step.

A complete motor-move loops through the motor stepping routine as many times as it takes to reach the destination. Once one motor is at its destination, the corresponding phase counter will remain the same, and the output to that motor will remain unchanged even though the other motor is still moving. A worst-case
calculation of this routine, including the position comparison, reveals that 92 machine cycles are executed. Assuming capable hardware, two stepper motors could be simultaneously controlled at speeds up to 6800 steps per second each. Even faster speeds could be achieved with a faster system clock speed.

Obviously some way to keep the motors at a reasonable speed for this system was needed. After the control signals are output to the motors, an approximately 0.01 second loop is entered at address STALL2. Directly before the delay loop is a provision allowing the user to change the stepping rate through a macro-command. The variable SPEED is originally initialized with the value 127. This value is the midpoint of a byte's range (0-255). As will be seen later in the serial communications routine, this value can be manipulated. This SPEED is multiplied by three, and then added to the base speed (BASESP). The 2-byte result is stored in TEMPHI so that the index register can load this value.

The instruction set for this chip actually has an instruction (XGDX), that can exchange the accumulators and the index register. However, as previously mentioned, the enhanced instruction set assembler could not be located, so this instruction was not used. The delay loop is now executed by counting up until zero is
reached. The default values for this loop cause a stepping rate of 100 steps per second.

Between each motor step, the limit switches are checked to see if an out-of-bounds condition has occurred (figures 6-16, and 6-17). Beginning at LIMITSW, the low four bits input at PORT1 are compared to see if any limits have been hit. Since Hall Effect switches were used, there is no concern about switch bounce. This allows a much faster limit routine that only needs to check the switches once. If a limit was hit, then a delay of 0.5 seconds is executed. This delay momentarily stops the stepper motors, allowing the system's mass to come to rest before the next step, which will always be in the opposite direction.

Starting at XLMAX, the limit switches are checked individually. If the X-maximum limit is hit, then ZACTION is checked to see if the calibration flag ($03) is set. If set, then the system is executing a calibration sequence. Since during a calibration, the destination was previously exaggerated to ensure a limit would be reached, the destination is now corrected to be XMAX. DESTX now holds the calibrated, final destination. At UPDATE the present X-position is stored as XMAX plus one. This is done because the limit switches are physically located one step outside the "active" area of the warehouse. All four limit switches
FIGURE 6-16A  Limit Switch Routine Flowchart
FIGURE 6-16B  Limit Switch Routine Flowchart
FIGURE 6-16C  Limit Switch Routine Flowchart
**LIMIT SWITCH CHECK & CORRECTION**

- First check switches, if one hit then stop
- inertia by pausing, then determine which hit
- and update present position.
- If ZACTION = 3 then calibration is in effect,
- change destination to MIN or MAX.
- Else leave destination unchanged.
- Limit reached = LOW at P10, P11, P12, or P13

```
LIMITSW  LDAA  PORT1
CMPA  #$00001111  ; Was a limit hit?
BCC  XMOTOR  ; No, take next step

INERTIA  LDX  #$65000
STOPPER  DEX  ; Yes, pause .4 sec

* Check limits one at a time - correct if hit

XMAXLIM  LDAA  PORT1
ANDA  #$00000001  ; Poll XMAX limit
BNE  UPDATE  ; Correct pres position

YMAXLIM  LDAA  PORT1
ANDA  #$00000010  ; Poll YMAX limit
BNE  UPDATE1  ; Correct pres position

XMINLIM  LDAA  PORT1
ANDA  #$00000100  ; Poll YMIN limit
BNE  UPDATE2  ; Correct pres position

YMINLIM  LDAA  PORT1
ANDA  #$00001000  ; Poll YMIN limit
BNE  UPDATE3  ; Correct pres position

* FIGURE 6-17 Limit Switch Routine Listing
```
are acted on in a similar manner.

To summarize the limit routine, as the system operates and a limit is reached, the destination is only altered if a calibration sequence is underway. Otherwise the destination is unchanged. In either case the system will start seeking that destination. Because the absolute coordinates of each limit switch is known to be one step outside of the minimum-maximum area, the present position is changed to reflect this location.

At the end of the limit switch routine, a jump is executed to XMOTOR, continuing the motor movement. This overall loop, from the phase calculation through the limit switch check, accomplishes one complete step of the stepper motors. A typical command will execute this loop several hundred times, and will finally exit to the HANDSHK routine when both destinations have been reached.

As its label suggests, the HANDSHK routine is responsible for the handshake sent to the host terminal. (figure 6-18). First, the queue is checked to see if there is room for several commands. Ample space permitting, the GO232 handshake will be sent to the host. Otherwise, the STOP232 handshake will be sent. The handshake is output through PORT2 bit 4, not as serial data; but rather, as a static control line (more in Chapter 9). For efficiency reasons, the host is
**FIGURE 6-18** Handshake Routine Flowchart and Listing

---

### Flowchart Diagram

1. **F**  
   - HANDSHK
   - **YES**
   - **IS QUEUE 85% FILLED?**
   - **NO**
   - GET THE GO HANDSHAKE
   - **GET THE STOP HANDSHAKE**
   - SEND HANDSHAKE TO HOST
   - **G**

---

### Listing

- HANDSHK LDA QENTRYS
- CMPA #QFILLED
- BCC FULL
- LDA #GO232 * Go - Send data
- BRA CTSDSR
- FULL LDA #STOP232 * Stop sending data
- CTSDSR STA PORT2

---

**Note:**

- HANDSHAKE TO HOST ROUTINE
- Permits serial transmission if queue open.
- If 85% filled (QFILLED), stop sending.
- Control sent to DSR & CTS lines of host:
  - GO = P24 LOW
  - STOP = P24 HIGH
prevented from sending anything when the queue is more than 85% full. As will later be seen, the queue can be completely filled (100 commands) in the serial communications routine, when a large block of data is sent.

The placement of this handshake routine prior to the ZAXIS routine has another purpose. Because the host will usually send data immediately upon receiving the handshake, the SCI interrupt will usually occur during the ZAXIS routine. This is the preferred place to service such an interrupt, since the motors are done moving. If data is sent during motor movement, the motors will simply be stopped until the transmission is finished.

Next encountered is the ZAXIS routine. Responsible for controlling the piston and electromagnet, this routine actually drops the piston and "grabs" onto or "releases" the blocks (figures 6-19, and 6-20). Physically, the piston is always up, and at the desired destination upon entry into this routine. The magnet is already on or off, due to the last move being a Pick Up or Put Down, respectively. First, ZACTION is checked to determine if this is part of a calibration sequence ($03). If it is, then most of the routine is jumped over, and no piston or electromagnet actions occur. This saves time because there is no reason to drop the
FIGURE 6-19A  2-Axis Routine Flowchart
FIGURE 6-19B  Z-Axis Routine Flowchart
Start of Z-Axis Routine

ZACTION holds info on type of action to be performed:

* 0 = Rest [same as put down, drops piston]
* 1 = Pick Up [drop piston w/mag off, mag on at bottom]
* 2 = Put Down [drop piston, mag same, off at bottom]
* 3 = Calibration [piston stays up, magnet stays same]

If no calibration has been performed for AUTOCAL moves, then one is squeezed in after the next Put Down, or Rest. No commands are lost in this process.

ZAXIS
LDA ZACTION
CMFA #03
* Calibrate ?
BEQ ANYQ
Yes, skip piston drop
LDA PORT3
ANDA #11111110
* Drop Piston
STAA PORT3
* Magnet same
LDAA DWNTIME
%
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
*
* -- FIGURE 6-20 -- Z-Axis Routine Listing
piston when calibrating the motors.

For the three other cases (Rest, Pick Up, Put Down), P30 is driven low, controlling the piston to drop. Next, a delay is executed which allows the piston to completely drop. At this point, the magnet is turned off (P31), and the last calibration counter is incremented. If the command is a Pick Up, then the magnet is energized (P31). If however, the command is not a Pick Up, then a test will be made to see if an automatic calibration is due. If more than AUTOCAL moves have occurred since the last calibration, then a calibration sequence will be instigated before the next command is taken from the queue.

Three observations will now be made. First, the auto-calibration procedure is skipped over on a Pick Up command to avoid unnecessary handling of the "blocks". In this case, auto-calibration will take effect at the completion of the next command. Second, there is actually no difference between Rest and Put Down. Both commands drop the piston with the magnet in the same state - set by the last command. They both turn the magnet off at the bottom - even if it was already off. They are treated as separate commands for illustrative purposes only, and Rest can be thought of as a "spare"-in case additional functions are desired in the future (see Chapter 10). Third, by the end of the ZAXIS
routine, the move is considered completed. With the exception of the calibration command, the piston is down at the destination and a new command can be executed. Notice that the Pick Up command merely "grabs" the block, but does not actually lift it. In all cases, lifting the piston is taken care of in the PREMOVE routine, which was already discussed.

Before the next command can be taken from the queue, a test is performed to determine if any commands are actually in the queue, or if it is empty. Starting at ANYQ, a twenty-five second loop is entered. Inside this loop, the variable QENTRYS is repeatedly checked for a non-zero condition. Since QENTRYS reflects the number of commands in the queue, zero indicates the queue is empty. If a command exists, or one is placed in the queue during this time, then a branch is taken to POPQ - where the command will be fetched. Otherwise, the loop falls through to the SNOOZE routine.

As its name implies, the SNOOZE routine reduces the power expended by the system between "long" intervals of non-use (figures 6-21, and 6-22). The current PORT3 outputs are stored in accumulator B, and then duplicated in accumulator A. From there, the LED (P35) is turned off - simulating the vacuum source being turned off. Next, the piston is controlled to drop (P30), and a delay is executed, ensuring the piston's complete drop.
FIGURE 6-21  Snooze Routine Flowchart
SNOOZE ROUTINE

* Save current outputs.
* Turn magnet off after dropping piston.
* Lower power consumption.
* Look for command in queue.
* Restore all outputs if command arrives.

E2F9 D6 06
E2FB 17
E2FC 8A 20
E2FE 84 FE
E300 97 06
E302 B6 FF E2
E305 CE 36 B0
E308 09
E309 26 FD
E30B 4A
E30C 26 F7
E30E 96 06
E310 8A 0E
E312 97 06
E314 86 EF
E316 97 03
E318 96 89
E31A 27 FC
E31C D7 06

SNOOZE LDAB PORT3  * Save current outputs
TBA
ORAA $00100000  * LED off
ANDA $11111110  * Piston down
STAA PORT3
LDA= DWTIME
OUTLP4 LDX $14000  * Wait for piston drop
INLP4 DEX
BNE INLP4
DECA BNE OUTLP4
LDA= PORT3
ORAA $00011110  * Mag off, low power
STAA PORT3
LDA= $GO232  * Send GO handshake
STAA PORT2
QTEST LDAA QENTRYS  * Any in queue?
BEQ QTEST  No, keep looking
STAB QTEST  Yes, restore outputs

FIGURE 6-22  Snooze Routine Listing
Now the magnet can safely be turned off (P31), and the stepper motors are taken out of the full power mode. The motors will still actively hold their position, but at greatly reduced input power. When the piston is down, the vacuum source has maximum airflow through it. Also by reducing unnecessary heat generation in the motors, component lifetime is increased.

Next, the GO232 handshake signal is again asserted, to be sure the host can send data. At QTEST a loop is continuously executed until a command appears in the queue. Upon detecting a command, the pre-snooze PORT3 outputs are asserted, and the system "wakes up". By checking QENTRYS for a non-zero number, as opposed to simply waiting for an interrupt, all host demands will be serviced, but only a valid, executable command in the queue will cause a "wake up". The next macro-command can now begin as if SNOOZE never occurred.

Before discussing the POPQ routine, a look at the command storage format is warranted. Because every column and row in the warehouse lies in the range of zero through seven, each three-byte command can be compressed into one byte - with no loss of information. This compression takes place in the SCI routine. For now, all we need to know is the format (figure 6-23). The two high-order bits (7,6) contain the Z-axis action to be performed. These two bits can accommodate four
COMPRESSED COMMAND FORMAT

<table>
<thead>
<tr>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Z msb</td>
<td>Z lsb</td>
<td>X msb</td>
<td>X lsb</td>
<td>X msb</td>
<td>Y msb</td>
<td>Y lsb</td>
<td>Y lsb</td>
</tr>
</tbody>
</table>

**Z-field:** (Bits 7, 6) → Action at desired location.

- 0 0 → Rest
- 0 1 → Pick Up
- 1 0 → Put Down
- 1 1 → Calibrate

**X-field:** (Bits 5, 4, 3) → Column of desired action.

- 0 0 0 → Column # 0
- 0 0 1 → Column # 1
- 0 1 0 → Column # 2
- 0 1 1 → Column # 3
- 1 0 0 → Column # 4
- 1 0 1 → Column # 5
- 1 1 0 → Column # 6
- 1 1 1 → Column # 7

**Y-field:** (Bits 2, 1, 0) → Row of desired action.

- 0 0 0 → Row # 0
- 0 0 1 → Row # 1
- 0 1 0 → Row # 2
- 0 1 1 → Row # 3
- 1 0 0 → Row # 4
- 1 0 1 → Row # 5
- 1 1 0 → Row # 6
- 1 1 1 → Row # 7

FIGURE 6-23 Compressed Command Format
actions: 00 = Rest, 01 = Pick Up, 10 = Put Down, and 11 = Calibrate. The next three bits (5,4,3) hold a binary representation of the desired column number (X-direction) where the action is to be performed (0-7). Likewise, the final three bits (2,1,0) hold the desired row number (Y-direction). This compressed format allows 100 bytes of precious internal RAM to store what otherwise would consume 300 bytes.

Knowing a command is in the queue, the POPQ routine begins (figures 6-24, and 6-25). If a SCI interrupt was allowed to disrupt the queue shifting, an incoming command might be placed in the wrong queue space. So, the interrupt mask is temporarily set to prevent the serial communication routine from calling while the queue is being shifted. Then the next command is taken from the top of the queue, and stored in TEMPLO. Now the entire queue is shifted upward by one position, with the higher RAM being overwritten by the entry directly beneath it. After shifting QENTRYS number of RAM locations, the number of queue entries is adjusted, and the interrupt mask is cleared. In the worst-case situation of a maximum queue (100 commands), the interrupts are disabled for only 0.002 seconds. This execution time suggests that the entire queue could be shifted thirty-six times without the risk of a serial communications overrun.
FIGURE 6-24A  Pop Queue/Expansion Routine Flowchart
CALIBRATE COMMAND?

YES

EXPAND Y FIELD INTO ABS COORS

SAVE AS NEXT Y DESTINATION

EXPAND X FIELD INTO ABS COORS

SAVE AS NEXT X DESTINATION

FIGURE 6-24B Pop Queue/Expansion Routine Flowchart
**Queue grows downward from QTOP.**

**QENTRYS is number of commands stored.**

**Each byte holds compressed commands:**

**Bits 7 & 6 hold piston action (Z).**

**Bits 5,4 & 3 hold column (X direction).**

**Bits 2,1 & 0 hold row (Y direction).**

---

**FIGURE 6-25 Pop Queue/Expansion Routine Listing**
At **EXPAND**, the compressed command originally taken from the queue top is loaded into accumulator B. Then two double-accumulator left shifts are performed. The results leave the $Z$-action field in the lower two bits of accumulator $A$. This is the "expanded" form of the $Z$-field, and the entire byte is stored in $ZACTION$.

Next, a test is made to see if the command is a calibrate. If it is, then the other two fields are unnecessary, and a long jump will be taken to $SMCALB$. The contorted logic involving $JUMPER$ is necessary due to the limited branching range. Now accumulator $A$ is again cleared, and the remaining compressed command is double shifted to isolate the $X$-field. After storing the $X$-field, the $Y$-field is shifted into accumulator $A$. Multiplying the row number ($Y$), in accumulator $A$, times the distance between adjacent column centers ($YDSTCEN$) results in a 16-bit number held by the double accumulator ($A:B$). By adding the $YMIN$ dimension, we have now fully expanded the $Y$-field, and converted its column number into an absolute coordinate in the warehouse. This 16-bit coordinate is stored as the new destination in the $Y$-direction ($DESTY$). Similarly, the $X$-field is expanded, converted, and stored as the new $X$-destination. Program execution jumps to $PREMOVE$, and the new command begins.
The next block of instructions is the serial communications interrupt routine (figures 6-26, and 6-27). This routine is called when a complete byte has been received from the host. The first instruction starts at BYTE1, where a loop is entered. Inside this loop a test is made to determine if a byte has been received. The test is considered false if neither the receive data register full flag (RDRF), nor the over-run framing-error flag (ORFE) is set. The loop continues looking for a byte, until after about 0.1 seconds, it is then safe to assume that no byte is coming, and the loop terminates with a jump to ENDSCI. When the interrupt is originally called, a byte will obviously be waiting, but as will be seen later, the loop is necessary for eventually terminating the interrupt.

Assuming a byte was received, the loop is exited, and the receive data register (RECDATA) is loaded into accumulator A. The received byte is then compared to the ASCII codes for "R", "U", and "D". If a match is found, then the appropriate Z-action code is stored in accumulator A for later compression. Since a valid first byte was received, a branch to get the next byte is taken to BYTE2. If however, no match has been found yet, then the received byte is checked for "C". This calibrate macro-command does not use the column and row fields, so the compressed format can be immediately
FIGURE 6-26A  Serial Communication Interrupt Flowchart
FIGURE 6-26B  Serial Communication Interrupt Flowchart
IS FIRST BYTE '<' ?

ALREADY AT MIN SPEED ?

DECREASE STEPPING SPEED

IS FIRST BYTE '=' ?

SET SPEED TO NORMAL (100 s/s)

FIGURE 6-26C  Serial Communication Interrupt Flowchart
FIGURE 6-26D  Serial Communication Interrupt Flowchart
FIGURE 6-26E  Serial Communication Interrupt Flowchart
Figure 6-26F  Serial Communication Interrupt Flowchart
SERIAL COMMUNICATION INTERRUPT ROUTINE

* Check for valid ASCII code, compress, store in queue*

---

**SERIAL COMMMUNICATION INTERRUPT ROUTINE**

BYTE1 LDX #10000 * Looked long enough ?

LOOK1 BNE JUMPER2 Yes, end serial comm.

JUMPER2 JMP ENDS1 No, keep waiting

CMPA #$00111111 * RDRF or DPE flag ?

BLS LOOK1 No, look more

LDAA RECDATA Yes, byte received

* * Wheel of fortune section*

E174 81 52 REST CMPA #'R * Is it a capital R ?

BNE UP No, test next letter

E178 86 00 LDAA #0 Yes, Z-action = 0

CALBRT CMPA #'C * Capital C ?

BNE CLEARQ No, next test

E190 86 C0 LDAA #$11000000 Yes, set Z-field

CLEARQ CMPA #24 Skip X,Y bytes

E192 7E 42 JJMP STOREENQ * Ctrl-X ?

E195 86 01 CMPA #0 * Yes, clear queue

E197 26 05 BRA BYTEX Done with first byte

LDAA $FF No, next test

E199 7F 00 89 CALBRT BNE BYTEX Done with first byte

E19C 81 3E CMPA #'>' * Greater than symbol ?

BNE SLOWER No, test next char

E19F 81 88 LDAA SPEED Yes, alter step speed

CLEARQ CMPA #255

E201 96 8A CMPA #0 * Already maximum speed

E205 27 AC INC SPEED * Increase speed

E207 7A 00 8A CMPA #'< * Get another 1st byte

CMPA #0 * Less than symbol ?

E20A 81 00 BNE BASE No, next test

E20B 96 8A LDAA SPEED Yes, alter speed

E20F 26 0B CMPA #0 * Already minimum speed

E211 96 8A BEQ BYTEX * Decrease speed

E213 86 01 CMPA #'=' * Get another 1st byte

E21B 27 AC DEC SPEED * Equal symbol ?

E21F 81 00 BRA BYTEX No, next test

E223 81 3D CMPA #1 * Set to base rate

E226 89 00 LDAA #127 Yes, set to base rate

E22A 97 8A STAA SPEED * Get another 1st byte

E22C 97 8A BRA BYTEX

STOP CMPA #'1 * Percent sign ?

BNE BYTEX No, Get another 1st

E23A 86 06 LDAA PORT3 Yes, stop system

E23B CA 0C TAB ORAB #$10000110

E23C D7 06 STAB PORT3 * Low power motors

E23D D6 11 BYTEX LDAA TRCSR * Keep looking at all

E23E C1 3F CMPA #$00111111 serial, if any, until

E23F 23 FA LDAA BYTEX caret (ASCII $5E)

E240 D6 12 LOOKE CMPA #$1 * found ?

E242 06 CMPA #0 * No, keep looking

E244 97 06 BNE BYTEX Yes, restore outputs

E247 87 06 STAA PORT3 * Get another 1st byte

E249 7E 63 JMP BYTEX

FIGURE 6-27A Serial Communication Interrupt Listing
**Start second byte receive sequence**

| E3E2 CE 27 10 | BYTE2 | LDX | $10000 |
| E3E5 09      | LOOK2 | DEX |
| E3E6 27 55   | BEQ   | ENDSCI |
| E3EA C1 3F   | LDAB  | TRCSR |
| E3EB 23 F7   | CMPB  | #$00111111 |
| E3EE D6 12   | BLS   | LOOK2 |
| E3F0 C1 30   | CMPB  | #$0   |
| E3F2 25 EE   | BCS   | BYTE2 |
| E3F4 C1 37   | CMPB  | #$7   |
| E3F6 22 EA   | BHI   | BYTE2 |
| E3F8 C0 30   | SUBB  | #$48  |
| E3FA 88      | ASLB  | ACB   |
| E3FB 88      | ASLB  | ACB   |
| E3FC 88      | ASLB  | ACB   |
| E3FE 88      | ASLB  | ACB   |
| E3FF 05      | ASLD  | A:B   |
| E400 05      | ASLD  | A:B   |
| E401 05      | ASLD  | A:B   |

**Waited too long**

**Start third byte receive sequence**

| E402 CE 27 10 | BYTE3 | LDX | $10000 |
| E405 09      | LOOK3 | DEX |
| E406 27 35   | BEQ   | ENDSCI |
| E408 D6 11   | LDAB  | TRCSR |
| E40A C1 3F   | CMPB  | #$00111111 |
| E40C 23 F7   | BLS   | LOOK3 |
| E40E D6 12   | LDAB  | RECDATA |
| E410 C1 30   | CMPB  | #$0   |
| E412 25 EE   | CMPB  | #$7   |
| E414 C1 37   | BHI   | BYTE3 |
| E416 22 EA   | SUBB  | #$48  |
| E418 C0 30   | ASLB  | ACB   |
| E41A 88      | ASLB  | ACB   |
| E41B 88      | ASLB  | ACB   |
| E41C 88      | ASLB  | ACB   |
| E41E 88      | ASLB  | ACB   |
| E41F 05      | ASLD  | A:B   |
| E420 05      | ASLD  | A:B   |
| E421 05      | ASLD  | A:B   |

**Looked long enough?**

**ACCA now holds compressed command**

*Store command at queue bottom*

| E422 7C 00 89 | STOREQ | INC | QENTRIES |
| E425 D6 89    | LDAB   | QENTRIES |
| E427 50       | NEG   | $0000 |
| E428 CE 00 00 | LDX   | X     |
| E42B 3A       | ABX   | STAA  |
| E42C A7 00    | STAA  | 0,X   |

**Store at queue bottom**

**Get more commands?**

| E42E 96 02    | LDAA  | PORT1 |
| E430 84 20    | ANDA  | #$00100000 |
| E432 27 13    | BEQ   | QUIT   |
| E434 96 89    | LDAA  | QENTRIES |
| E436 81 64    | CMPA  | #QMAX  |
| E438 24 09    | BCC   | NOROOM |
| E43A 7E E3 63 | JMP   | BYTE1  |

**End if Halt waiting**

**Queue totally filled?**

**No, get another**

**Resume where left off**

*Queue > 85% filled?*

*No, leave handshake*

*Yes, stop handshake*

*Normal exit sequence - Handshake status*

| E43B 96 89    | ENDSCE | LDAA  | QENTRIES |
| E43F 81 55    | CMPA   | #QFILLED |
| E441 25 04    | BCS    | QUIT   |
| E443 86 FF    | NOROOM | LDAA  | #STOP232 |
| E445 97 03    | STAA   | PORT2  |
| E447 3B       | QUIT   | RTI    |

*Resume where left off*
stored in accumulator A and a jump to STORENQ executed, bypassing the search for column and row characters.

If a first-byte match has still not been found, several special macro-commands are checked for, and if found, they are acted on immediately. These macros are only valid as "first bytes"; and when completed, searching continues for a "first byte" again. An ASCII byte representing "cntrl-X" will cancel all commands in the queue by clearing the QENTRYS counter. The greater-than symbol (>) will cause the stepping motors to move faster. By incrementing the SPEED variable, the stepping rate delay will be decreased, resulting in a small speed increase (about 0.25 steps/second). Since SPEED is originally set at 127, this special macro can be repeated for accumulative results. To avoid one-byte rollover, as well as excessive stepping rates, an attempt to increase SPEED above 255 is ignored. Similarly, a less-than symbol (<) slows down the stepper motors, and an equal symbol (=) restores the normal, or base speed to the system.

Another special macro is a percent sign (%). Designed for a temporary stop of the system, it initiates a loop that ignores all characters until a caret (^) is sent to resume system operations. To accommodate an indefinite stop, the motors are put in the low power mode until resuming.
Notice that all special macros are "two-fingered" characters to eliminate accidental use, and they take effect immediately - bypassing the queue. If a first-byte match was still not found, then the character is simply ignored, and a jump to BYTE1 continues the search for a valid first byte. Keeping in mind the loop at BYTE1, if no more characters are received, then the routine begins termination (ENDSCI).

At BYTE2 the first byte has already been received, and incoming characters are checked until a valid column number arrives (0-7). If no more characters are received, then a branch to ENDSCI will begin termination. The ASCII code arrangement allows a simplified 0-7 range check. Subtracting 48 from this code converts valid characters into column numbers. By shifting the column number left five times, and double shifting four more times, the column number becomes the X-field of a compressed command. Notice that the Z-field was previously loaded into ACCA by the first byte section, and is shifted also. Next, the third byte is likewise received, checked for validity, converted, and shifted into the Y-field position.

At this point the three-byte macro-command has been compressed into one byte, and resides in ACCA, ready to be stored in the queue. At STORENQ the queue size is adjusted and the twos-complement of QENTRYS is formed to
produce an offset from the queue top. The compressed
command is then stored at this queue bottom.

Next, a check is performed to ensure the absolute
maximum size of the queue (QMAX) is not exceeded. If
QMAX commands are already in the queue then the host is
sent the "stop" handshake, and the interrupt returns.
Also, a test of the halt switch status (P15) is
performed to see if a manual halt has been requested
(more in HALT routine section). If the queue still has
any space remaining, then a jump to BYTE1 is taken, and
the search for more characters is continued.

ENDSCI is the target of the three loops that look
for incoming bytes. When the host's data transmission
has ceased, the program will jump to ENDSCI. The queue
size is again checked to send the STOP232 handshake if
QMAX is reached. The RTI instruction is executed,
ending the interrupt, and program execution continues
where it left off. It should be mentioned that no
provision is made for receive error corrections. If an
overrun, framing, or parity error occurs, the erroneous
byte will be read in the usual manner. If the byte
happens to be a valid code occurring in the valid
sequence, then the wrong command will be executed. A
worst-case execution time calculation shows that the SCI
routine requires 0.00013 seconds to process a byte and
start looking for another. This suggests a baud rate
greater than 80,000 would be possible, without the risk of overrun or framing errors. Obviously, by transmitting at much lower rates, the concern about errors is practically nil.

The HALT routine allows another way for the system to be temporarily stopped, manually instead of by the host (figures 6-28, and 6-29). In the case of an emergency, a "worker" can hit the "panic button" to halt the system. Pressing the normally open, momentary contact, switch pulls the IRQ1 and P15 input lines low. An interrupt vector to the HALT routine is taken.

First, the current handshake is pushed on the stack and a "stop sending" handshake is sent to the host. Then the piston is lifted, the motors enter the low-power mode, and the alarm is sounded. Lifting the piston is performed in case the emergency was a crushed worker. After a delay for switch bounce, the halt switch (P15) is polled. Because the switch is checked for first a closure, and then a closure-release sequence, it can be made to act like either a momentary, or non-momentary switch. The user can press and quickly release the switch to halt, then press and release again to resume. Alternatively, by holding the switch down, a halt will be effected until the switch is released.

The original outputs, which were held in ACCB, are then restored. A delay loop is executed if the piston
ISRQ1 INTERRUPT

MALT

READ AND SAVE ACTIVE HANDSHAKE

SEND STOP HANDSHAKE

MAKE 2 COPIES OF PRESENT OUTS

PISTON UP ALARM ON

4 SEC DELAY

MOTORS HOLD IN LOW POWER MODE

POLL P15 READ P15 FOR HALT SW STATUS

A14

B14

FIGURE 6-28A Manual Halt Interrupt Flowchart
FIGURE 6-28B  Manual Halt Interrupt Flowchart
**HALT ROUTINE - Temporary Stop**

* Switch thrown on warehouse floor triggers IRQ1 and pulls P15 low.

---

HALT
LDAA PORT2
PSHA
LDAA #STOP232
STAA PORT2
LDAA PORT3
TAB
ORAA #00000001
ANDA #11101111
STAA PORT3
LDX #65535

*ACCA = ACCB
*Piston up
*Alarm on

* .4 sec delay for switch bounce and motor braking
*Low power motors

STALL3
DEX
BNE STALL3
ORAA #00001100
STAA PORT3
POLL15
LDA PORT1
ANDA #00100000
BNE POLL15
LDX #14000

*Halt switch down?
*No, keep looking
*Yes, Switch bounce delay

STALL4
DEX
BNE STALL4
P15POLL
LDA PORT1
ANDA #00100000
BEQ P15POLL
LDX #14000

*Switch up?
*No, keep looking
*Yes, Switch bounce delay

STALL5
DEX
BNE STALL5
ANDB #00000001
BNE ENDHALT
STAB PORT3
LDAA DWTIME
OUTLP5
LDX #14000

*Restore pre-halt outs
*Was piston down?
*No, end halt routine

OUTLP5
DEX
BNE INLP5
DECA
BNE OUTLP5

*Inner loop = .1 sec

ENDHALT
PUL
STAA PORT2
RTI

*Restore handshake
*End Halt, resume

---

**FATAL ERROR ROUTINE - caused by address error or catastropic event. Non-recoverable**

---

FATAL
LDAA #11111111
STAA PORT4
LDAB PORT3
ORB #00001111
STAB PORT3
LDAA #STOP232
STAA PORT2

*Motors off
*Alarm on, LED on
*Mag same, piston up

ERROR
NOP
BRA ERROR

*Send stop handshake
*Stuck in loop

---

**FIGURE 6-29 Halt and Fatal Error Interrupt Listings**
was originally down. At ENDHALT, the original handshake is restored from the stack, the interrupt ends, and normal execution resumes.

The IRQ1 interrupt was chosen over the non-maskable interrupt (NMI) because the IRQ1 will not directly interrupt the SCI routine, which could potentially miss a byte if not serviced when the "go" handshake is active. To speedily recognize the halt request when in the SCI routine, the P15 input line is checked before getting another "first byte" (typically 0.3 seconds maximum). This also prevents accidental setting of both the manual halt, and host terminal stop command.

A better use of the NMI interrupt is for the detection of catastrophic events. Although the hardware on this model does not exist, possible uses include low vacuum, power failure, fire, and earthquake detectors. By using the TRAP and several otherwise unused interrupts, a computer malfunction can be controlled before any physical damage is done. The TRAP interrupt has priority over all other interrupts except RESET. TRAP is initiated if the microcomputer fetches an undefined instruction, or attempts to address non-existing memory. Once a computer error of this type is detected, the worst case is assumed, and a system shutdown started. Also, by assigning the unused
interrupts with a common vectoring address, more, potentially undetectable, errors can be caught.

All of the above-mentioned catastrophic events or errors are considered fatal, and no attempt to resume normal operations will be made. Beginning at FATAL (figures 6-29, and 6-30), the motors are turned completely off to conserve remaining power, in case a power failure is imminent. Then the alarm is turned on, and every attempt is made to conserve vacuum pressure. By lifting the piston, this vacuum is prevented from outside venting, and a possibly "crushed worker" can escape. The host is then sent a stop handshake—eventually alerting it of a problem by not accepting any commands. The microcomputer then enters an endless loop.

At this point the only undiscussed routine is the demonstration loader (DEMO?). Optionally, it loads an internal 100 command demonstration into the queue—without a host terminal (figures 6-9, and 6-10).

Beginning at DEMO?, a user-selected switch (P17) is checked to see if a demonstration is requested. This switch is only checked once, immediately following the initializations. Loading the demonstration is done by keeping track of a two-byte address pointer in TEMPHI, and TEMPLO. This pointer sequences through the commands stored in high EPROM, while the queue is loaded from top
Figure 6-30 Fatal Error Interrupt Flowchart

Unused INTs. → Opcode Trap INT → Fatal

- Turn Motors Completely Off
- Turn Alarm On
- Lift Piston Magnet = Same
- Send Stop Handshake to Host
- Non-Escapable Loop
to bottom. This allows the demonstration listing to follow logically from top to bottom. The demonstration command's storage must start with the label "STDEMO", and end with "ENDDEMO". A maximum of 100 compressed commands can be used.

Looking at the demonstration listing (figure 6-31), the three fields (Z,X,Y) can clearly be distinguished—each given by one octal digit. This demonstration arranges the blocks in a pattern of the letters "0 U" stacked three high. Once the queue is loaded, the system operates in exactly the same manner as before. The significance of the demonstration capability is two-fold. First, it allows for system illustration and testing to be performed without a host terminal—asserting its "stand-alone" design. Secondly, as will be discussed in Chapter Ten, the capability exists for many 300-byte "super macro-commands" to be resident in firmware and executed with a single keystroke at the host terminal. Putting all the previously mentioned software routines together, we obtain a complete program. This program is responsible for every control function of the warehouse model. By carefully designing the software, the need for external electrical hardware is dramatically reduced — while increasing the flexibility of design changes. In fact, the only external hardware used, is for interfacing purposes.
**STORAGE of DEMO COMMANDS (compressed)**

* Listed in octal for clarity. Format: ZXY
* FIRST DIGIT: 0 = Rest, 1 = Up, 2 = Down, 3 = Calibrate
* SECOND: Column number (X-axis) # 0 - 7 are valid
* THIRD: Row number (Y-axis) # 0 - 7 are valid
* Total $ of commands must be $ < or = 100

<table>
<thead>
<tr>
<th>ORG</th>
<th>STDEMO FCB</th>
<th>8FF60</th>
</tr>
</thead>
<tbody>
<tr>
<td>FF60</td>
<td>FCB</td>
<td>@100</td>
</tr>
<tr>
<td>FF60 40</td>
<td>FCB</td>
<td>@227</td>
</tr>
<tr>
<td>FF61 97</td>
<td>FCB</td>
<td>@100</td>
</tr>
<tr>
<td>FF62 40</td>
<td>FCB</td>
<td>@227</td>
</tr>
<tr>
<td>FF63 97</td>
<td>FCB</td>
<td>@100</td>
</tr>
<tr>
<td>FF64 40</td>
<td>FCB</td>
<td>@227</td>
</tr>
<tr>
<td>FF65 97</td>
<td>FCB</td>
<td>@100</td>
</tr>
<tr>
<td>FF66 40</td>
<td>FCB</td>
<td>@217</td>
</tr>
<tr>
<td>FF67 8F</td>
<td>FCB</td>
<td>@100</td>
</tr>
<tr>
<td>FF68 40</td>
<td>FCB</td>
<td>@217</td>
</tr>
<tr>
<td>FF69 8F</td>
<td>FCB</td>
<td>@100</td>
</tr>
<tr>
<td>FF6A 40</td>
<td>FCB</td>
<td>@217</td>
</tr>
<tr>
<td>FF6B 8F</td>
<td>FCB</td>
<td>@100</td>
</tr>
<tr>
<td>FF6C 40</td>
<td>FCB</td>
<td>@206</td>
</tr>
<tr>
<td>FF6D 86</td>
<td>FCB</td>
<td>@100</td>
</tr>
<tr>
<td>FF6E 40</td>
<td>FCB</td>
<td>@206</td>
</tr>
<tr>
<td>FF6F 86</td>
<td>FCB</td>
<td>@100</td>
</tr>
<tr>
<td>FF70 40</td>
<td>FCB</td>
<td>@206</td>
</tr>
<tr>
<td>FF71 86</td>
<td>FCB</td>
<td>@100</td>
</tr>
<tr>
<td>FF72 40</td>
<td>FCB</td>
<td>@205</td>
</tr>
<tr>
<td>FF73 85</td>
<td>FCB</td>
<td>@100</td>
</tr>
<tr>
<td>FF74 40</td>
<td>FCB</td>
<td>@100</td>
</tr>
<tr>
<td>FF75 85</td>
<td>FCB</td>
<td>@205</td>
</tr>
<tr>
<td>FF76 40</td>
<td>FCB</td>
<td>@100</td>
</tr>
<tr>
<td>FF77 85</td>
<td>FCB</td>
<td>@205</td>
</tr>
<tr>
<td>FF78 40</td>
<td>FCB</td>
<td>@100</td>
</tr>
<tr>
<td>FF79 8C</td>
<td>FCB</td>
<td>@214</td>
</tr>
<tr>
<td>FF7A 40</td>
<td>FCB</td>
<td>@100</td>
</tr>
<tr>
<td>FF7B 8C</td>
<td>FCB</td>
<td>@214</td>
</tr>
<tr>
<td>FF7C 40</td>
<td>FCB</td>
<td>@100</td>
</tr>
<tr>
<td>FF7D 8C</td>
<td>FCB</td>
<td>@214</td>
</tr>
<tr>
<td>FF7E 40</td>
<td>FCB</td>
<td>@100</td>
</tr>
<tr>
<td>FF7F 94</td>
<td>FCB</td>
<td>@224</td>
</tr>
<tr>
<td>FF80 40</td>
<td>FCB</td>
<td>@100</td>
</tr>
<tr>
<td>FF81 94</td>
<td>FCB</td>
<td>@224</td>
</tr>
<tr>
<td>FF82 40</td>
<td>FCB</td>
<td>@100</td>
</tr>
<tr>
<td>FF83 94</td>
<td>FCB</td>
<td>@224</td>
</tr>
<tr>
<td>FF84 40</td>
<td>FCB</td>
<td>@100</td>
</tr>
<tr>
<td>FF85 9D</td>
<td>FCB</td>
<td>@235</td>
</tr>
<tr>
<td>FF86 40</td>
<td>FCB</td>
<td>@235</td>
</tr>
<tr>
<td>FF87 9D</td>
<td>FCB</td>
<td>@235</td>
</tr>
<tr>
<td>FF88 40</td>
<td>FCB</td>
<td>@100</td>
</tr>
<tr>
<td>FF89 9D</td>
<td>FCB</td>
<td>@100</td>
</tr>
</tbody>
</table>

* Demonstration routine that arranges blocks to spell "O U" in letters stacked three blocks high.

* User must supply blocks entered at column 0, row 0.

* Note auto-calibration after 50th move.

* Example:

```
...........put down at
:\::<.......column 3
::<.......row 5
::<:::
```

**FIGURE 6-31A** OU Demo Commands Listing
FIGURE 6-31B  Ou Demo Commands Listing

ENDDMEO FCB $277  * Last command

* Last command

* 50th command

Errors: 0

TABS(8), MARGINS=0,80 ,LAST PRINTED COL=54,55 OPEN

March 28, 1988

LAST PRINTED COL=54,55 OPEN
Even the stepper motor drive sequences are generated in software, without the need for additional sequencing chips. All communications with a host terminal are acted upon internally, and up to one hundred commands can be stored inside the one-chip microcomputer. The host terminal is then free to go about other tasks, while the automated warehouse computer independently executes all of the commands in the background.

The total program, with all its bookkeeping, communications, and control duties, occupies a mere 1500 bytes. Since this one-chip microcomputer can accommodate up to 8192 bytes of "internal" EPROM, it is clear that these one-chip designs have the capability of performing many complex functions with not only, speed and accuracy; but also, at a lower component count and cost. There can be no doubt that the one-chip microcomputer will play an important role in designs of the future.

Several additional features that were not fully developed in this system are discussed in the Enhancements and Modifications section of Chapter Ten. Provisions were made in the general software design to allow for these features to be added, but time constraints limited the complete testing and debugging by the author.
VII HOST TERMINAL/COMPUTER

Acting as the "manager" or link to the outside world, the host terminal is the user's interface to the automated warehouse. The following chapter will discuss the physical and electrical nature of the "standard" communication link between these two devices. After an introduction to the RS-232 standards, a brief discussion of a typical terminal used as a host will be given. Finally, the benefits of using a host computer will be explored. More information regarding the actual format of the accepted macro-commands is given in chapter nine.

The Electronic Industries Association (EIA) RS-232 is the standard interface in terms of the present computer and communications world. This standard covers four areas: (1) the mechanical characteristics of the interface, (2) the electrical signals across the interface, (3) the function of each signal, (4) subsets of signals for applications. A common misconception is that the DB-25 connector is defined in the standard: this is not true. The mechanical standards only define the assignment of signals to the connector pins, which piece of equipment has the female connector (the DCE), the recommended cable length (maximum 50 feet), and the maximum cable capacitance (2500 picofarads).
The DB-25 connector has become nearly universally associated with RS-232C, but it is not defined in the standard, and IBM and some other manufacturers use a different connector on much of their equipment [13].

The most important part of the standard is the functional and electrical sections. Functionally, the pins are named with respect to the data terminal equipment (DTE), which in our case is the host terminal or computer. At the other end of the connection is the data communications equipment (DCE), which is the model's microcomputer. Now we will look at the individual pins that are, or most likely could be, used to interface a host to the warehouse model (figure 7-1).

![Diagram of RS-232C Connector](image-url)

**FIGURE 7-1** RS-232C Connector
Pin 1 is the protective ground, which is intended to connect one end of the shield, if shielded cable is used. It is usually connected only to the DTE (host), and can be used to minimize interference in high noise environments. Pin 7 is the common (ground) reference for all signals between the DTE and DCE, which include data, control and timing signals.

Pins 2 and 3 pass all of the serial data, like a two lane highway. The DTE (host) transmits on pin 2, and receives on pin 3. Since all pins are named with respect to the DTE, these pins are named "Transmitted Data" (TD), and "Received Data" (RD), respectively. In order to respect the DTE's names, the DCE (warehouse) must, conversely, transmit on pin 3 and receive on pin 2.

Pins 4 and 5 are "Request to Send" (RTS) and "Clear to Send" (CTS). The RTS signal is simply an indication that the terminal is turned on and waiting. Although not used in this warehouse model system, RTS could be used to provide full two-way (duplex) communications between the host and warehouse.

Pins 6 and 20 correspond to "Data Set Ready" (DSR), and "Data Terminal Ready" (DTR). The DSR pin indicates that the DCE is up and running - ready to communicate, if needed. Similarly, the DTR pin is asserted whenever the DTE is operating and ready to communicate. Pin 8 is
designated the "Received Line Signal Detector" (DCD), many DTE's require this signal before they will transmit or accept data. In most non-modem applications, pin 8 is usually tied to pin 20 - so that the DTE will self-handshake itself whenever it is turned on (figure 7-2).

![Diagram of Self-Handshake Scheme]

**FIGURE 7-2 Self-Handshake Scheme**

Electrically speaking, the RS-232C signal levels are set for noise considerations. For all serially transmitted data, a positive voltage between 5 and 15 volts on pins 2 or 3 with respect to pin 7 (common ground), represents a logic 0 level (space). A negative voltage between -5 and -15 volts on these pins represents a logic 1 (mark). However, the voltage polarities are reversed for the same logic levels on the control lines. For this reversed scheme, most RS-232
buffers invert the data on the transmitting and receiving ends.

Several additional terms exist in the world of serial communications. The baud rate is a common unit of signal speed that equals the number of signalling events per second. Depending on the number of other bits transmitted along with the data, the baud rate may not equal the number of bits per second. In addition to the seven or eight data bits, start and stop bits are sent to help synchronize the receiver with the incoming bit stream. In the No-Return to Zero format (NRZ), an idle data line is given by a continuous logic 1 (mark). Since the start bit is always a 0 (space), and stop bits are always 1 (mark) - the identity of individual data bytes is easily confirmed. Obviously, additional stop bits simply duplicate an idle line. Most systems use one stop bit, but by using two instead, the receiver section has more time to process the last byte before the next arrives.

The parity bit, if used, is a simple error detection scheme. For odd parity, the transmitter adds each data bit and appends an additional (parity) bit, such that the combined sum of data and parity bits is odd. Conversely, the even parity sum is even. By adding the bits received, single bit errors can be detected by typical receivers. A framing error is
detected when the stop bit (always 1) is not received. An overrun error occurs when an incoming byte has been fully received but the last byte has not yet been taken out of the receiver buffer.

Conforming to all of the previously mentioned "standards", there still exist several methods of accomplishing successful serial communications. First, the methods currently used by the automated warehouse model's computer will be discussed, then an alternate scheme will be brought forth.

The automated warehouse microcomputer's serial communication interface has a limited number of formats available. The decision was made to transfer data at 150 baud - mainly for maximum data integrity. The start bit is internally sampled several times to determine if it is valid, and then each bit thereafter is sampled by a short pulse at the center of each bit time. By using a short sample at the center of a bit time, the chance of a random noise error is reduced. The higher the sampling frequency, the lower the chance of noise coinciding at that instant. For that reason, the lower baud rates are more noise-immune. No parity is used by the microcomputer, probably for efficiency: no-parity schemes save about ten percent of the transfer time. Only one stop bit is needed to signal the end of the
data bits. As previously mentioned, the NRZ format is followed, producing a logic 1 idle transmit line.

The electrical connections for our system are as shown in figure 7-3.

![Diagram of serial connections]

**FIGURE 7-3  Serial Connections in This System**

The host terminal is the DTE, while the warehouse is the DCE. The protective ground (Pin 1) at the DTE only, is connected to the cable's shield if one is used. Data is transmitted from the host to the warehouse through pin 2. Pin 3 is presently unused, but would naturally transfer data from the warehouse to the host. Pins 5 and 6 (CTS and DSR) are tied together at the host end, allowing one wire from the warehouse to control both host inputs. Of course these pins could be tied
together at the warehouse end if it is simpler. The third mandatory line is the signal ground, pin 7.

Half-duplex (one-way) communications are enabled when the DSR and CTS pins are asserted (+15v) by the warehouse. The host is then permitted to send any data on pin 2, as long as DSR and CTS are asserted. This assertion from the warehouse model, is from a simple output port (P24) - through the RS-232 inverting buffer. The microcomputer software drives P24 low (G0232) to make the assertion. Likewise, when P24 is driven high (STOP232), the CTS and DSR lines are not asserted, and the host will not send any data. This control-handshake line (DSR and CTS) is static, not serial. The P24 output is used as a traditional output, although it could easily be used to serially transmit data as will be discussed next.

The current handshake scheme works well in this case, but is far from the only method that could be used. Another method to control the flow of data is for the DCE (warehouse) to send special characters for pause and resume over the other serial data line (pin 3) to the DTE (host). Since this requires an "above average intelligence" on the part of the host, this method is really only suited for computers acting as host terminals. A software driver can be written to recognize the special characters and act appropriately
Typically the ASCII codes DC1 and DC3 ($17$ and $19$) are used, but are called XON and XOFF. These characters correspond to control-Q (resume) and control-S (pause) on the standard keyboard. The warehouse could send XOFF when the queue is 85% full and XON when below 65% full. This software handling of the flow control seems appropriate if the warehouse is to send messages back to the host, since the DTE receive data line (pin 3) will be in use anyhow.

With this method, the DSR and CTS controls can be hardwired to the DTR pin, and full-duplex communications can be employed with only three lines (TX, RD, Signal Ground) (see figure 7-5). Since the microcomputer's P24 output can be configured to act as a serial transmitter instead of its presently traditional output, the switch over should be a matter of software at both ends (see Chapter Ten). In fact, several widely available software drivers exist for an IBM PC to interface with a serial printer.

To assure compatibility with the "dumbest" of terminals, the software flow control method was not further pursued. A potential problem with either method is the loss of data when stopping transmission. When the warehouse queue is full, the "stop sending" control signal is activated. The host must receive this handshake in sufficient time to avoid sending any more
FIGURE 7-4  Flow Control Handshake Flowchart

FIGURE 7-5  Flow Control Handshake Electrical Connections
characters. If one extra character "dribbled" out before communications actually stopped, this character would be isolated in the SCI routine and could possibly lead to the loss of a macro-command. This condition would be further aggravated when sending a character to stop the data flow.

A careful examination of the SCI routine (figures 6-26F, and 6-27B) reveals why this is not a problem in the current system. First, the "third byte" processing is completed before a decision is made regarding the handshake. If the "stop sending" control activates the DSR and CTS pins of the host, it is done in enough time to avert the host from sending any new data. In fact, this control signal is sent sometime between the last half of the last character's stop bit and the beginning of the next character's start bit (figure 7-6). At 150 baud, this half-bit time allows the execution of 2200 microcomputer machine cycles to occur, easily enough time to get the handshake out. Secondly, if the additional character happened to be sent, it would very likely be an unnecessary character such as a blank space between macro-commands. In all of the various stages of testing the system, no serial communications errors of this nature were observed.

As previously discussed, any device conforming to the RS-232C specifications, and capable of
asynchronously producing ASCII characters at 150 baud, 8 data bits, no parity, and 1 or more stop bits, is a candidate for interfacing to the automated warehouse. Possible devices include terminals, computers, microcomputers, modems, and certain keyboards.

Terminals usually record all keystrokes in an internal, one line, buffer and release that buffer upon registering a carriage return or line feed. The released buffer is flushed to the output buffer. This output buffer could be only one to three lines long (256 bytes maximum). The output buffer constantly looks at the control lines for permission to send its data. If the warehouse has stopped accepting characters, because
it is full, the terminal buffer could overflow causing unpredictable results. This unlikely event should be watched out for if the user is planning on typing long and fast.

Another situation to avoid is the sending of only part of a macro-command followed by a pause (0.1 second or more). The warehouse will, upon not receiving the last character(s), discard the other part of the macro-command and continue its operations until the remaining partial command is received. This partial command will also be discarded since it makes no sense without the previous parts. The best solution is to simply finish a command or series of commands before pressing the carriage return. Normally, the entry of commands is straightforward, and the user is recommended to read Chapter Nine for more information regarding the actual format of macro-commands.

USING A COMPUTER AS HOST

Throughout the development and testing, a computer was used as the host. The actual machine used was an IBM PC-XT, but any computer capable of the previously mentioned RS-232 format will suffice. An intelligent terminal of this type affords the user many conveniences. One such convenience is the ability to store frequently used groups of macro-commands on a
floppy disk - then simply printing that particular file instead of manually retyping it each time. By using a print buffer, huge files can be dumped to the warehouse instantly - freeing the computer to go on to other tasks while the warehouse works in the background, taking more commands from the buffer as queue space permits. Of course, for smaller files or manual input, the warehouse queue can accommodate up to 100 commands as usual.

A BASIC-language program can check the user's inputs for obvious errors and allow the redefining of the function keys. For instance, issuing two Pick Up commands with no Put Down command between them, would be simple to detect by a BASIC-language program.

Several of these conveniences have been incorporated on a floppy disk that accompanies the warehouse model. The following discussion highlights some of the more useful MS-DOS commands that will probably be needed. For more information regarding these DOS commands consult an appropriate source [14].

**Initializing a Serial Port**

A> MODE COM1: 15, N, 8, 1, P

This command sets up serial communication port #1 for 150 baud, no parity, 8 databits, 1 stop bit, and continuous retries. The last delimiter (P) causes the computer to continuously check for the "Go" handshake

```bash
A> MODE COM1: 15, N, 8, 1, P
```
from the warehouse. Without it, the message "Timeout error" may appear, after a second, when the warehouse queue is full.

Temporary Redirection
A> TYPE TEST#1 > COM1:
This command sends disk file "TEST#1" to the serial port instead of the screen. The redirection is only in effect for this one line.

Permanent Redirection
A> MODE LPT1: = COM1:
This command permanently redirects all future printing operations at LPT1 to the serial port (COM1). Once this is in effect a file can be sent as follows:
A> PRINT TEST#1
The print spooler's default buffer size (512 bytes) can be increased to 3000 bytes by the following:
A> PRINT LPT1: /B:3000
Note: This last command can only be made once, and should be made immediately after booting for best results.
Using Batch Files

Assuming a standard text disk file containing twenty greater-than symbols ( > ) has been created and named "FASTER.IMM". The following batch file named "SPEEDUP.BAT" will output the macro-file by simply typing "SPEEDUP" at the DOS prompt.

    TYPE FASTER.IMM > COM1:
    REM
    REM     * * File Sent * *

Note: The last two lines are optional, but reassuring to the operator that the task was completed. Of course, if the warehouse queue is full and a print buffer is being used, these twenty characters are actually in the host computer's buffer - not the warehouse yet. For this reason, it is often advantages not to use a print buffer. Then the "** File Sent **" message will only appear when the warehouse has actually taken the macro-commands.

IBM PC Disk Included

The accompanying disk is rather obvious to the familiar user, therefore, only a brief description will be given here. The CONFIG.SYS file increases the Cntrl-Break checking for a "break" command, and sets up a 128k virtual disk in RAM, for less disk accesses. The AUTOEXEC.BAT file sets up the serial communications
format, and then copies the necessary files to the virtual disk (C:). The user is given a few options on the screen, and inputs can begin.

OPERATIONS WITHIN BASIC

A fairly primitive Basic language program has been included to illustrate some of the entry checking, and function key redefinition features that are possible with a host computer. This program is functional, and can be run in GWBASIC, but it is mainly a guide to some future improvements (see Chapter Ten). Simple string comparisons can be performed to match possible user inputs to a valid macro-command. Checks for suspicious entries are performed. A typical suspicious entry might be two successive Pick Up commands. The macro-command strings are output through Basic's LPRINT command. To use this program, the line printer (LPT1) must be previously redirected to the serial port (COM1). The Basic commands allowing direct output to a serial port seem to be intolerant to long delays that could occur if the warehouse queue is full. There probably is a way around this "glitch", but the LPRINT command works well enough to cause little concern.
VIII TESTING

A series of tests for accuracy, precision, and the general proper operation of the warehouse system was carried out. The individual tests, comments, and results are shown below. In nearly every test, the outcome was impressive - no detectable inaccuracies. In fact, the only deviations were due to mechanical tolerances and are considered acceptable.

Test #1 - Stacking Accuracy

1. Calibrate to column 0, row 0.
2. Perfectly stack three blocks at (0,0).
3. Send: U00 D77, U00 D15, U00 D51
   U77 D00, U15 D00, U51 D00

RESULTS: Stack is disassembled and reassembled with no measurable errors.

TEST #2 - Long Move Accuracy

1. Calibrate to column 0, row 0.
2. Place one block at (0,0).
3. Send: U00 D77
   R00 R70 R47 R04 R40
   U77 D00

RESULTS: Block moved, then piston only moved in star pattern, then block moved to origin with no measurable errors.

Total time (incl. piston) = 99 seconds
Average time per move = 99/9 = 11 seconds
Test #3 - Many Short Moves

1. Calibrate to column 0, row 0.
2. Place three blocks at (0,0), one at a time.
3. Send: U00 D01, U00 D10, U00 D11
   U10 D02, U01 D20, U11 D22
   U02 D00, U22 D00, U20 D00

RESULTS: Each block moved to two close locations, and stacked at origin with no error.

Total time = 144 seconds
Average time per move = 144/18 = 8.0 sec.

Test #4 - Base Stepping Rate

1. Send: R00 R77 R00
2. Record only moving time (no piston).
3. Repeat as necessary.
4. Assume 945 steps from col. 0 to col. 7
   Assume 588 steps from row 0 to row 7

RESULTS:

<table>
<thead>
<tr>
<th>FROM TO</th>
<th>STEPS</th>
<th>TIME (SEC)</th>
<th>STEPS /SEC</th>
<th>SEC/STEP</th>
</tr>
</thead>
<tbody>
<tr>
<td>col 0 col 7</td>
<td>945</td>
<td>9.39</td>
<td>100.6</td>
<td>.0099</td>
</tr>
<tr>
<td>col 7 col 0</td>
<td>945</td>
<td>9.49</td>
<td>99.6</td>
<td>.0100</td>
</tr>
<tr>
<td>row 0 row 7</td>
<td>588</td>
<td>5.90</td>
<td>99.7</td>
<td>.0100</td>
</tr>
<tr>
<td>row 7 row 0</td>
<td>588</td>
<td>5.87</td>
<td>100.2</td>
<td>.0100</td>
</tr>
</tbody>
</table>

COMMENTS: Base rate is 100 steps per second
Differences within test measurement error.
Test #5 - Increased Stepping Rate (using > macro)

1. Send: = >>>>>>>> ( 50 total ) >>>>>>>> R00 R77 R00
2. Repeat only R77 R00 as necessary.

RESULTS:

<table>
<thead>
<tr>
<th>FROM</th>
<th>TO</th>
<th>STEPS</th>
<th>TIME (SEC)</th>
<th>STEPS /SEC</th>
<th>SEC/STEP</th>
</tr>
</thead>
<tbody>
<tr>
<td>col 0</td>
<td>col 7</td>
<td>945</td>
<td>8.45</td>
<td>111.8</td>
<td>.0089</td>
</tr>
<tr>
<td>col 7</td>
<td>col 0</td>
<td>945</td>
<td>8.45</td>
<td>111.8</td>
<td>.0089</td>
</tr>
<tr>
<td>row 0</td>
<td>row 7</td>
<td>588</td>
<td>5.09</td>
<td>115.5</td>
<td>.0087</td>
</tr>
<tr>
<td>row 7</td>
<td>row 0</td>
<td>588</td>
<td>5.20</td>
<td>113.1</td>
<td>.0088</td>
</tr>
</tbody>
</table>

COMMENTS: Increase in speed of about .25 steps/sec for each ">" macro-command (above base).

Test #6 - Decreased Stepping Rate (using < macro)

1. Send: = <<<<<<<< ( 50 total ) <<<<<<<< R00 R77 R00
2. Repeat only R77 R00 as necessary.

RESULTS:

<table>
<thead>
<tr>
<th>FROM</th>
<th>TO</th>
<th>STEPS</th>
<th>TIME (SEC)</th>
<th>STEPS /SEC</th>
<th>SEC/STEP</th>
</tr>
</thead>
<tbody>
<tr>
<td>col 0</td>
<td>col 7</td>
<td>945</td>
<td>10.26</td>
<td>92.1</td>
<td>.0109</td>
</tr>
<tr>
<td>col 7</td>
<td>col 0</td>
<td>945</td>
<td>10.34</td>
<td>91.4</td>
<td>.0109</td>
</tr>
<tr>
<td>row 0</td>
<td>row 7</td>
<td>588</td>
<td>6.53</td>
<td>90.1</td>
<td>.0111</td>
</tr>
<tr>
<td>row 7</td>
<td>row 0</td>
<td>588</td>
<td>6.30</td>
<td>93.3</td>
<td>.0107</td>
</tr>
</tbody>
</table>

COMMENTS: Decrease in speed of about .17 steps/sec for each "<" macro-command (below base).

Test #7 - Accuracy at Increased Speed

1. Calibrate at (0,0).
2. Place a block at (0,0).
3. Send: = >>>>>>>>>>>>>>>>>>> ( 50 total ) >>>>>>>>>>> >>>>>>>>>>> 
   U00 D77, U77 D00 
4. Check if limit switches are hit while executing.

RESULTS: No limits hit or any measurable error.
Test #8 - Observation of Demonstration (O U)

1. Calibrate at (0,0) and load demo from host or restart system with demo switch on.
2. Feed blocks at (0,0).
3. Don't include initial calibration in exec. time.

RESULTS: Perfectly aligned stacks spell "O U".

   Number of commands     = 103
   Time to complete       = 1095 seconds
   Avg. total time/move   = 10.6 seconds
                           (including up/down, autocalibrate)

Test #9  X-Maximum Limit Switch Position

1. Calibrate at (0,0).
2. Place block at (0,0).
3. Send: U00 D50, C, U50 D00

RESULTS: No measurable error. * (see note below)

Test #10  Y-Maximum Limit Switch Position

1. Calibrate at (0,0).
2. Place block at (0,0).
3. Send: U00 D05, C, U05 D00

RESULTS: No measurable error. * (see note below)

* NOTE: If a limit switch is maladjusted, then an error can be introduced on a calibration to that switch. One X-step is about .025", and a Y-step is about .035".
IX OPERATING INSTRUCTIONS

Because the automated warehouse handles all of the actual movements, the user simply needs to turn the device on and "dictate" what and where. The basic operation is straightforward and outlined in the accompanying flowchart (figure 9-1), and macro-command examples. Additional information regarding the host terminal (or computer) is given in chapter seven.

The macro-commands can be broken into two main groups: (1) One and three character sequential commands, (2) One character immediate commands (see figure 9-2). The first group is used most often, supplying the what and where information. The first character is what action is to be performed. The options include R (Rest), U (Pick Up), and D (Put Down). The next two characters are where the action will be performed. The column number (0-7), is followed by the row number (0-7). With the exception of additional macro-commands, other characters are ignored. The user is free to use spaces, commas, and any lowercase letters to increase readability. These three 3-character macros are placed in the warehouse queue and executed sequentially. Another sequential macro is the character "C" (calibrate). It is similarly placed in the queue,
USER OPERATING INSTRUCTIONS

* HOST=150 BAUD
  8 DATA I STOP
  NO PAR, CONT RETRY

START

DO YOU WANT THE DEMO?

DO YOU WANT THE DEMO?

YES

TURN DEMO SWITCH ON

NO

TURN DEMO SWITCH OFF

TURN WAREHOUSE ON, CONNECT RS232

ARE YOU FINISHED?

ARE YOU FINISHED?

YES

TURN OFF AND UNPLUG WAREHOUSE

NO

ENTER COMMANDS FROM HOST

UNPLUG RS232 SERIAL LINK

TURN OFF HOST

DONE

* OPTIONAL IF DEMO ONLY.

FIGURE 9-1 User Operating Instructions Flowchart
SEQUENTIAL MACRO-COMMANDS

[Action] [Column #] [Row #]

[Action] is:  
R - Rest or drop piston only  
U - Pick Up Block  
D - Put Down Block

[Column #] is:  
0-7 column number (X-axis) where action will be performed.

[Row #] is:  
0-7 row number (Y-axis) where action will be performed.

C - Calibrate to nearest corner. One character. Sequential.

IMMEDIATE MACRO-COMMANDS

One Character

CNTRL-X - Clears queue. Effective immediately when received.

% - Temporary stop until ^ received. Immediate.

^ - Resume if stopped. Else ignored. Immediate.

> - Increase speed. Immediate. Accumulative.

< - Decrease speed. Immediate. Accumulative

= - Set to default speed. Immediate.

All other characters are ignored by warehouse.

Figure 9-2 Macro-Command Summary
but unlike the previous commands, this macro is only one character in length. By using the "C" macro, the user is instructing the warehouse to execute all previous commands, and then calibrate itself at the nearest corner. The column and row numbers are not needed, and if any are supplied they will be ignored.

The next group of macro-commands are all one character in length, and take effect immediately under most conditions. These macros are system functions and are not placed in the warehouse queue. They are only recognized if they occur as the "first byte" in a move sequence, and ignored elsewhere. The "first byte" simply means that all three bytes of the previous move have been found and the sequence has started overlooking for a first byte. For example, in the sequence:

`U 1 2, D 3 4, C, R 0 0U55D66`

the first bytes here are U, D, C, R, U, and D. The spaces and commas are ignored.

By invoking the macro "Ctrl-X" (one character), the queue is immediately cleared. The "%" (percent symbol) causes the warehouse to immediately halt all operations until a "^" (caret) is received. The ">" (greater than symbol) increases the stepper motor speed by about .25 steps per second. The "<" (less than symbol) decreases the speed. These last two speed commands are accumulative. Finally, the "=" (equal
symbol) negates all speed changes and returns the system to the base (default) speed.

As previously mentioned, the system functions take place immediately and affect the whole system. This includes all previously queued commands and the one currently executing. If however the queue is 85% full and accepting no more commands, the warehouse can not act on the immediate command until it is received. For this reason, it is in the user's interest to keep the serial communication line open.

Several examples are shown in figure 9-3. The first group illustrates the many ways to achieve the same results. The user is free to enter the macro-commands in the most convenient form. The shorter syntax form is quicker to enter and transmit, but may lack readability.

Next, in figure 9-3, is a complete example that exchanges two blocks. Of course this task could be typed on one line, but each new line only adds two characters to be transmitted (carriage return, and line feed).

Finally, figure 9-3 shows some common errors due to improper user input. The first example points out that lower case letters are ignored. The characters: space, 3, and 6, would also be ignored since they are not valid first bytes. The next example fails because of the
Examples of successful entries and effects

U 1 2
U12
UC>8=.#.?!CRUD 2
Pick Up at Column 1,Row 2

U12, D36
U12 & D36
U12, D 3 6, C

>>>>, U12, D 3 6
U12, >>>>>, D 3 6
U12, D 3 6, >>>>

Pick up at col 1,row 2
Up at 1,2 then Down 3,6
Up,Down,and then calibrate
Increase Speed then Up,Down

Example: Exchanging two blocks at (1,2), (4,5)

U 12, D 3 6
U 45, D 12
U 36, D 45

Common Errors

<table>
<thead>
<tr>
<th>COMMANDS</th>
<th>EFFECT</th>
</tr>
</thead>
<tbody>
<tr>
<td>d 36</td>
<td>Totally ignored (lower case &quot;d&quot;)</td>
</tr>
<tr>
<td>CALIBRATE 00,U12</td>
<td>Calibrate, Rest(0,0), Up(1,2)</td>
</tr>
<tr>
<td>&gt;&gt;&gt;U12, D36, =</td>
<td>No speed change, (= sign cancels)</td>
</tr>
<tr>
<td>U12, D36, Cntrl-x</td>
<td>Commands entered then cleared</td>
</tr>
<tr>
<td>U 1 &gt;&gt; 2</td>
<td>U12 but no speed change</td>
</tr>
</tbody>
</table>

Figure 9-3 Macro-Command Examples
capital "R" in CALIBRATE. Use the form: Calibrate, if the whole word must be spelled out. The next two examples show the nature of the immediate commands. The last immediate speed command will typically negate all previous speed macros. The fourth example points out a possible misuse of the cntrl-x command to clear the queue. The last example emphasizes the importance of observing the proper sequencing of the "first byte". The erroneous "U 1 >> 2" macros would work if entered as "U 1 2 >>" or ">> U 1 2".
MAINTENANCE AND TRANSPORTATION

Maintenance

Virtually no periodic maintenance should be required, the only exception being the vacuum piston. In the event of a problem, the following tips should prove useful.

Vacuum Piston

If the piston becomes sluggish, then additional lubrication is probably required. First, rule out vacuum supply leakage. A properly functioning piston should easily be retracted by human suction (mouth). Do not use any liquid lubrications (Teflon, WD-40, etc.), the surface tension created is too strong. Instead use only dry silicone or white graphite.

1. Put paper underneath piston.
2. Apply dry silicone to piston walls.
3. Also apply through top pipe nipple.
4. Exercise piston manually.

If cylinder looks darkened, then a significant amount of aluminum has probably gummed up inside the assembly. In this case, take the key screw out, and carefully disassemble the unit. Clean by wiping piston and cylinder walls with mineral spirits or equivalent (not
lacquer thinner). Then apply dry lubricant and reassemble.

Flexible Cables

Replace X-sled cables with one low temperature silicone cable (extra flexible) if available. Otherwise duplicate present arrangement. If desired, the shells on each connector can be reused – only the pins need to be replaced (.062"). Consult appendix for wiring.
Note: If cable has become kinked from long non-use, then work bend out by applying some heat.

Chains

Do not lubricate chains. Installing a chain is best done by catching about three links in the clamps. Then roll the chain onto the gears. The three larger chains are exactly 60 inches (600 links). This is the minimum length required, and a few more links would be helpful.

Blocks

If the 20-gauge galvanized sheet metal becomes loose, then apply contact cement to remedy. Make sure metal is fairly flat beforehand. Some pieces may have a curvature due to the cutting shear.
Limit Switches

The switches will slide if adjustment is necessary. Do not attempt unless there is a severe misalignment. Alignment is probably necessary if the limits are often hit when accessing row or column 0. Since each motor step is about 0.025 inches, adjustment is a touchy endeavor. If attempted, be sure that the sled "bottoms out" on the frame-stop, and not on the actual switch body. Use the lockwashers and thread cement (Loctite) to prevent vibrations from loosening adjustments.

Removing Electrical Cards

Do not pull on any components. Grasp only the card and pull straight up. Be sure to install the cards in their correct slot and orientation.

Replacing Vacuum Blower Motor

1. Remove top of assembly.
2. Disconnect vacuum solenoid connector.
3. Remove screws (2) from base ears.
4. Remove end piece - 7 screws (one on bottom).
5. Disconnect motor wires.
6. Cut silicone motor mount.
7. Remove motor while twisting.
8. Transfer intake breather assembly to new motor.
9. Slide motor in place.
10. Form new silicone motor mount around motor.
11. Replace wires, screws, and connector.
12. Check for leaks around seals.
Changing the EPROM

1. Turn power off and wait one minute.
2. Remove microcomputer card (MB1).
3. Use IC removal tool to separate EPROM from socket.
4. Eprom address range is $E000 - $FFFF
5. Install EPROM carefully (same direction as MCU).
6. Reinsert card into rack.

Transporting the Automated Warehouse Model

1. Move sleds to XMAX, YMIN position.
2. Tie sleds down with wire or nylon cable straps.
3. Retract piston (up) and fix in this position.
4. Wrap power cord in an appropriate space.
5. Warehouse can now be moved with XMIN, YMAX corner up vertically as necessary (doorways).
CONCLUSIONS

All the goals of this project were successfully completed, with very good results. An automated warehouse model was constructed that illustrates the powerful capabilities of a one-chip microcomputer unit and custom designed software. The microcomputer system is a "slave" to a user-operated terminal or computer.

The user enters high level macro-commands that are transmitted serially to the warehouse computer system. These commands are then processed and compressed by the microcomputer to be stored in an internal, one-hundred command, queue. Execution of the commands is carried out in the sequence they were input. Nearly 200 unique macro-commands can be recognized. Several "system" macro-commands are included which allow the physical speed to be changed by the user, stopping of all operations, and clearing the queue.

Since the one-chip microcomputer unit contains many previously separate components, the need for external electrical hardware is dramatically reduced. In fact, the model's only external hardware is for isolation and interfacing purposes. Even the stepper-motor drive sequences are generated in software, eliminating the need for additional motor control chips.

The model mechanically illustrates these control functions, and the accuracy that can be achieved with an
open-loop stepper motor system. The roughly four-foot by
three-foot warehouse model can access sixty-four
locations in eight columns and eight rows; stacked up to
three blocks high - for a total of 192 blocks in a fully
stocked warehouse. The longest move, diagonally from
corner to corner, takes the X and Y-stepper motors
through 945 and 588 steps, respectively. A pneumatic
piston/electromagnet assembly physically "picks" and
"places" the blocks. A typical move is completed in
about ten seconds with no perceptible errors.

The entire microcomputer software, with all of its
bookkeeping, communications, and control duties, uses
only 1500 bytes of code. It is obvious that these one-
chip MCU designs have the capability of performing many
complex functions with not only, speed and accuracy; but
also, at a lower component count and cost.

ENHANCEMENTS AND MODIFICATIONS

Several interesting concepts presented themselves
throughout the development of the automated warehouse.
Some of these ideas could be implemented in this system;
others are better suited for other microcomputer/stepper
motor control systems. The following section will
briefly discuss a few ideas that may interest the reader or spark future enhancements and modifications.

Debugging

As with any software product, micro-computer debugging can be one of the most frustrating chores imaginable! This system proved to be extra ordinarily difficult since it did not have a monitor or standard error reporting. Quite often, the only clue to a problem presented itself briefly and then "crashed" the system—making it nearly undetectable. By stopping the program at various break points, certain problem routines can be identified. One helpful way of stopping program execution is to use the software interrupt instruction (SWI). Then a service routine polls a certain limit switch, or whatever, before continuing.

Example:

```
COREPROG ...
  [Routine 1]
    SWI
  [Routine 2]
    SWI
    JMP COREPROG
SERVSWI     LDAA PORT1
            ANDA #$01
            BNE SERVSWI
            RTI
```
To check interrupt-driven routines, a simple core program can be written that waits for an interrupt (WAI) and then outputs some useful information, which can be read with a logic probe at Port4, for example:

\[
\text{START} \quad \ldots
\]
\[
\ldots
\]
\[
\text{[ INITIALIZE ]}
\]
\[
\ldots
\]
\[
\text{WAIT} \quad \text{WAI} \quad \star \text{Wait for interrupt}
\]
\[
\text{LDA} \quad \text{INFO} \quad \star \text{Get info}
\]
\[
\text{STAA} \quad \text{PORT4} \quad \star \text{Output useful info}
\]
\[
\text{POLLSW} \quad \text{LDA} \quad \text{PORT1}
\]
\[
\text{ANDA} \quad \#$01 \quad \star \text{Loop until switch}
\]
\[
\text{BNE} \quad \text{POLLSW}
\]

\*\* Interrupt Driven Routine \*\*

\[
\text{INTRRRUPT} \quad \ldots
\]
\[
\ldots
\]
\[
\ldots
\]
\[
\ldots
\]
\[
\text{STAA} \quad \text{INFO} \quad \star \text{Pass info}
\]
\[
\text{RTI}
\]

**Using the Spare Output**

A spare output has been provided for many possible uses. Available through Port3 bit 5, it is a typical, completely buffered output, capable of controlling several amperes. This spare output presently drives a LED - which simulates the vacuum source being turned off when in the Snooze mode. It would be relatively simple to control the actual vacuum motor with a triac if so desired. Be sure to provide complete isolation between the electronics and the 120V motor.
Improved Communications

The warehouse computer could send messages to the host by adding one serial communication line. In the event of an error, or whenever the host might request, certain information could be transferred. Besides the message routine, no major software changes would be needed. Port2, bit 4 would be configured as a serial transmitter instead of a simple output. No hardware changes would be needed at the warehouse end.

At the host terminal end, the RS-232 control lines could be changed to a "self-handshake", and some simple software might need to be developed utilizing flow control communications methods. The warehouse would serially send a prearranged character to stop and start the host's sending of data. Similar methods are used with many printers, which send XON (ASCII $19) and XOFF (ASCII $17). Once XOFF is sent, the host will stop sending and monitor all incoming characters (message) until the XON character is received. First, change the GO232 and STOP232 constants to $19 and $17. Then instead of storing the handshake to PORT2 in the PREMOVE routine, simply store the handshake to the transmit register.
Super Macro-Commands

Expanding upon the method used to load the demonstration commands, up to one hundred commands could be invoked with a single "supermacro" at the host terminal. In addition, these supermacros could be placed in the queue, only occupying one byte until they were executed. By eliminating the redundant "Rest" command, 64 distinct super macros could be identified by the Z-field of "00". The remaining 6-bits of the X- and Y-fields could represent supermacro numbers.

The SCI routine, upon recognizing a "S" first byte could take the other two numbers, compress them, and place the entire supermacro in the queue - just as the Rest command is now used. The difference would be in the expansion routine (POPQ). When a given supermacro worked its way to the front of the queue, its Z-field would flag a branch to a supermacro handler routine. Such a routine would determine the supermacro number, based on the X- and Y-fields. Then the appropriate block of commands could be loaded from a prearranged area of extra EPROM. This block loading would be exactly like the demonstration routine, but with different starting addresses for different supermacros.
Possible uses of some supermacros would be: (1) moving an entire column or row to the "loading docks" after executing all previous commands, (2) changing the system speed after a series of commands to accommodate different weight loads, (3) sending a message back to the host after completing a group movement.

Ramping Stepper Motor Rate

Although not necessary in this system, stepper motor ramping is often used to achieve higher system speeds. To accurately accelerate the motors and their mechanical systems to a higher speed, a variable stepping rate is used. Most of the necessary ramping software already exists in the warehouse computer. Recall that the macrocommands altering the speed simply change the timing loop delay between motor steps. Assuming 25-step, linear acceleration and deceleration curves will suffice, then only a few lines of supplemental code will need to be written. Working within the stepping speed delay routine, a two-byte counter can be used to monitor when a "knee" in the curves is reached. For the first 25 motor steps, the variable SPEED can be incremented — increasing the stepping rate each time. Once the 25th step has been taken, the system will be at full speed. By comparing
both X and Y-destinations with their present positions, the first axis closing the gap to within 25 steps can invoke the deceleration curve. To decelerate, the SPEED variable can be decremented providing a smooth arrival. If a limit is hit, the curves will have to start over. Of course, since both axes are simultaneously controlled, they will share the shortest acceleration/deceleration curve.

**Self Test**

To ensure a properly functioning microcomputer unit, a self test could be performed before starting the "real" program. Excluding some of the system registers, and the output ports, a test for stuck-at-0, and stuck-at-1 faults could easily be accomplished. First, check the address lines by comparing the $FFFF, and $0000 address locations against their known values. Then check the condition code register, accumulators, stack, index register, and RAM. By writing $FF to each location, and then verifying each reads $FF, stuck-at-0 faults can be detected. Similarly, by writing $00 to each location, stuck-at-1 faults can be detected.

Then a check sum of the entire EPROM could be performed. The whole self test could be accomplished in
about a tenth of a second. To check all four limit switches, the queue could be preloaded with two compressed commands, (€266) and (€377), which together with the initial calibration would exercise all four limits.

ENHANCED USER INTERFACE

An improved host program could certainly be developed. As long as the macro-command and serial communication protocols are observed, many possibilities exist, and should be straightforward to develop. Inventory control can be accomplished by using three-dimensional variables (8x8x3). Also, redefining the keyboard function keys, and mouse support are possibilities. Indeed, a completely graphical entry system, using menus and displaying all warehouse locations, is achievable in BASIC.
BIBLIOGRAPHY

CITED REFERENCES


ADDITIONAL REFERENCES


APPENDIX A

COMPLETE SOFTWARE LISTING
* AUTOMATED WAREHOUSE CONTROLLER (Version AWC22) *
* Developed by Steve Ricca, 1987-88 *
* All rights reserved *

* Define assembler labels, constants, variables *
* Source code = 6801 assembly language *

<table>
<thead>
<tr>
<th>Label</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>AUTOCAL</td>
<td>EQU 50</td>
</tr>
<tr>
<td>DATDIR1</td>
<td>EQU $0000</td>
</tr>
<tr>
<td>DATDIR2</td>
<td>EQU $0001</td>
</tr>
<tr>
<td>DATDIR3</td>
<td>EQU $0004</td>
</tr>
<tr>
<td>DATDIR4</td>
<td>EQU $0005</td>
</tr>
<tr>
<td>GO232</td>
<td>EQU $111011111</td>
</tr>
<tr>
<td>PORT1</td>
<td>EQU $0002</td>
</tr>
<tr>
<td>PORT2</td>
<td>EQU $0003</td>
</tr>
<tr>
<td>PORT3</td>
<td>EQU $0006</td>
</tr>
<tr>
<td>PORT4</td>
<td>EQU $0007</td>
</tr>
<tr>
<td>QFILLED</td>
<td>EQU 85</td>
</tr>
<tr>
<td>QMAX</td>
<td>EQU 100</td>
</tr>
<tr>
<td>QTOP</td>
<td>EQU $00FF</td>
</tr>
<tr>
<td>RECDATA</td>
<td>EQU $0012</td>
</tr>
<tr>
<td>RMODECR</td>
<td>EQU $0010</td>
</tr>
<tr>
<td>STOP232</td>
<td>EQU $11111111</td>
</tr>
<tr>
<td>TRCSR</td>
<td>EQU $0011</td>
</tr>
</tbody>
</table>

* Local Variables - RAM Locations *

<table>
<thead>
<tr>
<th>Label</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>ORG</td>
<td>$0080</td>
</tr>
<tr>
<td>DESTX</td>
<td>RMB 2</td>
</tr>
<tr>
<td>DESTY</td>
<td>RMB 2</td>
</tr>
<tr>
<td>LASTCAL</td>
<td>RMB 1</td>
</tr>
<tr>
<td>PRSXPOS</td>
<td>RMB 2</td>
</tr>
<tr>
<td>PSSYPOS</td>
<td>RMB 2</td>
</tr>
<tr>
<td>QENTRYS</td>
<td>RMB 1</td>
</tr>
<tr>
<td>SPEED</td>
<td>RMB 1</td>
</tr>
<tr>
<td>TEMPHI</td>
<td>RMB 2</td>
</tr>
<tr>
<td>TEMPO</td>
<td>EQU TEMPHI+1</td>
</tr>
<tr>
<td>XPHASE</td>
<td>RMB 1</td>
</tr>
<tr>
<td>YPHASE</td>
<td>RMB 1</td>
</tr>
<tr>
<td>ZACTION</td>
<td>RMB 1</td>
</tr>
</tbody>
</table>
Table of Phase Bits

Bits 7-4 = X motor, Bits 3-0 = Y motor

These must stay in this order for this system

CW= $5, 6, A, 9  CCW= $9, A, 6, 5

Toward minimum  Toward maximum

<table>
<thead>
<tr>
<th>ORG</th>
<th>FFD0</th>
<th>phase sequence #</th>
</tr>
</thead>
<tbody>
<tr>
<td>PHBITS</td>
<td>FCB</td>
<td>%10011001</td>
</tr>
<tr>
<td></td>
<td>FCB</td>
<td>%10011010</td>
</tr>
<tr>
<td></td>
<td>FCB</td>
<td>%10101100</td>
</tr>
<tr>
<td></td>
<td>FCB</td>
<td>%10101001</td>
</tr>
<tr>
<td></td>
<td>FCB</td>
<td>%10101010</td>
</tr>
<tr>
<td></td>
<td>FCB</td>
<td>%10100110</td>
</tr>
<tr>
<td></td>
<td>FCB</td>
<td>%10100101</td>
</tr>
<tr>
<td></td>
<td>FCB</td>
<td>%01101001</td>
</tr>
<tr>
<td></td>
<td>FCB</td>
<td>%01101010</td>
</tr>
<tr>
<td></td>
<td>FCB</td>
<td>%01100110</td>
</tr>
<tr>
<td></td>
<td>FCB</td>
<td>%01100101</td>
</tr>
<tr>
<td></td>
<td>FCB</td>
<td>%01011001</td>
</tr>
<tr>
<td></td>
<td>FCB</td>
<td>%01011010</td>
</tr>
<tr>
<td></td>
<td>FCB</td>
<td>%01010110</td>
</tr>
<tr>
<td></td>
<td>FCB</td>
<td>%01010101</td>
</tr>
</tbody>
</table>

**Dimension Constants**

<table>
<thead>
<tr>
<th>ORG</th>
<th>FFD0</th>
<th>BASESP</th>
<th>63650</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPTIME</td>
<td>FCB</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>XMAX</td>
<td>FDB</td>
<td>2945</td>
<td></td>
</tr>
<tr>
<td>XMIN</td>
<td>FDB</td>
<td>2000</td>
<td></td>
</tr>
<tr>
<td>XDSTCEN</td>
<td>FCB</td>
<td>135</td>
<td></td>
</tr>
<tr>
<td>YMAX</td>
<td>FDB</td>
<td>2588</td>
<td></td>
</tr>
<tr>
<td>YMIN</td>
<td>FDB</td>
<td>2000</td>
<td></td>
</tr>
<tr>
<td>YDSTCEN</td>
<td>FCB</td>
<td>84</td>
<td></td>
</tr>
</tbody>
</table>

**Interrupt Vectors**

<table>
<thead>
<tr>
<th>ORG</th>
<th>FFD0</th>
<th>TRAP</th>
<th>FATAL</th>
</tr>
</thead>
<tbody>
<tr>
<td>TOP</td>
<td>FDB</td>
<td>00</td>
<td></td>
</tr>
<tr>
<td>OCF</td>
<td>FDB</td>
<td>00</td>
<td></td>
</tr>
<tr>
<td>ICF</td>
<td>FDB</td>
<td>00</td>
<td></td>
</tr>
<tr>
<td>IRQ1</td>
<td>FDB</td>
<td>00</td>
<td></td>
</tr>
<tr>
<td>SCI</td>
<td>FDB</td>
<td>00</td>
<td></td>
</tr>
<tr>
<td>IRQ1</td>
<td>FDB</td>
<td>00</td>
<td></td>
</tr>
<tr>
<td>SOFTW</td>
<td>FDB</td>
<td>00</td>
<td></td>
</tr>
<tr>
<td>NMI</td>
<td>FDB</td>
<td>00</td>
<td></td>
</tr>
<tr>
<td>RES</td>
<td>FDB</td>
<td>00</td>
<td></td>
</tr>
</tbody>
</table>

* Must reside here

* Address error

* Serial communications

* Timer (not used)

* Temporary stop

* Catastrophe

* Reset
START PROGRAM - Initialize, set-up serial protocol

* ORG $100
* Start
* Declare stack area
* Queue empty
* Serial protocol
* Stop handshake to host
* P24 = output
* Enable interrupts

** Configure Ports**

* LDA PORT1
  * Port 1 all inputs (8)
* LDA PORT3
  * LED on, others off
* LDA PORT4
  * Port 3 all outputs (8)
* LDA PORT5
  * Port 4 all outputs (8)

** Initialize Stepper Motors**

* CLR XPHASE
  * Motors start at phase 0
* CLR YPHASE
  * Warm up, low power
* LDX XMAX
  * Initial X Position
* LDX YMAX
  * Initial Y Position

** ESTABLISH INITIAL POSITION**

* Assume slightly closer to XMIN, YMIN corner
* forcing calibration to nearest corner (0,0)

** LOADING OF DEMONSTRATION COMMANDS**

* Check if demo requested
* Low at P17 loads demo

* DEMO? LDA PORT1
  * Switch on?
  * Yes, load demo commands
  * Get starting address
  * Store address
  * Update # commands
  * Form 2's complement
  * Calculate Q address
  * Store at Q bottom
  * Load 1 demo command
  * Load 1 demo command
  * Store address
  * Get address
  * Get address
  * Get address
  * Store to next command
  * Done loading?
  * No, get another

Legend:
- ORG
- START
- LDS
- CLR
- STAA
- LDAA
- CLI
- LDD
- ADD
- LSRR
- SUBL
- STD
- STX
- INC
- NEGB
- ABX
- STA
- CPX
- BLS
- ANDA
- BNE
- LDX
SMART CALIBRATE ROUTINE

* Alters destinations to beyond nearest corner

SMCALB LDD XMAX * Change destinations
      ADDD #1234 beyond maximum X
      STD DESTX
      LDD YMAX
      ADDD #1234 beyond maximum Y
      STD DESTY

XCALB LDD XMAX
      ADDD XMIN * Calculate midpoint X
      LSRD
      SUBD FRSXPOS
      BLS YCALB * Past midpoint ?
      * No, change dest. to before minimum X
      SUBD #1234
      STD DESTX

YCALB LDD YMAX
      ADDD YMIN * Calculate midpoint Y
      LSRD
      SUBD FRSYPOS
      BLS ZCALB * Past midpoint ?
      * No, change dest. to before minimum Y
      SUBD #1234
      STD DESTY

ZCALB LDAA #3 * Declare calibration
       STAA ZACTION sequence in effect
       CLR LASTCAL * Calibration counter

PREMOVE ROUTINE

* Get system ready for movement.
* Piston up, motors at full power

PREMOVE LDAA PORT3 * Full power
      ANDA #11110011 * Mag same
      ORAA #00000001 * Piston up
      STAA PORT3
      LDAA UPTIME

OUTLP1 LDX #16250 * Wait for piston up
      INLP1 DEX
      BNE INLP1 * Inner loop = .1 sec
      BNE OUTLP1

-----------------------------
** * MOTOR STEPPING ROUTINE * **
* Compare positions, calculate next phase sequence, * *
* table look up, output, delay * 

*-----------------------------------------------------*
XMOTOR LDX PRSXPOS
CPX DESTX
BCS BEFOREX
BHI PASTX
LDX PRSYPOS
CPX DESTY
BNE YMOTOR
JMP HANDSHK

* * Start Phase Calculation

BEFOREX INC XPHASE
LDD PRSXPOS
ADD $0001
STD PRSXPOS
JMP YMOTOR

PASTX DEC XPHASE
LDD PRSXPOS
SUB $0001
STD PRSYPOS

YMOTOR LDX PRSYPOS
CPX DESTY
BCS BEFOREY
BHI PASTY

BEFOREY INC YPHASE
LDD PRSYPOS
ADD $0001
STD PRSYPOS

XPHASE
PRSXPOS
YO001
PRSXWS
YMOTOR

XPHASE
PRSXPOS
YO001
PRSXWS

Y PHASE
PRSYPOS
YO001
PRSYPOS

* * Put X & Y phase sequences together, forming offset

MERGE LDX #PHBITS
LDB A XPHASE
AND $00000011
ASLB

PASTY DEC YPHASE
LDD PRSYPOS

$00,X
PORT4

* Delay determines motor stepping speed

DELAY2 LDAA SPEED
LDAB #$3
MUL
ADD $BASESP
STD TEMPHI
LDX TEMPHI

STALL2 INX
BNE STALL2

* Base address of bits
* Modulo-4 counter
* ACCB = 0000XX00
* Add to base address
* Modulo-4 counter
* Add to base address
* 1-of-16 bit patterns
* output to motor coils

* Variable speed
* Multiplication factor
* Base speed(100 s/sec)
* Delay loop
LIMIT SWITCH CHECK & CORRECTION

First check switches, if one hit then stop inertia by pausing, then determine which hit and update present position.

If ZACTION = 3 then calibration is in effect, change destination to MIN or MAX.

Else leave destination unchanged.

Limit reached = LOW at P10, P11, P12, or P13.

* * * Check limits one at a time - correct if hit

XMAXLIM LDAA PORT1
    ANDA #$00000001
    CMPA #$00000001
    BCC XMOTOR
    * Was a limit hit?

    CMPA ZACTION
    LDD XMAX
    STD DEStX
    * Poll XMAX limit

    BNE UPDATE
    YMAXLIM LDAA PORT1
    ANDA #$00000010
    CMPA ZACTION
    BNE XMAML
    LDD YMAX
    STD DEStY
    * Poll YMAX limit

    BNE UPDATE
    XMINLIM LDAA PORT1
    ANDA #$00001000
    CMPA ZACTION
    BNE XMINDONE
    LDD XMIN
    STD DEStX
    * Poll XMIN limit

    BNE UPDATE
    YMINLIM LDAA PORT1
    ANDA #$00001000
    CMPA ZACTION
    BNE XMINDONE
    LDD YMIN
    STD DEStY
    * Poll YMIN limit

    BNE UPDATE
    SUBD #$01
    STD PRSXYPOS
    * Correct pres position

    SUBD #$200000100
    YMINLIM LDAA PORT1
    CMPA ZACTION
    BNE XMINDONE
    LDD YMIN
    STD DEStY
    * Correct pres position

    SUBD #$01
    STD PRSXYPOS
    * Correct pres position

    XMINDONE JMP XMOTOR

Start next motor step
HANDSHAKE TO HOST ROUTINE
Permits serial transmission if queue open.
If 85% filled (QFILLED), stop sending.
Control sent to DSR & CTS lines of host:
GO = P24 LOW
STOP = P24 HIGH

HANDSHK  LDA PORT2  BNE QENTRYS
CMFA H101  CMPA H102
BCC PORT2  LDAA H103
LDA PORT2  #GO232
BRA CTSDSR
FULL  LDA PORT2  #STOP232
CTSDSR STAA PORT2

---

Start of Z-Axis Routine

ZACTION holds info on type of action to be performed:
0 = Rest [same as put down, drops piston]
1 = Pick Up [drop piston w/mag off, mag on at bottom]
2 = Put Down [drop piston, mag same, off at bottom]
3 = Calibration [piston stays up, magnet stays same]

If no calibration has been performed for AUTOCAL moves, then one is squeezed in after the next Put
Down, or Rest. No commands are lost in this process.

---

ZAXIS  LDA PORT3
CMFA H01  CMPA H103
BEC PORT3  ANYQ
LDA  PORT3  #00000010
ANAD PORT3  LASTCAL
INCA PORT3  #AUTOCAL
DEC PORT3  #AUTO
ANDA PORT3

* Test if any commands in queue

---

25O PORT3  #250
6185 PORT3  #6185
QENTRYS PORT3  LDA H103
POFQ PORT3  BNE PORT3
INLP3 PORT3  BNE PORT3
OUTLP3 PORT3  BNE PORT3
SNOOZE ROUTINE
   Save current outputs.
   Turn magnet off after dropping piston.
   Lower power consumption.
   Look for command in queue.
   Restore all outputs if command arrives.

SNOOZE LDAB PORT3
   Save current outputs

TBA
   ACCA = ACCB

ORAA #00100000
   LED off

ANDA #11111110
   Piston down

STAA PORT3

LDAA DWNTIME

OUTLP4 LDX #14000
   Wait for piston drop

INLP4 DEX
   BNE INLP4
   Inner loop = .1 sec

DECA

BNE OUTLP4

LDAA PORT3

E516 96 03

QTEST LDAA QENTRYS
   Any in queue?

E318 96 89

B6Q QTEST
   No, keep looking

E31A 27 FC

STAB PORT3
   Yes, restore outputs

E31C D7 06
* Queue grows downward from QTOP.
* QENTRYS is number of commands stored.
* Each byte holds compressed commands:
  * Bits 7 & 6 hold piston action (Z)
  * Bits 5,4, & 3 hold column # (X direction)
  * Bits 2,1, & 0 hold row # (Y direction)

---

**POP OFF QUEUE & EXPAND ROUTINE**

- **Queue grows downward from QTOP.**
- **QENTRYS is number of commands stored.**
- **Each byte holds compressed commands:**
  - Bits 7 & 6 hold piston action (Z)
  - Bits 5,4, & 3 hold column # (X direction)
  - Bits 2,1, & 0 hold row # (Y direction)

---

E31E DF
E31F 96 FF
E320 97 8C
E321 96 89
E325 CE 00 FE
E328 60 00
E32A E7 01
E32C 09
E32D 4A
E32E 26 F8
E330 7A 00 89
E333 0E

**POPQ SEI**
- *Prevent SCI interrupt
- *Save top command
- *Shift entire queue up one position
- *Fix # commands counter
- *Enable SCI interrupt

**EXPAND**

- **Compressed command**
- A:B=00000000:ZZXXXYYY
- Store Z-field
- ZACTION
- Calibrate?
- Yes, skip XY expand
- Shift X-field into A
- Shift Y-field into A
- ACCA = 00000YY
- Dist betw Y centers
- A:B = ACCA times ACCB
- Add Y offset
- Abs. coor. new Y dest
- A:B = ACCA times ACCB
- Dist betw X centers
- Add X offset
- Abs. coor. new X dest
- Start new move
* SERIAL COMMUNICATION INTERRUPT ROUTINE *
* Check for valid ASCII code, compress, store in queue *

BYTE1  LDX  #10000  * Looked long enough ?
       BNE JUMPER2  Yes, end serial comm.
       JUMPER2  JMP ENDSCL  No, keep waiting
       CMPA #100111111  * RDRF or ORFE flag ?
       BLS LOOK1  No, look more
       LDAA RECDATA  Yes, byte received
               * * Wheel of fortune section
       REST  CMPA #R  * Is it a capital R ?
             BNE UP  No, test next letter
             LDAA $0  Yes, Z-action = 0
             BRA BYTE2  Done with first byte
       UP  CMPA #U  * Is it a capital U ?
             BNE DOWN  No, test next letter
             LDAA $1  Yes, Z-action = 1
             BRA BYTE2  Done with first byte
       DOWN  CMPA #D  * Capital D ?
             BNE CALBRT  No, next test
             LDAA #1000000  Yes, set Z-field
       CALBRT  JMP STORENQ  Skip X,Y bytes
       CLEARQ CMPA #24  * Capital C ?
             BNE SLOWER  No, next test
             LDAA SLOWER  Yes, set Z-field
       SLOWER  CMPA #X  * Greater than symbol ?
             BNE SLOWER  No, next test
             LDAA SPEED  Yes, alter step speed
       BASE  CMPA #0  * Already minimum speed
             BEQ BYTE1  * Increase speed
             INC SPEED  * Get another 1st byte
       BRA BYTE1  * Less than symbol ?
       BYTE1  CMPA #<  * Equal symbol ?
             BNE BASE  No, next test
             BRA BYTE1  Yes, set to base rate
       SLOPER CMPA #>  * Get another 1st byte
             BNE BASE  Yes, alter speed
             LDAA SPEED  * Percent sign ?
       BRA BYTE1  No, Get another 1st
       LDAA PORT3  Yes, stop system
       TAB  CMPA #\  * Low power motors
       BNE BYTE1  * Keep looking at all
each, if any, until 
caret (ASCII $5E) is found
       STAB PORT3  * ^ found ?
       ORAB #000011000  No, keep looking
       PORT3  BNE BYTE1  * Get another 1st byte
       BYTE1  CMPA #127  Yes, restore outputs
       STAA PORT3  * Get another 1st byte
       SLOPER  BRA BYTE1  * Get another 1st byte
       BASE  CMPA #\  * Percent sign ?
             BNE BYTE1  No, Get another 1st
       BASE  CMPA #127  Yes, stop system
             STAA PORT3  * Low power motors
             XORAB #000011000  * Keep looking at all
                             serial, if any, until 
caret (ASCII $5E) is found
             PORT3  BNE BYTE1  * ^ found ?
             BYTE1  CMPA #127  Yes, restore outputs
             STAA PORT3  * Get another 1st byte
             JMP BYTE1  * Get another 1st byte

E163 CE 27 10
E166 09
E167 26 03
E169 7E E4 3D
E16C 96 11
E16E 81 3F
E170 23 F4
E172 96 12
E174 81 52
E176 26 04
E178 86 00
E17A 20 66
E17C 81 55
E17E 26 04
E180 86 01
E182 20 5E
E184 81 44
E186 80 02
E188 20 56
E18A 81 43
E18C 26 05
E18E 80 C0
E192 7E E4 22
E194 81 18
E196 26 05
E199 7F 00 89
E19C 20 C5
E19E 81 3E
E1A0 26 0B
E1A2 96 8A
E1A4 81 FF
E1A6 27 8B
E1A8 7C 00 8A
E1AB 20 86
E1AD 81 3C
E1AF 26 0B
E1B1 96 8A
E1B3 81 00
E1B5 27 AC
E1B7 7A 00 8A
E1BA 20 A7
E1BC 81 3D
E1BE 26 06
E1C0 86 7F
E1C2 97 8A
E1C4 20 9D
E1C6 81 25
E1C8 26 99
E1CA 96 06
E1CE 16 67
E1CD CA OC
E1CF D7 06
E1D1 D6 11
E1D3 26 04
E1D5 23 FA
E1D7 D6 12
E1D9 C1 5E
E1DB 26 F4
E1D7 97 06
E1DF 7E E3 63
**Start second byte receive sequence**

187

```
E3E2 CE 27 10  BYTE2 LDX #10000
E3E5 09  DEX
E3E6 27 55  BEQ ENDSCE
E3E8 D6 11  LDAB TRCSR
E3EA C1 3F  CMPB #$00111111  * RDRF or ORFE flags ?
E3EC 23 F7  BLS LOOK2
E3EE BC 12  LDAB RECDATA
E3F0 C1 30  CMPB #$0  * In range of ASCII 0 - 7 ?
E3F2 25 EE  BCS BYTE2
E3F4 C1 37  CMPB #$7
E3F6 22 EA  SHI BYTE2
E3FA C0 10  SUBB $48  * Conversion to decimal
E3FB 58  ASLB  * ACCB = 00000XX0
E3FC 58  ASLB  * ACCB = 000XXX00
E3FD 58  ASLB  * ACCB = 000XXX00
E3FE 58  ASLB  * ACCB = 0XXXXXXX0
E3FF 05  ASLD  * ACCB = XXX00000
E400 05  ASLD  * A:B=00000ZZZ:XX000000
E401 05  ASLD  * A:B=000ZZXXZ:XXX00000

**Start third byte receive sequence**

```

E402 CE 27 10  BYTE3 LDX #10000
E405 09  DEX
E406 27 35  BEQ ENDSCE  * Looked long enough ?
E408 D6 11  LDAB TRCSR
E40A C1 3F  CMPB #$00111111  * RDRF or ORFE flags ?
E40C 23 F7  BLS LOOK3
E40E D6 12  LDAB RECDATA
E410 C1 30  CMPB #$0
E412 25 EE  BCS BYTE3
E414 C1 37  CMPB #$7
E416 22 EA  SHI BYTE3
E418 C0 30  SUBB $48  * Conversion to decimal
E41A 58  ASLB  * ACCB = 0000ZZZ0
E41B 58  ASLB  * ACCB = 000ZZZ00
E41C 58  ASLB  * ACCB = 00ZZZ000
E41D 58  ASLB  * ACCB = 0ZZZ0000
E41E 58  ASLB  * ACCB = ZZZ00000
E41F 05  ASLD  * A:B=0ZZZZYYY:YYYY0000
E420 05  ASLD  * A:B=0ZZZZYYY:YYYY0000
E421 05  ASLD  * A:B=ZZZZZZZZ:00000000

**ACCA now holds compressed command**

**Store command at queue bottom**

```

E422 7C 00 89  STORENQ INC QENTRYS  * # commands in queue
E425 D6 89  LDAB QENTRYS  * Two's complement
E427 50  NEGB
E426 CE 00 00  LDX #$0000
E42B 3A  AXB
E42C A7 00  STAA 0,X  * Store at queue bottom

**Get more commands ?**

```

E42E 96 02  LDAA PORT1
E430 84 20  ANDA #$00100000  * End if Halt waiting
E432 27 13  BEQ QUIT
E434 96 89  LDAA QENTRYS
E436 81 64  CMPA #$MAX  * Queue totally filled ?
E438 24 09  BCC NOROOM  Yes, no room for more
E43A 7E E3 63  JMP BYTE1  No, get another

**Normal exit sequence - Handshake status**

```

E43D 96 89  ENDSCE LDAA QENTRYS  * Queue > 85% filled ?
E43F 81 55  CMPA #$FILLED  * No, leave handshake
E441 25 04  BCS QUIT
E443 86 FF  LDAA #$STOP232  Yes, stop handshake
E445 97 03  STAA PORT2
E447 3B  QUIT RTI  * Resume where left off

******************************************************************************
```
HALT ROUTINE - Temporary Stop

Switch thrown on warehouse floor triggers IRQ1 and pulls P15 low.

-----------------------------------------------------
HALT LDAA PORT2 * Save active handshake
PSHA LDEA #STOP232 * Send stop handshake
STAA PORT2 LDEA PORT3
LDEA PORT3 TAB
ORAA #\$00000001 * Piston up
ANDA #\$11101111 * Alarm on
STAA PORT3 LDX #65535 * .4 sec delay for
LDAA PORT2 switch bounce and
LDAA STALL3 motor braking
ORAA #\$00001100 * Low power motors
STAA PORT3

STALL3 DEX
BNE STALL3
ORAA #\$00000011
Piston up
ANDA #\$11101111
Alarm on
STAA PORT3
STX #965535
delay for
STALL3 DEX
BNE STALL3
P15POLL LDAA PORT1
ANDA #\$00100000 * Halt switch down ?
BNE P15POLL
LDX #14000
Yes,

STALL4 DEX
BNE STALL4
P15POLL LDAA PORT1
ANDA #\$00100000 * Switch up ?
BNE P15POLL
LDX #14000
Yes,

STALL5 DEX
BNE STALL5
STAB PORT3
ANDA #\$00000001 * Was piston down ?
BNE ENDHALT
LDAA DWNTIME

OUTLP5 LDX #14000
Yes, delay for
INLP5 DEX
piston drop
BNE INLP5
DECA BNE OUTLP5

ENDHALT PULA
STAA PORT2 * Restore handshake
RTI * End Halt, resume

-----------------------------------------------------
FATAL ROUTINE - caused by address error or catastrophic event. Non-recoverable

-----------------------------------------------------
FATAL LDAA #\$11111111 * Motors off
STAA PORT4
LDAB PORT3
ANDB #\$111001111 * Alarm on, LED on
STAB PORT3
LDAA #STOP232 * Mag same, piston up
STAA PORT2

ERROR NOP
BRA ERROR * Stuck in loop

*-----------------------------------------------------*
**STORAGE of DEMO COMMANDS** (compressed)

* Listed in octal for clarity. Format: ZXY
* *FIRST DIGIT: 0= Rest, 1= Up, 2= Down, 3= Calibrate*
* SECOND: Column number (X-axis) # 0 - 7 are valid*
* THIRD: Row number (Y-axis) # 0 - 7 are valid*
* Total # of commands must be < or = 100

---

<table>
<thead>
<tr>
<th>FF60</th>
<th>ORG</th>
<th>$FF60</th>
</tr>
</thead>
<tbody>
<tr>
<td>FF60 40</td>
<td>STDEMO FCB</td>
<td>@100</td>
</tr>
<tr>
<td>FF61 97</td>
<td>FCB</td>
<td>@227</td>
</tr>
<tr>
<td>FF62 40</td>
<td>FCB</td>
<td>@100</td>
</tr>
<tr>
<td>FF63 97</td>
<td>FCB</td>
<td>@227</td>
</tr>
<tr>
<td>FF64 40</td>
<td>FCB</td>
<td>@100</td>
</tr>
<tr>
<td>FF65 97</td>
<td>FCB</td>
<td>@227</td>
</tr>
<tr>
<td>FF66 40</td>
<td>FCB</td>
<td>@100</td>
</tr>
<tr>
<td>FF67 8F</td>
<td>FCB</td>
<td>@217</td>
</tr>
<tr>
<td>FF68 40</td>
<td>FCB</td>
<td>@100</td>
</tr>
<tr>
<td>FF69 8F</td>
<td>FCB</td>
<td>@217</td>
</tr>
<tr>
<td>FF6A 40</td>
<td>FCB</td>
<td>@100</td>
</tr>
<tr>
<td>FF6B 8F</td>
<td>FCB</td>
<td>@217</td>
</tr>
<tr>
<td>FF6C 40</td>
<td>FCB</td>
<td>@100</td>
</tr>
<tr>
<td>FF6D 86</td>
<td>FCB</td>
<td>@206</td>
</tr>
<tr>
<td>FF6E 40</td>
<td>FCB</td>
<td>@100</td>
</tr>
<tr>
<td>FF6F 86</td>
<td>FCB</td>
<td>@206</td>
</tr>
<tr>
<td>FF70 40</td>
<td>FCB</td>
<td>@100</td>
</tr>
<tr>
<td>FF71 86</td>
<td>FCB</td>
<td>@206</td>
</tr>
<tr>
<td>FF72 40</td>
<td>FCB</td>
<td>@100</td>
</tr>
<tr>
<td>FF73 85</td>
<td>FCB</td>
<td>@205</td>
</tr>
<tr>
<td>FF74 40</td>
<td>FCB</td>
<td>@100</td>
</tr>
<tr>
<td>FF75 85</td>
<td>FCB</td>
<td>@205</td>
</tr>
<tr>
<td>FF76 40</td>
<td>FCB</td>
<td>@100</td>
</tr>
<tr>
<td>FF77 85</td>
<td>FCB</td>
<td>@205</td>
</tr>
<tr>
<td>FF78 40</td>
<td>FCB</td>
<td>@100</td>
</tr>
<tr>
<td>FF79 8C</td>
<td>FCB</td>
<td>@214</td>
</tr>
<tr>
<td>FF7A 40</td>
<td>FCB</td>
<td>@100</td>
</tr>
<tr>
<td>FF7B 8C</td>
<td>FCB</td>
<td>@214</td>
</tr>
<tr>
<td>FF7C 40</td>
<td>FCB</td>
<td>@100</td>
</tr>
<tr>
<td>FF7D 8C</td>
<td>FCB</td>
<td>@214</td>
</tr>
<tr>
<td>FF7E 40</td>
<td>FCB</td>
<td>@100</td>
</tr>
<tr>
<td>FF7F 94</td>
<td>FCB</td>
<td>@224</td>
</tr>
<tr>
<td>FF80 40</td>
<td>FCB</td>
<td>@100</td>
</tr>
<tr>
<td>FF81 94</td>
<td>FCB</td>
<td>@224</td>
</tr>
<tr>
<td>FF82 40</td>
<td>FCB</td>
<td>@100</td>
</tr>
<tr>
<td>FF83 94</td>
<td>FCB</td>
<td>@224</td>
</tr>
<tr>
<td>FF84 40</td>
<td>FCB</td>
<td>@100</td>
</tr>
<tr>
<td>FF85 9D</td>
<td>FCB</td>
<td>@235</td>
</tr>
<tr>
<td>FF86 40</td>
<td>FCB</td>
<td>@100</td>
</tr>
<tr>
<td>FF87 9D</td>
<td>FCB</td>
<td>@235 &lt; - - - 235</td>
</tr>
<tr>
<td>FF88 40</td>
<td>FCB</td>
<td>@100</td>
</tr>
<tr>
<td>FF89 9D</td>
<td>FCB</td>
<td>@235</td>
</tr>
</tbody>
</table>

* First command
  * Demonstration routine that arranges blocks to spell " O U " in letters stacked three blocks high.

* User must supply blocks entered at column 0, row 0.
  * Note auto-calibration after 50th move.

* Example:
  
  ..........put down at ..........column 3
  ::::::::::row 5
  ::::::::

*
FF8A 40  FCB  $100
FF8B 9E  FCB  $236
FF8C 40  FCB  $100
FF8D 9E  FCB  $236
FF8E 40  FCB  $100
FF8F 9E  FCB  $236
FF90 40  FCB  $100
FF91 BB  FCB  $273
FF92 40  FCB  $100
FF93 BB  FCB  $273
FF94 40  FCB  $100
FF95 BB  FCB  $273
FF96 40  FCB  $100
FF97 BA  FCB  $272
FF98 40  FCB  $100
FF99 BA  FCB  $272
FF9A 40  FCB  $100
FF9B BA  FCB  $272
FF9C 40  FCB  $100
FF9D B9  FCB  $271
FF9E 40  FCB  $100
FF9F B9  FCB  $271
FFA0 40  FCB  $100
FFA1 B9  FCB  $271
FFA2 40  FCB  $100
FFA3 B8  FCB  $270
FFA4 40  FCB  $100
FFA5 B8  FCB  $270
FFA6 40  FCB  $100
FFA7 B8  FCB  $270
FFA8 40  FCB  $100
FFA9 B0  FCB  $260
FFAA 40  FCB  $100
FFAB B0  FCB  $260
FFAC 40  FCB  $100
FFAD B0  FCB  $260
FFAE 40  FCB  $100
FFAF A8  FCB  $250
FFB0 40  FCB  $100
FFB1 A8  FCB  $250
FFB2 40  FCB  $100
FFB3 A8  FCB  $250
FFB4 40  FCB  $100
FFB5 A1  FCB  $241
FFB6 40  FCB  $100
FFB7 A1  FCB  $241
FFB8 40  FCB  $100
FFB9 A1  FCB  $241
FFBA 40  FCB  $100
FFBB A2  FCB  $242
FFBC 40  FCB  $100
FFBD A2  FCB  $242
FFBE 40  FCB  $100
FFBF A2  FCB  $242
FFC0 40  FCB  $100
FFC1 A3  FCB  $243
FFC2 40  FCB  $100
FFC3 A3  FCB  $243
FFC4 40  FCB  $100
FFC5 A3  FCB  $243
FFC6 BF  FCB  $277
ENDDEMO FCB  $277  * Last command
*******************************************************************************************
*  March 28, 1988
*  TABS(8), MARGINS=0,80 ,LAST PRINTED COL=54,55 OPEN

Errors: 0
APPENDIX B

SCHEMATIC DIAGRAMS
Microcomputer Pin

Assignments

<table>
<thead>
<tr>
<th>Pin</th>
<th>Assignment</th>
</tr>
</thead>
<tbody>
<tr>
<td>P0</td>
<td>UAC SOLENOID</td>
</tr>
<tr>
<td>P1</td>
<td>ELECTROMAGNET</td>
</tr>
<tr>
<td>P2</td>
<td>X FULL POWER</td>
</tr>
<tr>
<td>P3</td>
<td>Y FULL POWER</td>
</tr>
<tr>
<td>P4</td>
<td>ALARM</td>
</tr>
<tr>
<td>P5</td>
<td>SPARE</td>
</tr>
<tr>
<td>P6</td>
<td></td>
</tr>
<tr>
<td>P7</td>
<td></td>
</tr>
<tr>
<td>P8</td>
<td>COIL 1 *</td>
</tr>
<tr>
<td>P9</td>
<td>COIL 2 *</td>
</tr>
<tr>
<td>P10</td>
<td>COIL 3 *</td>
</tr>
<tr>
<td>P11</td>
<td>COIL 4 *</td>
</tr>
<tr>
<td>P12</td>
<td></td>
</tr>
<tr>
<td>P13</td>
<td></td>
</tr>
<tr>
<td>P14</td>
<td>COIL 1 *</td>
</tr>
<tr>
<td>P15</td>
<td>COIL 2 *</td>
</tr>
<tr>
<td>P16</td>
<td>COIL 3 *</td>
</tr>
<tr>
<td>P17</td>
<td>COIL 4 *</td>
</tr>
<tr>
<td>P18</td>
<td></td>
</tr>
<tr>
<td>P19</td>
<td></td>
</tr>
<tr>
<td>P20</td>
<td></td>
</tr>
<tr>
<td>P21</td>
<td></td>
</tr>
<tr>
<td>P22</td>
<td></td>
</tr>
</tbody>
</table>

* HALT
* MODE
* USER
* SERIAL IN
* HANDSHAKE OUT
MICROCOMPUTER SCHEMATIC

DIAGRAM:

- HD63P01
- U10: 2.54 MHz oscillator
- 510 Ω resistors
- 3300 Ω resistors
- P10, P11, P12, P13, P14, P15, P16, P17
- P20, P21, P22, P23, P24, P30, P31, P32, P33, P34, P35, P36, P37
- P38: RES
- P39: STBY
- P40: NMI
- P41: IRQ1
- P42: RES
- P43: STBY
- P44: P20
- P45: P21
- P46: P22
- P47: P23
- P48: P24
- 74LS245

NOTES:

- +5 V power supply
- HALT input
- LIMIT SWITCH output
- IRQ1 input

COMPONENTS:

- HD63P01: Microcontroller
- 2.54 MHz oscillator
- 510 Ω, 3300 Ω resistors
- P10, P11, P12, P13, P14, P15, P16, P17
- P20, P21, P22, P23, P24
- P30, P31, P32, P33, P34, P35, P36, P37
- P38: RES, STBY
- P40: NMI
- P41: IRQ1
- P42: RES
- P43: STBY
- P44: P20
- P45: P21
- P46: P22
- P47: P23
- P48: P24
- 74LS245: Logic Gate

POWER SUPPLY:

- +5 V power supply

INPUTS:

- HALT
- IRQ1

OUTPUTS:

- LIMIT SWITCH
- P10, P11, P12, P13
Output Interfacing Circuit

Stepper Motor Drive Circuit
## MOTHERBOARD (MB)

<table>
<thead>
<tr>
<th>I/O</th>
<th>NAME *</th>
<th>FUNCTION</th>
<th>EDGE PIN CONNECTOR</th>
<th>WIRE COLOR</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>PS1 -</td>
<td>GND LOGIC</td>
<td>1, 2, A</td>
<td>GRY/BLK</td>
</tr>
<tr>
<td>I</td>
<td>PS1 +</td>
<td>POS UNREG</td>
<td>3, B</td>
<td>GREY</td>
</tr>
<tr>
<td>O</td>
<td>MB +5</td>
<td>+5 V LOGIC</td>
<td>C, D</td>
<td>GREEN</td>
</tr>
<tr>
<td>I</td>
<td>IRQ1</td>
<td>HALT/EMERG</td>
<td>8</td>
<td>RED</td>
</tr>
<tr>
<td>I</td>
<td>MB P10</td>
<td>XMAX LIMIT</td>
<td>S</td>
<td>WHITE</td>
</tr>
<tr>
<td>I</td>
<td>MB P11</td>
<td>YMAX LIMIT</td>
<td>P</td>
<td>WHITE</td>
</tr>
<tr>
<td>I</td>
<td>MB P12</td>
<td>XMIN LIMIT</td>
<td>N</td>
<td>GREEN</td>
</tr>
<tr>
<td>I</td>
<td>MB P13</td>
<td>YMIN LIMIT</td>
<td>M</td>
<td>GREEN</td>
</tr>
<tr>
<td>I</td>
<td>MB P14</td>
<td>UNUSED</td>
<td>13</td>
<td>-</td>
</tr>
<tr>
<td>I</td>
<td>MB P15</td>
<td>HALT/EMERG</td>
<td>12</td>
<td>WHITE</td>
</tr>
<tr>
<td>I</td>
<td>MB P16</td>
<td>UNUSED</td>
<td>11</td>
<td>-</td>
</tr>
<tr>
<td>I</td>
<td>MB P17</td>
<td>DEMO SWITCH</td>
<td>10</td>
<td>BLUE</td>
</tr>
<tr>
<td>I</td>
<td>MB P23</td>
<td>RECV SERIAL</td>
<td>5</td>
<td>RED</td>
</tr>
<tr>
<td>O</td>
<td>MB P24</td>
<td>HANDSHAKE</td>
<td>6</td>
<td>GREEN</td>
</tr>
<tr>
<td>O</td>
<td>MB P30</td>
<td>VAC SOL</td>
<td>Z</td>
<td>GREEN</td>
</tr>
<tr>
<td>O</td>
<td>MB P31</td>
<td>MAGNET</td>
<td>Y</td>
<td>RED</td>
</tr>
<tr>
<td>O</td>
<td>MB P32</td>
<td>X FULL PWR</td>
<td>X</td>
<td>YELLOW</td>
</tr>
<tr>
<td>O</td>
<td>MB P33</td>
<td>Y FULL PWR</td>
<td>W</td>
<td>YEL/BLK</td>
</tr>
<tr>
<td>O</td>
<td>MB P34</td>
<td>ALARM</td>
<td>19</td>
<td>BLUE</td>
</tr>
<tr>
<td>O</td>
<td>MB P35</td>
<td>SPARE</td>
<td>20</td>
<td>ORANGE</td>
</tr>
<tr>
<td>O</td>
<td>MB P36</td>
<td></td>
<td>21</td>
<td>-</td>
</tr>
<tr>
<td>O</td>
<td>MB P37</td>
<td></td>
<td>22</td>
<td>-</td>
</tr>
<tr>
<td>O</td>
<td>MB P40</td>
<td>X COIL #1</td>
<td>V</td>
<td>BLACK</td>
</tr>
<tr>
<td>O</td>
<td>MB P41</td>
<td>X COIL #2</td>
<td>U</td>
<td>GREEN</td>
</tr>
<tr>
<td>O</td>
<td>MB P42</td>
<td>X COIL #3</td>
<td>T</td>
<td>RED</td>
</tr>
<tr>
<td>O</td>
<td>MB P43</td>
<td>X COIL #4</td>
<td>18</td>
<td>WHITE</td>
</tr>
<tr>
<td>O</td>
<td>MB P44</td>
<td>X COIL #1</td>
<td>17</td>
<td>BLACK</td>
</tr>
<tr>
<td>O</td>
<td>MB P45</td>
<td>X COIL #2</td>
<td>15</td>
<td>GREEN</td>
</tr>
<tr>
<td>O</td>
<td>MB P46</td>
<td>X COIL #3</td>
<td>16</td>
<td>RED</td>
</tr>
<tr>
<td>O</td>
<td>MB P47</td>
<td>X COIL #4</td>
<td>14</td>
<td>WHITE</td>
</tr>
</tbody>
</table>
**DRIVER BOARD #1 (DB1)**

<table>
<thead>
<tr>
<th>I/O</th>
<th>SOURCE/DEST</th>
<th>FUNCTION</th>
<th>EDGE PIN</th>
<th>WIRE COLOR</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>MB +5</td>
<td>+5 V Reg</td>
<td>21</td>
<td>GREEN</td>
</tr>
<tr>
<td>I</td>
<td>PS2 -</td>
<td>GND Raw</td>
<td>1,2,A,B</td>
<td>WH/BLK</td>
</tr>
<tr>
<td>I</td>
<td>PS2 +</td>
<td>POS Raw</td>
<td>22,Y,Z</td>
<td>WH/RED</td>
</tr>
<tr>
<td>I</td>
<td>MB P32</td>
<td>X FULL PWR</td>
<td>18</td>
<td>YELLOW</td>
</tr>
<tr>
<td>I</td>
<td>MB P33</td>
<td>Y FULL PWR</td>
<td>V</td>
<td>YEL/BLK</td>
</tr>
<tr>
<td>I</td>
<td>MB P40</td>
<td>X COIL #1</td>
<td>19</td>
<td>BLACK</td>
</tr>
<tr>
<td>I</td>
<td>MB P41</td>
<td>X COIL #2</td>
<td>20</td>
<td>GREEN</td>
</tr>
<tr>
<td>I</td>
<td>MB P42</td>
<td>X COIL #3</td>
<td>X</td>
<td>RED</td>
</tr>
<tr>
<td>I</td>
<td>MB P43</td>
<td>X COIL #4</td>
<td>W</td>
<td>WHITE</td>
</tr>
<tr>
<td>I</td>
<td>MB P44</td>
<td>Y COIL #1</td>
<td>16</td>
<td>BLACK</td>
</tr>
<tr>
<td>I</td>
<td>MB P45</td>
<td>Y COIL #2</td>
<td>17</td>
<td>GREEN</td>
</tr>
<tr>
<td>I</td>
<td>MB P46</td>
<td>Y COIL #3</td>
<td>U</td>
<td>RED</td>
</tr>
<tr>
<td>I</td>
<td>MB P47</td>
<td>Y COIL #4</td>
<td>T</td>
<td>WHITE</td>
</tr>
<tr>
<td>O</td>
<td>DB1 XFP</td>
<td>X FULL PWR</td>
<td>13,14,P,R</td>
<td>BRN/GRN</td>
</tr>
<tr>
<td>O</td>
<td>DB1 YFP</td>
<td>Y FULL PWR</td>
<td>11,12,M,N</td>
<td>BROWN</td>
</tr>
<tr>
<td>O</td>
<td>DB1 X1</td>
<td>X COIL #1</td>
<td>10,L</td>
<td>ORANGE</td>
</tr>
<tr>
<td>O</td>
<td>DB1 X2</td>
<td>X COIL #2</td>
<td>9,K</td>
<td>RED</td>
</tr>
<tr>
<td>O</td>
<td>DB1 X3</td>
<td>X COIL #3</td>
<td>8,J</td>
<td>BLUE</td>
</tr>
<tr>
<td>O</td>
<td>DB1 X4</td>
<td>X COIL #4</td>
<td>7,H</td>
<td>GREEN</td>
</tr>
<tr>
<td>O</td>
<td>DB1 Y1</td>
<td>Y COIL #1</td>
<td>6,F</td>
<td>ORANGE</td>
</tr>
<tr>
<td>O</td>
<td>DB1 Y2</td>
<td>Y COIL #2</td>
<td>5,E</td>
<td>RED</td>
</tr>
<tr>
<td>O</td>
<td>DB1 Y3</td>
<td>Y COIL #3</td>
<td>4,D</td>
<td>BLUE</td>
</tr>
<tr>
<td>O</td>
<td>DB1 Y4</td>
<td>Y COIL #4</td>
<td>3,C</td>
<td>GREEN</td>
</tr>
</tbody>
</table>
## DRIVER BOARD #2 (DB2)

<table>
<thead>
<tr>
<th>I/O</th>
<th>SOURCE/DEST NAME *</th>
<th>FUNCTION</th>
<th>EDGE PINS CONNECTOR</th>
<th>WIRE COLOR</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>MB +5</td>
<td>+5 V REG</td>
<td>21</td>
<td>GREEN</td>
</tr>
<tr>
<td>I</td>
<td>PS2 -</td>
<td>GND RAW</td>
<td>1,2,A,B</td>
<td>WH/BLK</td>
</tr>
<tr>
<td>I</td>
<td>PS2 +</td>
<td>POS RAW</td>
<td>22,Y,Z</td>
<td>WH/RED</td>
</tr>
<tr>
<td>I</td>
<td>MB P30</td>
<td>VAC SOL</td>
<td>R</td>
<td>GREEN</td>
</tr>
<tr>
<td>I</td>
<td>MB P31</td>
<td>MAGNET</td>
<td>N</td>
<td>RED</td>
</tr>
<tr>
<td>I</td>
<td>MB P34</td>
<td>ALARM</td>
<td>V</td>
<td>BLUE</td>
</tr>
<tr>
<td>I</td>
<td>MB P35</td>
<td>SPARE</td>
<td>T</td>
<td>ORANGE</td>
</tr>
<tr>
<td>O</td>
<td>DB2 VAC</td>
<td>VAC SOL</td>
<td>3,C</td>
<td>GREEN</td>
</tr>
<tr>
<td>O</td>
<td>DB2 MAG</td>
<td>MAGNET</td>
<td>4,D</td>
<td>RED</td>
</tr>
<tr>
<td>-</td>
<td>-</td>
<td>ALARM</td>
<td>INTERNAL</td>
<td></td>
</tr>
<tr>
<td>-</td>
<td>-</td>
<td>LED</td>
<td>5,E</td>
<td></td>
</tr>
</tbody>
</table>

* Name is read as follows:
  
  On Motherboard: Name based on microprocessor ports.
  On DB1 & DB2: If input, then name is source.
  If output, the new name is given.

### NAME CODES

- **PS1** = Logic power supply
- **PS2** = Driver power supply
- **MB** = Motherboard
- **DB1** = Driver board #1
- **DB2** = Driver board #2
**Y-AXIS FLEXIBLE HARNESS**
9-Circuit (0.062" Pins)

<table>
<thead>
<tr>
<th>PIN #</th>
<th>SOURCE</th>
<th>FLEX CABLE COLOR</th>
<th>FUNCTION</th>
<th>MOTOR COLOR</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>DB1 Y1</td>
<td>WHITE</td>
<td>Y COIL #1</td>
<td>WH/RED</td>
</tr>
<tr>
<td>2</td>
<td>DB1 Y2</td>
<td>RED</td>
<td>Y COIL #2</td>
<td>RED</td>
</tr>
<tr>
<td>3</td>
<td>DB1 Y3</td>
<td>BLACK</td>
<td>Y COIL #3</td>
<td>WH/GRN</td>
</tr>
<tr>
<td>4</td>
<td>DB1 Y4</td>
<td>GREEN</td>
<td>Y COIL #4</td>
<td>GREEN</td>
</tr>
<tr>
<td>5</td>
<td>DB1 YFP</td>
<td>DRAIN</td>
<td>COMMON</td>
<td>BLK &amp; WH</td>
</tr>
<tr>
<td>6</td>
<td>MB P13</td>
<td>GREEN</td>
<td>Y MIN LIMIT</td>
<td>-</td>
</tr>
<tr>
<td>7</td>
<td>MB P11</td>
<td>WHITE</td>
<td>Y MAX LIMIT</td>
<td>-</td>
</tr>
<tr>
<td>8</td>
<td>PS1 +</td>
<td>RED</td>
<td>+5V LOGIC</td>
<td>-</td>
</tr>
<tr>
<td>9</td>
<td>PS1 -</td>
<td>BLK&amp;DRN</td>
<td>GND LOGIC</td>
<td>-</td>
</tr>
</tbody>
</table>
**STEPPER MOTOR HARNESS**
6-Circuit (.092" Pins)

![Diagram of 6-pinch stepper motor harness]

### X-AXIS CABLE

<table>
<thead>
<tr>
<th>PIN #</th>
<th>MOTOR FUNCTION</th>
<th>SOURCE</th>
<th>WIRE COLOR</th>
<th>MOTOR COLOR</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>X COIL #1</td>
<td>DB1 X1</td>
<td>ORANGE</td>
<td>WH/RED</td>
</tr>
<tr>
<td>2</td>
<td>X COIL #2</td>
<td>DB1 X2</td>
<td>RED</td>
<td>RED</td>
</tr>
<tr>
<td>3</td>
<td>X COIL #3</td>
<td>DB1 X3</td>
<td>BLUE</td>
<td>WH/GRN</td>
</tr>
<tr>
<td>4</td>
<td>X COIL #4</td>
<td>DB1 X4</td>
<td>GREEN</td>
<td>GREEN</td>
</tr>
<tr>
<td>5</td>
<td>COMMON</td>
<td>DB1 XFP</td>
<td>WH&amp;B/BLK</td>
<td>WH &amp; BLK</td>
</tr>
<tr>
<td>6</td>
<td>SHIELD</td>
<td>DB1 XFP*</td>
<td>DRAIN</td>
<td>WH &amp; BLK</td>
</tr>
</tbody>
</table>

### Y-AXIS CABLE

<table>
<thead>
<tr>
<th>PIN #</th>
<th>MOTOR FUNCTION</th>
<th>SOURCE</th>
<th>WIRE COLOR</th>
<th>FLEXIBLE HARNESS #</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Y COIL #1</td>
<td>DB1 Y1</td>
<td>ORG</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>Y COIL #2</td>
<td>DB1 Y2</td>
<td>RED</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>Y COIL #3</td>
<td>DB1 Y3</td>
<td>BLUE</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>Y COIL #4</td>
<td>DB1 Y4</td>
<td>GREEN</td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td>COMMON</td>
<td>DB1 YFP</td>
<td>WH&amp;B/BLK</td>
<td>5</td>
</tr>
<tr>
<td>6</td>
<td>SHIELD</td>
<td>DB1 YFP*</td>
<td>DRAIN</td>
<td>5</td>
</tr>
</tbody>
</table>

* Shield/Drain is connected at one end to motor common or XFP or YFP. Connecting to GND defeats Low Power Mode.
### X-Limit Switch Harness

<table>
<thead>
<tr>
<th>#</th>
<th>Function</th>
<th>Color</th>
<th>Connects</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>+5 V</td>
<td>Red</td>
<td>MB +5</td>
</tr>
<tr>
<td>2</td>
<td>GND</td>
<td>Blk &amp; ORN</td>
<td>MB GND</td>
</tr>
<tr>
<td>3</td>
<td>XMIN</td>
<td>Green</td>
<td>MB P12</td>
</tr>
<tr>
<td>4</td>
<td>XMAX</td>
<td>White</td>
<td>MB P10</td>
</tr>
</tbody>
</table>

### Y-Limit Switch Harness

<table>
<thead>
<tr>
<th>#</th>
<th>Function</th>
<th>Color</th>
<th>Connects</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>+5 V</td>
<td>RED</td>
<td>MB +5</td>
</tr>
<tr>
<td>2</td>
<td>GND</td>
<td>BLK &amp; ORN</td>
<td>MB GND</td>
</tr>
<tr>
<td>3</td>
<td>YMIN</td>
<td>GREEN</td>
<td>MB P13</td>
</tr>
<tr>
<td>4</td>
<td>YMAX</td>
<td>WHITE</td>
<td>MB P11</td>
</tr>
</tbody>
</table>
### VACUUM SOLENOID and ELECTROMAGNET HARNESS

![Diagram of 1-4 connections]

<table>
<thead>
<tr>
<th>#</th>
<th>FUNCTION</th>
<th>SOURCE</th>
<th>COLOR</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>+VAC SOL</td>
<td>DB2 VAC</td>
<td>GREEN</td>
</tr>
<tr>
<td>2</td>
<td>SHIELD/GND</td>
<td>PS2 -</td>
<td>WH/BLK</td>
</tr>
<tr>
<td>3</td>
<td>GND</td>
<td>PS2 -</td>
<td>DRAIN</td>
</tr>
<tr>
<td>4</td>
<td>+MAGNET</td>
<td>DB2 MAG</td>
<td>RED</td>
</tr>
</tbody>
</table>

### OTHER CONNECTORS

![Diagram of 1-2 connections]

<table>
<thead>
<tr>
<th>#</th>
<th>FUNCTION</th>
<th>SOURCE</th>
<th>COLOR</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>+ VAC SOL</td>
<td>DB2 VAC</td>
<td>B1K/RIB</td>
</tr>
<tr>
<td>2</td>
<td>SHIELD/GND</td>
<td>PS2 -</td>
<td>BLACK</td>
</tr>
<tr>
<td>1</td>
<td>+ MAGNET</td>
<td>DB2 MAG</td>
<td>BLACK/RIB</td>
</tr>
<tr>
<td>2</td>
<td>SHIELD/GND</td>
<td>PS2 -</td>
<td>BLACK</td>
</tr>
<tr>
<td>1</td>
<td>HALT/EMG</td>
<td>MB P15, IRQ1</td>
<td>RED &amp; WHT</td>
</tr>
<tr>
<td>2</td>
<td>GND LOGIC</td>
<td>PS1 -</td>
<td>BLK &amp; DRN</td>
</tr>
</tbody>
</table>
APPENDIX D

SUPPLIERS REFERENCE
<table>
<thead>
<tr>
<th>COMPONENT</th>
<th>PART NUMBER</th>
<th>SUPPLIERS*</th>
<th>SEE #</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alarm</td>
<td>12 Volt Buzzer</td>
<td>N, G, B, J</td>
<td>1</td>
</tr>
<tr>
<td>Connectors</td>
<td>2,4,6-Circuit</td>
<td>N, F, L</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>9 - Circuit</td>
<td>F, L</td>
<td>3</td>
</tr>
<tr>
<td>Crystal</td>
<td>2.4576 MHz</td>
<td>B</td>
<td>6</td>
</tr>
<tr>
<td>Darlington Trans.</td>
<td>TIP 101</td>
<td>G,b,f,m,n</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>TIP 106</td>
<td>G,b,f,m</td>
<td>5</td>
</tr>
<tr>
<td>Drive Chain</td>
<td>31GCF-75-E</td>
<td>R</td>
<td>7</td>
</tr>
<tr>
<td></td>
<td>31GCF-5FT</td>
<td>R</td>
<td>8</td>
</tr>
<tr>
<td>EPROM</td>
<td>2764-25</td>
<td>G, B</td>
<td>9</td>
</tr>
<tr>
<td>Flexible Tubing</td>
<td>Latex Rubber</td>
<td></td>
<td>10</td>
</tr>
<tr>
<td>Limit Switch</td>
<td>Micro 37XL31XB</td>
<td>P, F, K</td>
<td>11</td>
</tr>
<tr>
<td>Microcomputer</td>
<td>HD63P01M1</td>
<td>G,D,O,d,m,o</td>
<td>12</td>
</tr>
<tr>
<td>Octal Bus Transc</td>
<td>74LS245</td>
<td>G, B, F</td>
<td>13</td>
</tr>
<tr>
<td>Optoisolator</td>
<td>4N28GE</td>
<td>G, B</td>
<td>14</td>
</tr>
<tr>
<td></td>
<td>PS2401A-4NEC</td>
<td>B</td>
<td>15</td>
</tr>
<tr>
<td>Plug-in Board</td>
<td>276-154A</td>
<td>N, f, g</td>
<td>16</td>
</tr>
<tr>
<td></td>
<td>276-190</td>
<td>N, f, g</td>
<td>17</td>
</tr>
<tr>
<td>RS-232 Interface</td>
<td>MAX232CPE</td>
<td>G, H, I</td>
<td>18</td>
</tr>
<tr>
<td>Sprocket Gear</td>
<td>31B4-24</td>
<td>R</td>
<td>19</td>
</tr>
<tr>
<td></td>
<td>31B4-28</td>
<td>R</td>
<td>19</td>
</tr>
<tr>
<td></td>
<td>31B4-56</td>
<td>R</td>
<td>19</td>
</tr>
<tr>
<td></td>
<td>31B4-72</td>
<td>R</td>
<td>19</td>
</tr>
<tr>
<td>Stepper Motors</td>
<td>MO61-FD02</td>
<td>E, Q, a</td>
<td>20</td>
</tr>
<tr>
<td>Vacuum Source</td>
<td>TM21K931</td>
<td>E, c</td>
<td>21</td>
</tr>
<tr>
<td>Voltage Regulator</td>
<td>LM340T-5</td>
<td>G, B, N</td>
<td>22</td>
</tr>
</tbody>
</table>
SUPPLIERS

*Key to suppliers data - Capital letter indicates exact replacement. Lower case indicates substitutions. Order of appearance (left to right) is best search path.*

(A) BODINE ELECTRIC CO.
2500 W. Bradley Place
Chicago, IL 60618
(312) 478-3515

(B) DIGI-KEY CORPORATION
P.O. Box 677
Thief River Falls, MN 56701-9988
(800) 344-4539

(C) GRAINGER'S
3640 Interchange Road
Columbus, Ohio 43204
(614) 276-5231

(D) HITACHI AMERICA, LTD.
6 Parklane Blvd., #558
Dearborn, MI 48126
(313) 271-4410

(E) H & R COMPANY
401 E. Erie Avenue
Philadelphia, PA 19134
(215) 426-1708

(F) HUGHES-PETERS
481 East 11th Avenue
Columbus, Ohio 43211-2601
(614) 294-5351

(G) JAMECO ELECTRONICS
1355 Shoreway Road
Belmont, CA 94002
(415) 592-8097

(H) LYONS CORPORATION
4812 Frederick Road, Suite 101
Dayton, Ohio 45414
(513) 278-0714
(I) MAXIM INTEGRATED PRODUCTS
510 N. Pastoria Avenue
Sunnyvale, CA 94086
(408) 737-7600

(J) MCM ELECTRONICS
858 E. Congress Park Dr.
Dayton, Ohio 45459
(800) 762-4315

(K) MICRO SWITCH
4540 Honeywell Court
Dayton, Ohio 45424
(513) 237-4075

(L) MOLEX/WALDOM ELECTRONICS
4301 West 69th street
Chicago, IL 60629
(312) 585-1212

(M) MOTOROLA
Semiconductor Products Sector
3102 North 56th Street
Phoenix, AZ 85018-6606
(800) 521-6274

(N) RADIO SHACK
500 One Tandy Center
Fort Worth, TX 76102

(O) REPTRON ELECTRIC
404 E. Wilson Bridge Road
Columbus, Ohio 43085
(614) 436-6675

(P) R & D ELECTRONICS
1202H Pine Island Road
Cape Coral, FL 33909
(813) 772-1441

(Q) SUPERIOR ELECTRIC
Bristol, CT 06010
(203) 582-9561

(R) WINFRED M. BERG, INC
499 Ocean Avenue
East Rockaway, NY 11518
(516) 599-5010
COMPONENT NOTES

1. (Alarm) Any - approx. 9 volts, 2A maximum.

2. (Conn) Molex standard (.092 pins) . Reuseable shells.

3. Molex miniature. Reuseable shells, .062" pins = #1561-60

4. (Dar1) TIP 101 = NPN, 5A, Gain > 1000
   Subst: TIP 120, or typical.

5. TIP 106 = PNP, 5A, Gain > 1000
   Subst: TIP 121, or typical.

6. (Xtal) Par. resonant, HC-18 package = small,
   system clock = Xtal / 4 .

7. (Chain) Polyurethane coated steel cable,
   1/10 " pitch, 75 pitches.

8. Same as above (7), minimum 600 pitches.

9. (EPROM) Max access = 300 us, Subst: 2764A-25,
   2764-20, 27C64-15 .

10. (Flex) Medical supply (.25" ID, thin wall),
    Subst: Low temp silicone.

11. (Limit) Hall Effect, 5V, open collector,
    normally high (off).

12. (MCU) Subst: HD63PA01, HD63PB01 Piggyback Eprom
    Possible Subst: HD63701, MC68701 int Eprom.

13. (Tranc) Narrow DIP package (standard).
    Subst: 74ALS245


15. Quad optoisolator, Subst: Any same pinout.

16. (Plug) Any 22/44 edge card, bus structure recmmnd.

17. Any 22/44 edge card. Handles larger currnts.

18. (RS232) +5 V powered, dual Xmit/Rec buffer,
    EIA RS-232C specs.
19. (Gear) Matches chains. 1/4" shaft. Trailing number is teeth.

20. (Step) Slo-Syn, 200 steps/rev, 5 volts, 1A/winding, Min hold torque = 53 oz-in, H & R # TM23K607. Subst: Bodine size 23 frame.

21. (Vac) 120V, 2.1A, AC/DC series wound. Subst: Any (-1 psi minimum).

22. (Reg) Positive 5 V. TO-220 case. Same as 7805T.