The overall goal of "Telerobotic Control of a Mobile Coordinated Robotic Servicer" project is to develop advance control methods that would enhance the usage of robotic systems for space applications. Towards this end, several algorithms have been developed from this project. One area of development was to extend the methodology of the Observer/Kalman Filter Identification (OKID) approach, developed at NASA Langley, to such design problems as frequency spectrum reconstruction, improved parameter estimation from frequency data and recursion structures to improve computational performance. This area addressed the identification issue of systems which can then be followed by regulation design as is typical in self-tuning adaptive control. The approach has applicability to many types of systems, including robotics, when the system structure or parameter set is unknown or has variations.

The second area of control research focused on fuzzy control which is a non-parametric (non-model-based) knowledge-based approach. In this area, adaptive algorithms were developed using self-tuning scaling factor schemes in the fuzzifier, self-learning schemes in the control rulebase and optimization to extend the method to multi-input, multi-output systems. As a knowledge-based approach, the MIMO adaptive fuzzy controller uses a computationally efficient rulebase to determine control commands when the system model (the robot dynamics) is partially unknown or varies with time.

The final phase of this effort was devoted to the design, fabrication and testing of a robot manipulator arm which is attached to a mobile robotic system, a rover, built at the Mars Mission Research Center. The rover is currently under teleoperation mode and will have capabilities for full autonomy. The manipulator arm along with the mobile robotic system will be used to test all of the control algorithms that have been developed though this effort as well as other programs at the Mars Mission Research Center.

What follows is the MS thesis of Mr. Mike Brown. Mike spent a summer at NASA Langley working in the Spacecraft Dynamics Branch. His thesis develops the design and testing of the manipulator arm on the teleoperated mobile robotic system.
ABSTRACT

BROWN, JR., ROBERT MICHAEL. A Microcontroller-Based Three Degree-of-Freedom Manipulator Testbed. (Under the direction of Gordon K. F. Lee.)

A wheeled exploratory vehicle is under construction at the Mars Mission Research Center at North Carolina State University. In order to serve as more than an inspection tool, this vehicle requires the ability to interact with its surroundings. A crane-type manipulator, as well as the necessary control hardware and software, has been developed for use as a sample gathering tool on this vehicle. The system is controlled by a network of four Motorola M68HC11 microcontrollers. Control hardware and software were developed in a modular fashion so that the system can be used to test future control algorithms and hardware. Actuators include three stepper motors and one solenoid. Sensors include three optical encoders and one cable tensiometer.

The vehicle supervisor computer provides the manipulator system with the approximate coordinates of the target object. This system maps the workspace surrounding the given location by lowering the claw, along a set of evenly spaced vertical lines, until contact occurs. Based on this measured height information and prior knowledge of the target object size, the system determines if the object exists in the searched area. The system can find and retrieve a 1.25 in diameter by 1.25 in tall cylinder placed within the 47.5 in² search area in less than 12 minutes. This manipulator hardware may be used for future control algorithm verification and serves as a prototype for other manipulator hardware.
A MICROCONTROLLER-BASED THREE DEGREE-OF-FREEDOM MANIPULATOR TESTBED

by

ROBERT MICHAEL BROWN, JR.

A thesis submitted to the Graduate Faculty of North Carolina State University in partial fulfillment of the requirements for the Degree of Master of Science

MECHANICAL ENGINEERING

Raleigh
1995

APPROVED BY:

Chair of Advisory Committee
BIOGRAPHY

Robert M. Brown Jr. was born in Rocky Mount, NC, on December 1, 1965, to Mike and Marie Brown. He graduated from the North Carolina School of Science and Mathematics in June 1984. While attending NCSU and taking part in the cooperative engineering program, he spent five semesters working for NASA at Wallops Island, VA. He received a B. S. of Aerospace Engineering from NCSU in May 1989. He was married to Kathy Tyndall (NCSU '89) in June 1989 after which he spent two years working for NASA at Wallops Island, VA. In May of 1991, Mr. Brown left NASA to work at the National Undersea Research Center at the University of North Carolina at Wilmington. In January 1993 he enrolled in the graduate program at NCSU.
ACKNOWLEDGMENTS

I would like to recognize my family, professors, and friends. Without the assistance and support of Kathy Tyndall Brown, Mike and Marie Brown, Dr. Gordon Lee, Dr. Larry Silverberg, Chih-Kang Chao, Keita Ikeda, and the faculty and staff of the Mars Mission Research Center this goal would have been unattainable.
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>BIOGRAPHY</td>
<td>ii</td>
</tr>
<tr>
<td>ACKNOWLEDGMENTS</td>
<td>iii</td>
</tr>
<tr>
<td>LIST OF FIGURES</td>
<td>vi</td>
</tr>
<tr>
<td>LIST OF SYMBOLS</td>
<td>vii</td>
</tr>
<tr>
<td>Chapter One: INTRODUCTION</td>
<td>1</td>
</tr>
<tr>
<td>A. Background Information</td>
<td>1</td>
</tr>
<tr>
<td>B. Research Objectives and Problem Development</td>
<td>3</td>
</tr>
<tr>
<td>C. Thesis Organization</td>
<td>5</td>
</tr>
<tr>
<td>Chapter Two: SYSTEM HARDWARE DESCRIPTION</td>
<td>7</td>
</tr>
<tr>
<td>A. Crane Structure</td>
<td>7</td>
</tr>
<tr>
<td>B. Coordinate System and Workspace</td>
<td>8</td>
</tr>
<tr>
<td>C. Actuators</td>
<td>9</td>
</tr>
<tr>
<td>Stepper Motors and Drivers</td>
<td>9</td>
</tr>
<tr>
<td>Solenoid Actuated Claw</td>
<td>10</td>
</tr>
<tr>
<td>D. Sensors</td>
<td>11</td>
</tr>
<tr>
<td>Optical Encoders</td>
<td>11</td>
</tr>
<tr>
<td>Cable Tensiometer</td>
<td>12</td>
</tr>
<tr>
<td>E. Microcontrollers</td>
<td>13</td>
</tr>
<tr>
<td>Motorola 68HC11E9 General Description</td>
<td>14</td>
</tr>
<tr>
<td>Master Controller</td>
<td>17</td>
</tr>
<tr>
<td>Motor Controller</td>
<td>18</td>
</tr>
</tbody>
</table>
LIST OF FIGURES

FIGURE 1-1: NCSU MARS ROVER .......................................................... 4
FIGURE 2-1: MANIPULATOR SYSTEM ............................................. 7
FIGURE 2-2: CRANE STRUCTURE ................................................... 8
FIGURE 2-3: COORDINATE SYSTEM .............................................. 9
FIGURE 2-4: END EFFECTOR WITH SAMPLE OBJECT .................... 11
FIGURE 2-5: Tensiometer HARDWARE ......................................... 13
FIGURE 2-6: CONTROLLER HARDWARE ....................................... 14
FIGURE 4-1: SEARCH PATTERN .................................................... 27
FIGURE 4-2: TYPICAL RELIEF MAP OF MINOR GRID .................... 28
FIGURE 4-3: CASE 1 ACTUATOR OUTPUT .................................... 31
FIGURE 4-4: CASE 2 ACTUATOR OUTPUT .................................... 32
FIGURE 4-5: CASE 3 ACTUATOR OUTPUT .................................... 33
FIGURE 4-6: CASE 3 RELIEF MAP ............................................... 34
LIST OF SYMBOLS

pp ........................................ two ASCII characters representing vehicle pitch (degrees)

qq ........................................ two ASCII characters representing vehicle roll (degrees)

r .............................................. radius coordinate of claw position (inches)

rr.r ........................................... four ASCII characters representing the radius coordinate of claw position (inches)

z ............................................... height coordinate of claw position (inches)

zz.z ........................................... four ASCII characters representing the height coordinate of claw position (inches)

θ ............................................... angular coordinate of claw position (degrees)

θθθ ........................................... three ASCII characters representing the angular coordinate of claw position (degrees)

( )a .................................................................................. actual value

( )d .................................................................................. desired value

( )t .................................................................................. target value
Robotic vehicles are ideal for the exploration of hostile environments. These devices allow humans to investigate areas that would otherwise be difficult or impossible to reach. In order to serve as more than inspection tools, these robots must have the ability to interact with their surroundings. An undersea vehicle on a scientific mission must often collect sediment and water samples. A Space Station assembly vehicle must be able to position and connect building materials. An emergency response robot could open doors and move debris while searching for injured victims in a burning building. A robotic vehicle in a hazardous material spill area could be used to locate and close a critical valve.

A. Background Information

Three basic types of joints, revolute, prismatic, and suspended cable, are typically used by manipulator systems. Revolute joints, like a human elbow, rotate about an axis. Prismatic joints, like an extension ladder, extend or retract along a linear path. Suspended cable systems, used in place of rigid structural members on crane systems, also extend or retract. The key difference is that the path followed by a payload suspended by a cable is a function of gravitational effects and environmental disturbances.

A revolute joint system, such as the manipulator arm used on the NASA space shuttle, is very maneuverable. The variable direction of approach, made
possible by the slender structural members and multiple revolute joints, allows
the retrieval of unsymmetrical payloads. In the weightless environment of space,
the joint actuators must position the end-effector and damp unwanted motion.
However, in a similar system operating vertically in a gravity field, the joint
actuators must also support the manipulator structure and payload. Lower
payload capacities, relative to a crane with identical actuators, result.

Prismatic joints are often used in systems where precision is more
important than range of motion. Extremely fine control of an end effector
trajectory is possible with rigid links and prismatic joints. This advantage is
gained at the expense of mechanical complexity and additional weight. Loss of
mobility also results since the distance that a joint can extend is limited by the
length of the telescoping member.

Suspended cable joints, found in crane systems, are capable of extreme
ranges of motion. Cable, unlike rigid members, can be stored in great lengths on
winch drums. Since the structure, not the actuators, of a crane carries bending
loads, relatively high payload-to-system-weight ratios can be achieved. The key
disadvantages of a crane system are the difficulties in controlling all six degrees-
of-freedom of the end effector and in damping undesired motion.

The National Institute of Standards and Technology (NIST) has developed
a six degree-of-freedom crane called ROBOCRANE\textsuperscript{2}. This system uses cables
as structural links, winches as actuators, and cable travel encoders as sensors.
A cable, a winch actuator, and an encoder are required for each controlled
degree-of-freedom to ensure a fully constrained system. This technology is applicable to various types of crane platforms, such as tower, boom, and overhead, as well as lower degree-of-freedom systems.

Depending on system requirements and research objectives, algorithms used in crane control vary from classical to discrete to fuzzy logic schemes. The Motorola M68HC11 is a relatively inexpensive microcontroller allowing the use of both classical and fuzzy control techniques.

B. Research Objectives and Problem Development

An autonomous wheeled exploratory vehicle is currently under construction at the Mars Mission Research Center at North Carolina State University. This vehicle, pictured in Figure 1-1, will be tasked with the exploration of unfamiliar terrain. In order to effectively carry out this mission, the vehicle must avoid dangers, such as boulders and crevasses, gather information, such as visual images and sensor data, and collect physical samples, such as rock and soil. The latter mission requirement makes a manipulator subsystem necessary.

The purpose of the research presented in this thesis is to develop a manipulator that serves two purposes. The first goal is to provide the required environmental sample gathering tool for a Mars vehicle prototype. The second goal is to provide a platform for future robotic manipulator research activities. A crane-type manipulator system configuration was selected to insure adequate payload capacity. Motorola microcontrollers were selected so that the prototype
payload capacity. Motorola microcontrollers were selected so that the prototype system will be capable of implementing both traditional and modern control techniques. The structural hardware, electrical hardware, and control software have been designed and constructed in a modular fashion. Future researchers will be able to further optimize the system by modifying individual hardware and software components.

Figure 1-1: NCSU Mars Rover

Based on the existing vehicle design, the following criteria must be met by the vehicle and manipulator subsystem.

1) The vehicle must
   • Provide 12 V and 48 V electrical power.
- Provide an ASCII string, via a serial link, containing manipulator platform pitch and roll as well as object location in cylindrical coordinates. The string format will be ±pp,±qq,±θθθ,±rr,r,±zz.z.
- Disable the wheel motor subsystem during manipulator operation.
- Confirm retrieval of desired object with vehicle sensor devices, such as vision or ultrasound.

2) The manipulator system must
- Fit in a space that measures 18 in long by 8 in wide by 18 in tall.
- Weigh no more than 20 lb.
- Find and retrieve a typical environmental sample, approximated by a 1.25 in diameter by 1.25 in tall cylinder, when provided with a target location inside the workspace and within 4 in of actual object location.
- Be capable of lifting a payload weighing up to 1 lb.

C. Thesis Organization

This thesis is divided into seven chapters. Chapter 2, System Hardware Description, describes mechanical and electrical system components. In Chapter 3, Control Software Description, a discussion is presented on the software embedded in each of the four controllers. Chapter 4, Application of Control Software to Crane System, details the system tests and results. Finally, Chapter 5, Conclusions and Suggestions for Future Work, states conclusions
and offers ideas for system improvements. Appendices contains commented computer code for each type of controller and programming instructions.
Chapter Two: SYSTEM HARDWARE DESCRIPTION

The manipulator system is composed of the master controller, three motor controllers, three motor drivers, three stepper motors, three optical encoders, a solenoid actuated claw, a tensiometer, and the crane structure. These components are pictured in Figure 2-1 and are discussed in the following sections of this chapter.

A. Crane Structure

Four major components, illustrated in Figure 2-2, make up the structure of the manipulator. The tower is the vertical structure about which the boom pivots. The lower flange of the boom acts as a track for the trolley. The claw is suspended on a aramid fiber cable from the trolley. The tower, boom, and trolley
are constructed of readily available components to allow modification by future users. The main structural components are formed from prefabricated fiberglass I-beam, channel, and angle stock. The boom and cable drum are driven directly by their respective motors. The trolley position is controlled via a chain drive with the third motor.

![Crane Structure](image)

**Figure 2-2: Crane Structure**

**B. Coordinate System and Workspace**

The three coordinates used to describe the position of the closed manipulator claw tips are illustrated in Figure 2-3. The radius \( r \) and angle \( \theta \) are standard polar coordinates when the system is viewed from above. The radius is measured from the rotational axis to the trolley center. The angle is measured counter-clockwise from home position. Note that the angle illustrated in Figure 2-3 is in the negative direction. The height \( z \) is the distance from the
baseplane to the claw tips, where a positive value of $z$ is used for points above the plane.

![Figure 2-3: Coordinate System](image)

**C. Actuators**

**Stepper Motors and Drivers**

The three positioning actuators in this device are Pacific Scientific Powermax P21NRXA-LDF-M1-00 stepper motors. Each is driven by a Pacific Scientific Sigma Model 5210 motor driver. Each motor requires 2.5 A at 12 V and provides a holding torque of 114 oz-in. In addition to power, the motor driver requires two logic inputs. The level of the direction input determines the direction
of rotation. A square wave applied to the second input results in a motor step for every wave period.

The motor responsible for lifting the claw and captured object was selected to maximize the payload capacity of the system. Identical motors were selected to control boom and trolley location, to standardize hardware, and to increase modularity.

**Solenoid Actuated Claw**

The claw, illustrated in Figure 2-4, is the same type used in arcade games. A more forgiving control system and minor claw modification result in a much better success rate. The claw has three fingers, located 120° apart, that are activated by a 48V solenoid. Original tests, involving a range of object types, demonstrated that, while the claw was very effective at “scooping” up a large object, such as a four inch diameter sphere, it was not capable of holding most smaller objects in its fingertips. To include small objects in the target range, these fingers were modified by the addition of claw tips. These tips, acting as fingernails, are 0.063 inch diameter rods protruding one-quarter inch from the fingertips. They result in a great enhancement in gripping capability.
D. Sensors

Optical Encoders

A U. S. Digital Model E2-512-250-IE optical encoder is mounted on the shaft of each stepper motor. The resolution of each encoder is 512 counts per revolution. An index pulse, once per revolution, allows the motor controllers to find home position from the power-on position. Sensor output consists of two
square wave signals that, except for phase, are identical. The lead-lag relationship of these two signals reflects the direction of motion of the encoder. In general, these signals can be decoded by the motor controller. However, due to high frequency “ringing” of the stepper motor after a single step command, a separate chip was used. This chip, an LSI Computer Systems LS7166 24 bit multimode counter, can accurately decode the encoder signals even with the high frequency changes in direction of encoder rotation associated with ringing. The internal 24 bit counter containing the motor position can be read by the motor controller via an 8 bit data bus. The use of this chip relieves the microcontroller of the burden of constantly monitoring the encoder output.

**Cable Tensiometer**

A slight loss of cable tension, such as occurs when the claw makes contact with some surface or object, causes an interrupt service routine on the master controller to be activated. A discussion of this software routine can be found in Chapter 3. Figure 2-5 illustrates the mechanical components of the sensor. The key electrical component that enables this interrupt is a conditioned single-pole double-throw (SPDT) switch. This switch is mounted on a lever whose position is controlled by the cable tension. The switch and conditioning circuit\(^{11}\), detailed in Appendix A, control the state of pin PA3 on the master controller. This pin has input capture capabilities that are used to trigger the interrupt service routine.
E. Microcontrollers

Controller hardware consists of one master controller and three motor controllers. These components are pictured in Figure 2-6 and are discussed in the following sections.
Motorola 68HC11E9 General Description

The Motorola 68HC11E9 is a one of a family of devices called microcontrollers or MCUs. An MCU combines discrete communication circuitry, a processor, a data bus, and memory into a small, low power, single chip computer. The 68HC11 MCU has 40 input/output pins that allow serial and parallel communication and analog-to-digital conversion. The 16 bit memory address of the 68HC11 allows the use of 64K bytes of memory. Internal memory in the 68HC11E9 consists of 512 bytes of random access memory (RAM), 512 bytes of electronically erasable programmable read only memory (EEPROM), 12K bytes of erasable programmable read only memory (EPROM), a 64 byte register block, and a 64 byte bootstrap interrupt vector block. The remainder of the 64K space can be accessed using an external memory chip. Detailed
hardware information can be found in M68HC11 Reference Manual, M68HC11 E Series Technical Data, and M68HC11 E Series Programming Reference Guide.

While hardware defines the limits of a microcontroller's capability, the software is the tool necessary to realize these limits. Program size, execution speed, mathematical capability, and design are all important considerations when measuring the effectiveness of any computer code.

Code for the 68HC11 can be written using a number of high level languages and assemblers. While languages such as Lisp and C provide data structures and mathematical functions that allow the intuitive coding of complex behavior, Motorola Assembly Language was used to generate all code used in this project. The primary reason for this selection was to simplify code troubleshooting. Debugging code in the PCbug11 or BUFFALO environment is straightforward using the disassembler. When using this feature, code is viewed as assembly code regardless of the original language. The usable result of debugged and assembled code is an ASCII file in S-record format. This machine language version of the original program can be read, edited, and transported to the MCU.

In order to program and debug the MCU, a Universal Evaluation Board (EVBU) is used. This board, with an MCU inserted, has many of the same functions as the final version of the motor controller. It provides the MCU with regulated power, a RS-232 serial interface, an oscillator, and access to all MCU
pin logic levels. Since the MCU is a self-contained computer, some software must be present and running before any meaningful communication with any other system can occur. Two software applications, provided by Motorola with the EVBU, are BUFFALO and PCbug11. While allowing the user to perform nearly the same tasks, these two applications work in very different ways.

BUFFALO is a complex piece of code that must be previously loaded into MCU memory. When the EVBU is reset, BUFFALO begins execution. This program allows the user to use a VT100 terminal emulation program and serial link to connect to the MCU. Once the connection is established, code can be loaded into RAM or EEPROM, executed, and debugged. The two most significant limitations of BUFFALO are that it must already be loaded into MCU memory and that it cannot modify EPROM. BUFFALO is well documented and discussed in the User's Manual provided with the EVBU.

PCbug11 is a DOS-based application capable of connecting to an unprogrammed MCU. During initialization, a small program, called a talker, is loaded into MCU memory. This small but powerful piece of code, allows the user to read and program any available MCU memory location in RAM, EEPROM, or EPROM. Since only a small portion of memory is used temporarily for the talker, a much larger block of code can be transferred to MCU memory. The use of PCbug11 is documented in the PCbug11 User's Manual.

One important feature of the MCU is the receiver wake-up operation. When multiple controllers are used, every controller receives any message sent
by a controller on the network. A system must be devised to allow a receiving MCU to determine if it is being addressed. The address-mark wake-up feature, available on the 68HC11 MCU, solves this dilemma. Each MCU is placed in a dormant state by enabling the RWU bit in the SCCR2 register. In order to select any controller, a byte of information must be sent in which the most significant bit is set. The remaining seven bits are used as a coded address. This byte of information will wake up each controller. The software running on each MCU is responsible for determining if the encoded address matches its own. If no match exists, the software is responsible for placing the MCU back in the dormant mode.

**Master Controller**

The master controller includes a 5 V voltage regulator, a M68HC11 microcontroller, a crystal oscillator circuit, and a reset circuit. In addition, the master controller, detailed in Appendix A, also contains an external 32K RAM chip, a claw solenoid activation circuit, the conditioning circuit for the tensiometer switch, and serial communications hardware for five serial ports. The external RAM allows for increased program size and faster reprogramming time than using internal EPROM. Using this external memory requires that the MCU be used in expanded mode. As a result, ports B and C are no longer usable as external I/O pins. The claw activation circuit consists of a transistor driven relay switch. The conditioning circuit for the tensiometer is discussed in the tensiometer section. The communication hardware consists of three RS-232
serial port drivers and one asynchronous communications interface adapter (ACIA). Two of the RS-232 drivers each control two serial ports. These four ports, connected to the TxD and RxD MCU pins, are used to communicate with the motor controllers. The last driver is used, along with the ACIA, to allow serial communication with the supervisor computer via the Port C data bus.

**Motor Controller**

Like the master controller, the motor controller, detailed in Appendix B, contains a 5 V voltage regulator, a M68HC11 microcontroller, a crystal oscillator circuit, and a reset circuit. In addition, it contains optical encoder decoding circuit and one RS-232 serial driver. The decoding circuit is discussed in the section on optical encoders. The single serial driver is used to allow communication with the master controller via the RxD and Txd MCU pins.
Chapter Three: CONTROL SOFTWARE DESCRIPTION

Each of the four microcontrollers contains embedded control software. These programs were coded in assembly language for the Motorola M68HC11 series microcontroller. Each program is written as a text file, assembled, and downloaded to the microcontroller RAM, EPROM, or EEPROM using PCBUG11 or BUFFALO. A listing of the code used in the master controller can be found in Appendix C. A listing of the code used in a typical motor controller can be found in Appendix D. Appendix E contains Motorola application notes outlining the steps necessary to write, assemble, store, and run a piece of sample code.

The master controller contains the code necessary to service the supervisor computer, the claw actuator and sensor, and each of the three motor controllers. The motor controller code, identical in each case except for constants defining position limits, controller address, and motor speed, is responsible for driving and sensing motor position and communicating with the master controller.

Since the exploratory vehicle system is currently under construction, an IBM-compatible 486DX-33 personal computer running PROCOMM terminal emulation software is used in place of the supervisor computer. Any computer with a 9600 baud serial connection and software to access that port can be substituted for the supervisor computer.
Discussion of the two programs is divided into four sections. Each section focuses on the algorithm used to control the interaction between two hardware component systems.

A. Supervisor / Master Controller Loop

The link between the supervisor and the master controller has two functions. First, the supervisor must provide the master controller with an ASCII string containing platform orientation and target coordinates. Second, the master controller provides claw trajectory information that can be used for system monitoring. The supervisor must examine the sample object and resubmit the command if the correct object was not retrieved.

All data transfer is accomplished via a 9600 baud serial connection. Data strings are in ASCII format to allow easier debugging and system monitoring. These strings are converted into hexadecimal values upon receipt by the master.

Each time the master receives information from a motor controller or sensor, a character string is sent to the supervisor. This string contains position information for each motor as well as the current state of the claw actuator. Motor positions are written as absolute angles, in degrees, in hexadecimal form.

B. Master Controller / Motor Controller Loop

The master controller, upon receiving the approximate target location, calculates the desired motor positions. These positions are functions of the desired position \((\theta, r, z)_d\) and platform orientation \((p, q)\). The master algorithm
approximates the motor positions by assuming that the platform is level during manipulator operation. This assumption decouples the effects of motor positions on claw position. Each motor is assumed to control one degree-of-freedom and have no effect on the other two degrees-of-freedom.

Since the claw is suspended on a cable, it will always move along a vertical line. When pitch and roll are both zero, this vertical line is parallel with the z axis. In this situation, the distance that the claw must be extended or retracted is a function only of the desired z coordinate. Similarly, the trolley motor only affects r and the boom motor only affects θ. When some platform pitch or roll exists, there will be an error in claw position whose magnitude varies with motor positions and platform orientation. Since the object of the maneuver is not to reach some given position, but rather to find some object within the search area, the claw position error only becomes a problem if it is sufficient to position the search area away from the target object. The amount of error that is acceptable in the system is a function of the search pattern area and grid resolution.

Given that the platform orientation is neglected, the system is completely decoupled. As a result, each motor controls a single degree-of-freedom. The desired motor positions are calculated based on measured values of trolley and claw travel in degrees per inch, measured motor angles when the claw is in the home position, and the desired cylindrical coordinates. Values for these constants can be found in the software listings included in Section 0.
The master controller issues all commands to the motor controllers and claw in a serial fashion. After sending each motor command, the master waits for the motor controller to achieve and feedback its desired position. Two routines, docmds and docmds2, are used to issue a string of commands. The first, docmds, performs them in the order of boom motion, trolley motion, claw motion, and claw activation. The second, docmds2, commands the motors in the reverse order and but still activates the claw last. The first routine is used to approach an object. Since this object could be in a depression, the claw is kept at as high as possible until directly over the target site. The second routine, used after the object is captured, lifts the claw completely before moving the boom or trolley.

A portion of the main routine of the master controller software, called whenever a valid target position input string is received, is listed below.

```
jsr findit
jsr putaway
jsr gohome
```

The first routine, findit, is responsible for searching for and grasping the object. The search algorithm is addressed in Chapter 4. The second routine, putaway, directs the claw to move to a receptacle and release the object. The last routine, gohome, sends the claw back to its home position.

**C. Motor Controller / Motor Loop**

The motor controller receives a hexadecimal number, in ASCII characters, representing the desired absolute angle of the motor. The actual motor position
is read from the decoder chip and converted into degrees. These two values are compared and a desired rotation, in degrees, is calculated. This number is converted into motor steps. The motor controller then drives the direction pin on the motor driver high for forward motion or low for reverse motion. A square wave is then applied to the driver input pin. The number of pulses in this wave corresponds to the number of desired steps. The frequency of the wave determines motor speed. The motor controller again reads the decoder chip and the process is repeated as necessary. An error of $1^\circ$ is allowed between desired and final actual motor position. This allowance is necessary due to the encoder resolution and integer division necessary to convert the encoder value to degrees. When the final position is attained, the motor controller echoes its current absolute position to the master controller.

**D. Master Controller / Claw Loop**

The master controller, in addition to performing high level motor position control and communication with the supervisor, is responsible for claw activation and obstacle contact detection. The claw is commanded via a relay on the master controller circuit board. Contact between the claw and some obstacle is monitored via a boom mounted switch and conditioning circuitry mounted on the master controller circuit board.

The master controller enables or disables the claw by varying the state of one of the microcontroller output pins. The pin indirectly drives the claw solenoid using a transistor and a relay.
A sudden and sustained loss of tension occurs as a result of contact between the claw and some surface. The loss of cable tension causes the activation of an interrupt service routine on the master controller to command the motor controller to stop and slightly raise the claw. This motion allows the claw to better grip the target object.

This feature is vital to the success of the searching algorithm. The claw is closed to minimize the projected area on the work surface and to limit contact to one point instead of three points. The claw is then lowered at predetermined points until contact is made. The absolute heights of these points are stored until all points are searched. After all nine heights are measured, they are converted into quarters of an inch above the lowest of the nine points. The predetermined object height is 1.25 in. Software selects the first of the nine points that happens to be higher than 1 in. If no object tall enough is found then the search pattern may be repeated at another location. In general, a complete search includes the mapping of 47.5 in\(^2\). The search pattern and mapping technique are discussed in more detail in Chapter 4.
Chapter Four: APPLICATION OF CONTROL SOFTWARE TO CRANE SYSTEM

A. Test Scenarios

A series of three test cases was used to determine the ability of this manipulator device to retrieve an object. In case one, the claw moves to a specified location, grips the object if one exists, moves to the drop zone, releases the object, and returns home. In case two, the boom and trolley are moved to the positions specified. At this point, the claw is lowered until it makes contact with the ground or an object. The claw then lifts slightly, grips the object, and completes the maneuver. In case three, a relief map of the area surrounding the point of initial contact is created. Based on information contained in this map and prior knowledge of the target object, either the object is located or a new area is searched. Ultimately, either the object is found and the maneuver completed or the search is abandoned.

In case one, the claw moves toward the specified location \((\theta, r, z)_d\) by first swinging the boom into position \((\theta_d)\) and then moving the trolley \((r_d)\). Once the claw is suspended above the desired point, it is lowered to the appropriate height \((z_d)\). In the trajectory used in this case, only one motor is in motion at any given time. Controller hardware does not limit the system to this serial motion. This method is used to avoid undesired contact between the claw and the environment. Due to the crane structure, any motion of boom or trolley requires
that no obstacles be present in the space through which the suspended claw moves. Keeping the claw retracted until all other motion is complete decreases the chances of an unwanted collision with obstacles in the workspace. Upon reaching the desired location, the claw is activated. No method of target object confirmation is currently in place as part of this system. The vehicle supervisor computer is responsible for confirming that the correct object was retrieved using some part of its sensor array. The object, having been retrieved, is moved to a previously defined point and dropped into a receptacle. The claw is then returned to home position.

The key difference in case two is that the final height of the claw (z) is not necessarily the height specified (zd). When the exact height is unknown by the supervisor, a value at the limit or beyond the reachable workspace is used. Whenever the claw makes contact with some object before the specified height is reached, the claw descent is halted by the master controller. Next, the claw suspension cable is retracted a distance of between one-half inch and one inch. This claw height above the ground was determined by trial-and-error to be ideal for gripping the target object. The claw is then closed and the remainder of the maneuver is identical to case one.

In case three, the specified boom and trolley positions (θ, r)d define the central vertical axis of a search space instead of the vertical line along which the object lies. As in case two, the claw is lowered until it makes contact with the workspace. In case three, however, this motion occurs at least at the nine points
illustrated in grid 1 of Figure 4-1. The height coordinate (z) at each point is measured and is normalized by subtracting the lowest height found. The heights are then converted to a number of quarter inches. A typical resulting relief map is illustrated in Figure 4-2. Since the object dimensions are assumed known, the maximum measured height can be compared with a minimum anticipated object height. If the object is determined to exist at one of the search points, then the maneuver is completed as in the first two cases. If the object is not found at any of the search points, then a new section of the workspace is searched. This mapping process is repeated in the pattern shown in Figure 4-1 until either all the specified areas are searched or the object is found.

Figure 4-1: Search Pattern
Figure 4-2: Typical Relief Map of Minor Grid

B. Results and Evaluation

In case one, the object was retrieved if the boom position was within approximately 3°, the trolley position was within 0.5 in, and the claw height was within 0.5 in of the object center. The primary limitation of this scheme is that the specified position provided by the vehicle supervisor must be fairly accurate. Precise object location may prove difficult for the vehicle sensors to measure due to the small size of the object, the natural terrain background, and the difficulty in sensing within 8 in of the vehicle. Actuator output is plotted in Figure 4-3. The target position used in this case is +00,+00,-180,+07.0,-10.0.

In case two, the burden of sensing object height is shouldered by the manipulator device. The vehicle sensors must still synthesize a fairly accurate two dimensional image of the object as viewed from above. While an
improvement over case one, the vehicle sensors still limit mission success. The
target object will typically be sighted before it is within reach of the manipulator.
The vehicle must then move into position and stop before retrieval can occur. As
in case one, this close proximity surface may be difficult to map. Actuator output
is plotted in Figure 4-4. The target position used in this case is +00,+00,-
180,+07.0,-10.0. Note that the only difference in case one and case two is that
the claw height did not reach -10 in. Instead, a collision with an obstacle forced
the claw to stop and grip at approximately -2 in.

Case three actuator outputs are illustrated in Figure 4-5. The boom angle,
trolley radius, and claw height all demonstrate the multipoint sounding technique
used to map the area. As in the previous two cases, the input string was
+00,+00,-180,+07.0,-10.0. In this case, however, the object was actually located
at +00,+00,-150,+07.0,-02.0. The resulting relief map is illustrated in Figure 4-6.
In this case, the specified coordinates must only be accurate enough to ensure
that the object is within the search area. As long as the object is within two
horizontal inches in the radial direction and four horizontal inches in the
tangential direction of the specified position, the object can be found and
retrieved. The ability of the manipulator to map the area and find the object is a
big advantage. Mission success in retrieving the object is much more likely with
this scenario. The major limitation of this method is that the maximum resolution
of the search pattern is limited by the physical dimensions of the claw. Due to
the large diameter of the closed claw, undesired contact between claw and
object sometimes occurs. This unwanted contact can cause the object to move to a previously mapped location and be missed in the search or the claw height to be inaccurately measured. A future modification of the system would be to modify the claw so that contact between the claw and objects not directly beneath it would be reduced.
Figure 4-3: Case 1 Actuator Output
Figure 4-4: Case 2 Actuator Output
Figure 4-5: Case 3 Actuator Output
In all three cases, an intermittent hardware or software error occasionally gives grossly inaccurate count values in motor position. This problem has been resolved in software by comparing the actual counter value and an estimated value. When gross differences occur, the estimate is used by the motor controller and the master controller is notified.
Chapter Five: CONCLUSIONS AND SUGGESTIONS FOR FUTURE WORK

The manipulator system successfully meets the stated requirements of size, weight, and functionality. A clay cylinder measuring 1.25 in by 1.25 in was used as the target object during system testing. Given an initial boom angle within 24° and an initial trolley position within 4 in of the actual values, the system will locate and retrieve the target object approximately 80% of the time.

The search and recovery algorithm eliminates the need for precisely measured target coordinates. This feature allows the rover to dedicate its sensors to more important data gathering. It also eliminates the need for sensors dedicated to surveying the area within 8 in of the vehicle base. The vehicle can locate an object at a distance limited only by its sensor arrays, move into position near the object, and provide the manipulator with estimated coordinates relative to the vehicle.

The manipulator does occasionally miss the target in its search and recovery effort. In these cases, the claw typically causes the object to move into an already measured point when sounding the area immediately surrounding the object. As a result, the object does not show up in the search map. In these situations, the vehicle sensor system will sense the absence of the target object and will resubmit the command. This tactic of multiple attempts is usually successful.
The system is constructed in a modular form that will allow future users to optimize the interaction between the vehicle system and the manipulator subsystem. Sensors and actuators can be modified to provide new performance characteristics. Control software can be altered for testing of new control and search algorithms.

Modification of the end effector could improve system effectiveness in three ways. A smaller claw would both allow a higher resolution mapping algorithm and reduce unwanted claw/object interaction. This change will allow a finer resolution and smaller object search capabilities. Redesigning the fingers of the claw could increase the acceptable variation of object size, shape, and consistency.

The master controller software listed in the Appendices neglects the effects of platform orientation on actual position. Since the claw is suspended by a cable, a change in position of the motor controlling claw height results in claw motion along the z axis of the world coordinate system. Any pitch or roll of the vehicle will cause the z axis of the manipulator coordinate system to no longer be parallel with the z axis of the world coordinate system. Inclusion of these effects when calculating desired motor positions based on a specified target coordinates would result in increased accuracy in claw placement.
Chapter Six: REFERENCES


A. Master Controller Circuit Diagram
B. Motor Controller Circuit Diagram
C. Master Controller Program Listing
MASTER SOFTWARE FOR MARS ROVER MANIPULATOR

Based on:
08/19/94 ct0.asm acia in / sci out ;with sequence input ok
--------- CK.Chao

Modifications:
11/01/94 mast1.asm Read MAXZN bytes of data & echo to motor CPU.
11/07/94 mast2.asm Run main loop until command received from supervisor. Echo command once then return to main loop
11/08/94 mast3.asm PA7 commands claw upon receipt of C1 command.
C(anything besides 1) results in claw off command. Any other string is sent out SCI
11/10/94 added lines in INITPA to disable IRQ on pins PA0, PA1, PA2, PA3.
12/01/94 mast4.asm -- Modified input command structure to address(one byte hex), theta(four ASCII characters representing a two byte hex).
12/03/94 mast5.asm -- Receive and echo output from motor controllers.
Each motor controller sends the master address (5), four ASCII characters representing a two byte hex number (the motor position in degrees), two ASCII characters representing the controller mode (closed loop (C0) or open loop (O1, O2, or O3)), and four ASCII characters representing the command received by the motor (desired motor position in degrees).
12/12/94 mast6.asm -- Rearrange functions to allow for easier inclusion of conversion routine. Output printed whenever flag (ffbk1,ffbk2,ffbk3,ffbk4) set. Input command syntax modified to "pp,ixq, ittt, irr.r, izz.s". Command syntax is verified after receipt. An error message is written to ACIA if an invalid command is received.
01/03/95 mast7.asm -- Configure PA3 (IC4) as the interrupt pin (on falling edge) indicating loss of tension in crane cable. Interrupt Service Routine (SLAK_ISR) will be called. Modified putaway routine to take claw all the way to the top before moving to drop zone.
01/09/95 mast8.asm -- Add routines to search around nominal target for high spot. After identification, lift at high spot. Modify output format to eliminate mode information and add 0 as prefix and h as suffix to angle information. This will allow Mathcad plot. Add PRNTDAT call in ACIA_ISR to print home position after valid input string is added to home position to plot. Added routine findit to search for object around input (theta,r). Moved "docms" from main.
**********
* jump table
**********
org #$00ee
jmp ACIA_ISR
org #$00e2
jmp SCI0_ISR
org #$00e5
jmp SCI1_ISR
org #$00e8
jmp SCI2_ISR
org #$00d3
jmp SLAK_ISR

**********
* EEPROM contents
**********
ORG del_r
FDB 2
ORG del_theta
FDB 8

**********
* the variables
**********
org $R_DATA

rdata rmb MAXIN
box1 rmb MAXIN
box2 rmb MAXIN
box3 rmb MAXIN
box4 rmb MAXIN
box5 rmb MAXIN
box6 rmb MAXIN
box7 rmb MAXIN
box8 rmb MAXIN
cntr1 rmb MAXIN
cntr2 rmb MAXIN
cntr3 rmb MAXIN
cntr4 rmb MAXIN
cntr5 rmb MAXIN
cntr6 rmb MAXIN
cntr7 rmb MAXIN
cntr8 rmb MAXIN
cntr9 rmb MAXIN
hex2 rmb 1
sci0_in rmb MAXSCI
scil_in rmb MAXSCI
sci2_in rmb MAXSCI
datain rmb 1
ffbk2 rmb 1
ffbk3 rmb 1
fcmdc rmb 1
fcmd1 rmb 1
fcmd2 rmb 1
fcmd3 rmb 1
fstop rmb 1
pitch rmb PCHAR
roll rmb QCHAR
theta rmb TCHAR
radius rmb RCHAR
height rmb ZCHAR
th1 rmb 2
th2 rmb 2
th3 rmb 2
th3a rmb 4
th3e rmb 2
temp rmb 2
tmp16 rmb 2
clawcmd rmb 1
stpbgt rmb 2
hgt1 rmb 2
hgt2 rmb 2
hgt3 rmb 2
hgt4 rmb 2
hgt5 rmb 2
hgt6 rmb 2
hgt7 rmb 2
hgt8 rmb 2
hgt9 rmb 2
target rmb 1
try rmb 1
whole_in rmb 2
half_in rmb 2
stack rmb 1
SHIFTREG RMB 2  input shift register
TMP1 RMB 1

org $4500    * eventually B700
del_x rmb 2
del_theta rmb 2

***********
* the code starts here.
***********
org $2000
jmp START

**************************************************************************
* THE FUNCTION LIBRARY
**************************************************************************
***********
* bin2hex -- Separates each character of a hex number and calls outhex.
***********
bin2hex pshb
      psbb
      lsrb
      lsrb
      lsrb
      lsrb
      anda #00000111
      jsr  outhex
      pula
      anda #00000111
      jsr  outhex
zh2hex pulb
rts

***********
* getsci -- if a character has been received on sci port, this character
* retrieves and places in accumulator A.
***********
getsci ldx $REGBR
        ldab SCSR,X  * if RDRF is 0 then wait
        bitb #20
        beq getsci
        ldab SCSR,X
zg_sci rts

***********
* HEXBIN(a) - Convert the ASCII character in A
* to binary and shift into shftreg.
***************
HEXBIN  PSHA
        PSHB
        PSHX
  *      JSR  UPCASE  convert to upper case
  CMPA  #$'0'
  BLT  HEXNOT  jump if a < $30
  CMPA  #$'9'
  BLT  HEXNMB  jump if 0-9
  CMPA  #$'A'
  BLT  HEXNOT  jump if $39 > a < $41
  CMPA  #$'F'
  BGT  HEXNMB  jump if a > $46
  ADDA  #$9
  JMP  convert $A-$F
HEXNMB  ANDA  #$OF
        LDX  #$SREG
        LDAB  #4
        HEXSHFT  ASL 1,X 2 byte shift through carry bit
        ROL 0,X
        DECB
        BGT  HEXSHFT
        ORAA 1,X
        STAA 1,X
        BRA  HXXRTS
HEXNET  NOP
*      INC  TMP1  indicate not hex
HEXRTS  PULX
        PULB
        PULA
        RTS

***************
*  onacia  -- initializes acia port
***************
onacia:  ldx  #REGBAS
        cl1
        ldda  OPTION.X
        ora  #$00100000
        staa  OPTION.X
        ldda  #$03
        staa  ACIA
        ldda  #$00010110
  *
        ora  #$10000000
        staa  ACIA
zonacia  rts

***************
*  onsci  -- initializes sci port
***************
onsci:  ldx  #REGBAS
        ldda  $B96
        staa  BUD.X
        ldda  #$0001110
        staa  SCCR2,X
        ldda  #$00000000
        staa  SCCR1,X
zonsci  rts

***************
*  outhex  -- converts hex to ASCII and transmits out ACIA port
***************
  outhex  cmpa  #$10
           bge  ge2A
           adda  #$30
           jsr  putacia
           bra  zo_hex
ge2A:  adda  #$37
       jsr  putacia

51
zo_hex rts

**********
* outhexs -- converts hex to ASCII and transmits out SCI port
**********
outhexs cmpa #10
    bge ge2as
    adda #$30
    jsr putsci
    bra zo_hexs
ge2as: adda #$37
    jsr putsci
zo_hexs rts

**********
* putacia -- puts byte in accumulator A out acia port.
**********
putacia ldab ACIA
    bitb #$02
    beq putacia
    anda #$ff
    staa ACIA+1
zp_acia rts

**********
* putsci -- puts byte in accumulator A out sci port.
**********
putsci idx SCRB,X
    ldab #REGBAS
    bitb #$80
    beq putsci
    anda #$ff
    staa SCDR,X
zp_sci rts

**********
* scib2h -- Separates each character of a hex number and calls outhexs.
**********
scib2h psha
    lsra
    lsra
    lsra
    anda #%00001111
    jsr outhexs
    pulb
    anda #%00001111
    jsr outhexs
zscib2h rts

**********
* SLODOWN -- kills time when necessary
**********
SLODOWN psha
    pshb
    ldad #$ffff
SLO1 subd #$8001
    bne SLO1
    pulb
    pula
    rts

	

**********
* THE FUNCTIONS SPECIFIC TO MASTER OPERATION
**********

**********
* ASC2HEX -- Converts an ASCII character (0-9) in accumulator B to a
hex number. This number is returned in accumulator B.

```
*       ASC2HEX cmpb #0' blt za2h
cmpb #9' bgt za2h
subb #$30
za2h rts
```

```
******

checkin -- Read input string and verify syntax
******

checkin ldy #R_DATA
ldaa 0,Y
jsr isassign
cmpb #1'
bne syner1
ldaa 1,Y
jsr isint
cmpb #1'
bne syner1
ldaa 2,Y
jsr isint
cmpb #1'
bne syner1
ldaa 3,Y
jsr iscomsp
cmpb #1'
bne syner1
ldaa 4,Y
jsr isassign
cmpb #1'
bne syner1
ldaa 5,Y
jsr isint
cmpb #1'
bne syner1
ldaa 6,Y
jsr isint
cmpb #1'
bne syner2
ldaa 7,Y
jsr iscomsp
cmpb #1'
bne syner2
ldaa 8,Y
jsr isassign
cmpb #1'
bne syner2
ldaa 9,Y
jsr isint
cmpb #1'
bne syner2
ldaa 10,Y
jsr isint
cmpb #1'
bne syner2
ldaa 11,Y
jsr isint
cmpb #1'
bne syner2
ldaa 12,Y
jsr iscomsp
cmpb #1'
bne syner2
ldaa 13,Y
jsr isint
cmpb #1'
bne syner2
ldaa 14,Y
jsr iscomsp
cmpb #1'
bne syner2
ldaa 15,Y
jsr isint
cmpb #1'
bne syner2
ldaa 16,Y
jsr iscomsp
cmpb #1'
bne syner2
ldaa 17,Y
jsr isint
cmpb #1'
bne syner2
ldaa 18,Y
jsr iscomsp
cmpb #1'
bne syner2
ldaa 19,Y
jsr isint
cmpb #1'
bne syner2
ldaa 20,Y
jsr iscomsp
cmpb #1'
bne syner2
ldaa 21,Y
jsr isint
cmpb #1'
bne syner2
ldaa 22,Y
jsr iscomsp
cmpb #1'
bne syner2
ldaa 23,Y
jsr isint
cmpb #1'
bne syner2
ldaa 24,Y
jsr iscomsp
cmpb #1'
bne syner2
ldaa 25,Y
jsr isint
cmpb #1'
bne syner2
ldaa 26,Y
jsr iscomsp
cmpb #1'
bne syner2
ldaa 27,Y
jsr isint
cmpb #1'
bne syner2
ldaa 28,Y
jsr iscomsp
cmpb #1'
bne syner2
ldaa 29,Y
jsr isint
cmpb #1'
bne syner2
ldaa 30,Y
jsr iscomsp
cmpb #1'
bne syner2
ldaa 31,Y
jsr isint
cmpb #1'
bne syner2
ldaa 32,Y
jsr iscomsp
cmpb #1'
bne syner2
ldaa 33,Y
jsr isint
cmpb #1'
bne syner2
ldaa 34,Y
jsr iscomsp
cmpb #1'
bne syner2
ldaa 35,Y
jsr isint
cmpb #1'
bne syner2
ldaa 36,Y
jsr iscomsp
cmpb #1'
bne syner2
ldaa 37,Y
jsr isint
cmpb #1'
bne syner2
ldaa 38,Y
jsr iscomsp
cmpb #1'
bne syner2
ldaa 39,Y
jsr isint
cmpb #1'
bne syner2
ldaa 40,Y
jsr iscomsp
cmpb #1'
bne syner2
ldaa 41,Y
jsr isint
cmpb #1'
bne syner2
ldaa 42,Y
jsr iscomsp
cmpb #1'
bne syner2
ldaa 43,Y
jsr isint
cmpb #1'
bne syner2
ldaa 44,Y
jsr iscomsp
cmpb #1'
bne syner2
ldaa 45,Y
jsr isint
cmpb #1'
bne syner2
ldaa 46,Y
jsr iscomsp
cmpb #1'
bne syner2
ldaa 47,Y
jsr isint
cmpb #1'
bne syner2
ldaa 48,Y
jsr iscomsp
cmpb #1'
bne syner2
ldaa 49,Y
jsr isint
cmpb #1'
bne syner2
ldaa 50,Y
jsr iscomsp
cmpb #1'
bne syner2
ldaa 51,Y
jsr isint
cmpb #1'
bne syner2
ldaa 52,Y
jsr iscomsp
cmpb #1'
bne syner2
ldaa 53,Y
jsr isint
cmpb #1'
bne syner2
ldaa 54,Y
jsr iscomsp
cmpb #1'
bne syner2
ldaa 55,Y
jsr isint
cmpb #1'
bne syner2
ldaa 56,Y
jsr iscomsp
cmpb #1'
bne syner2
ldaa 57,Y
jsr isint
cmpb #1'
bne syner2
ldaa 58,Y
jsr iscomsp
cmpb #1'
bne syner2
ldaa 59,Y
jsr isint
cmpb #1'
bne syner2
ldaa 60,Y
jsr iscomsp
cmpb #1'
bne syner2
ldaa 61,Y
jsr isint
cmpb #1'
bne syner2
```

53
* docmds -- Move manipulator to commanded position

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

docmds ldaa fcmd1
        cmpa #0'
beq doem1
jsr outcmd1
wait1 ldaa ffbk1
cmpa #'1'
beq adocmds
bra wait1
adocmds jsr doout
doem1 ldaa fcmd2
cmpa #'0'
beq doem2
jsr outcmd2
wait2 ldaa ffbk2
cmpa #'1'
beq bdocmds
bra wait2
bdocmds jsr doout
doem2 nop
lda fcmd3
cmpa #'0'
beq doem3
jsr outcmd3
wait3 ldaa ffbk3
cmpa #'1'
beq cdocmds
ldaa fstop
cmpa #'1'
bne wait3
ldaa #'0'
staa fstop
staa fcmd1
staa fcmd2
bra bdocmds
cdocmds jsr doout
doem3 jsr grip
jsr doout
bset TFLG1,X #%00001000  * Clear IC4 flag
bset TMSK1,X #%00001000  * Enable IC4 interrupt
ddocmds ldaa #'0'
staa datain
zdocmds rts

*******
* docmds2 -- Move manipulator to commanded position in reverse order
*******

docmds2 ldaa fcmd3
cmpa #'0'
beq doem21
jsr outcmd3
wait21 ldaa ffbk3
cmpa #'1'
beq adocmd2
bra wait22
adocmd2 jsr doout
doem21 ldaa fcmd2
cmpa #'0'
beq doem22
jsr outcmd2
wait22 ldaa ffbk2
cmpa #'1'
beq bdocmds2
bra wait22
bdocmds2 jsr doout
doem22 ldaa fcmd1
cmpa #'0'
beq doem23
jsr outcmd1
wait23 ldaa ffbk1
cmpa #'1'
beq cdocmds

************
* dout -- write output to screen is necessary.
************

doout
  ldaa ffbk1
  suba $30
  adda ffbk2
  suba $30
  adda ffbk3
  suba $30
  adda ffbkc
  suba $30
  cmpa $00
 .beq zdout

ldaa # '0'
  staa ffbk1
  staa ffbk2
  staa ffbk3
  staa ffbkc
  jsr PRNTDUT

zdout

************
* findit -- Search for object around given approximate position. Algorithm
* assumes that the beginning (input) point is the beginning of
* the search. This point lies at the center of an imaginary
* tic-tac-toe board. All nine points are sounded and their
* heights (th3 values) are stored in hgt1-hgt9. If the center
* point is del_z above the perimeter, then the claw makes a
* grab at the center point. If any of the perimeter squares
* are del_z taller than the center,
* that point becomes the new center and the search pattern is
* run again. If two perimeter points are both equal and
* taller than the center, then... This process repeats
* until...
* if (hgti-del_z > all other hgt values)
*  Lift at center
* Else if (hgti
*  Case 2:
*  
*  cntrl (i=1:9) are the centers of imaginary tic-tac-toe
*  boards that will be used in the search for the object.
*  boxi (i=1:9) are the individual squares of the particular
*  tic-tac-toe board currently being searched.

************
findit
  psha
  pshb
  pshx
  pshy

  ldy #rdata   * Save rdata to cntrl for later use.
  ldx #cntrl

svcntrl ldaa 0,Y
  staa 0,X
iny
inx
  cpw #rdata+#MAXIN
  blt svcntrl

  ldy #rdata   * Save rdata to cntrl2 for later use.
  ldx #cntrl2

svcntrl2 ldaa 0,Y
* Save rdata to cntr3 for later use.

* Save rdata to cntr4 for later use.

* Save rdata to cntr5 for later use.

* Save rdata to cntr6 for later use.

* Save rdata to cntr7 for later use.

* Save rdata to cntr8 for later use.

* Save rdata to cntr9 for later use.
jsr pattern
lda $1'
staa try

search
ldy #rdata
ldaa try

try1
cmpa #1'
beq try2
ldx #cntr1
bra strdata

try2
cmpa #2'
beq try3
ldx #cntr2
bra strdata

try3
cmpa #3'
beq try4
ldx #cntr3
bra strdata

try4
cmpa #4'
beq try5
ldx #cntr4
bra strdata

try5
cmpa #5'
beq try6
ldx #cntr5
bra strdata

try6
cmpa #6'
bne nomore
ldx #cntr6
bra strdata

nomore
jmp giveup

strdata
lda 0, X
staa 0, Y
iny
inx
cpy #rdata+#MAXIN
blt strdata

ldy #rdata
ldx #box1

svrdat1
lda 0, Y
staa 0, X
iny
inx
cpy #rdata+#MAXIN
blt svrdat1

ldy #rdata
ldx #box2

svrdat2
lda 0, Y
staa 0, X
iny
inx
cpy #rdata+#MAXIN
blt svrdat2

ldy #rdata
ldx #box3

svrdat3
lda 0, Y
staa 0, X
iny
inx
cpy #rdata+#MAXIN
blt svrdat3

ldy #rdata
ldx #box4

svrdat4
lda 0, Y

* Modify cntr values to show spread.

* Set cntr to search around first

* Save cntr to rdata to be searched. If
* all 9 grids have been searched, then
* give up.

* six square case

* Save rdata to box1 for later use.

* Save rdata to box2 for later use.

* Save rdata to box3 for later use.

* Save rdata to box4 for later use.
staa 0,X
iny inx cpy #rdata+#MAXIN
blt svrdat4
ldy #rdata
ldx #box5
svrdat5 ldla 0,Y staa 0,X
iny inx cpy #rdata+#MAXIN
blt svrdat5
ldy #rdata
ldx #box6
svrdat6 ldla 0,Y staa 0,X
iny inx cpy #rdata+#MAXIN
blt svrdat6
ldy #rdata
ldx #box7
svrdat7 ldla 0,Y staa 0,X
iny inx cpy #rdata+#MAXIN
blt svrdat7
ldy #rdata
ldx #box8
svrdat8 ldla 0,Y staa 0,X
iny inx cpy #rdata+#MAXIN
blt svrdat8
ldy #rdata
ldx #box9
svrdat9 ldla 0,Y staa 0,X
iny inx cpy #rdata+#MAXIN
blt svrdat9
jsr pattern * Modify box positions to spread pattern
ldaa #'1'
staa clawcmd
jsr docmds
ldy #rdata
ldx #box1
ldrdat1 ldla 0,X staa 0,Y
iny inx cpy #rdata+#MAXIN
blt ldrdat1
ldaa #'1'
staa datain
jsr makecmd
* Save rdata to box5 for later use.
* Save rdata to box6 for later use.
* Save rdata to box7 for later use.
* Save rdata to box8 for later use.
* Save box1 to rdata for sounding.
* Close claw
* Sound box1

59
JSR  DECMDS
LDD  #TH3S
STD  #BGT1
JSR  GOTO

LDA  #RDATA
LDA  #BOX2

LDRDAT2 LDAA  0,X
STA  0,Y
INX
INX
CPY  #RDATA+MAXIN
BLT  LDRDAT2

LDA  '#1'
STA  DATAIN
JSR  MAKECMD
JSR  DECMDS
LDD  #TH3S
STD  #BGT2
JSR  GOTO

LDA  #RDATA
LDA  #BOX2

LDRDAT3 LDAA  0,X
STA  0,Y
INX
INX
CPY  #RDATA+MAXIN
BLT  LDRDAT3

LDA  '#1'
STA  DATAIN
JSR  MAKECMD
JSR  DECMDS
LDD  #TH3S
STD  #BGT3
JSR  GOTO

LDA  #RDATA
LDA  #BOX3

LDRDAT5 LDAA  0,X
STA  0,Y
INX
INX
CPY  #RDATA+MAXIN
BLT  LDRDAT5

LDA  '#1'
STA  DATAIN
JSR  MAKECMD
JSR  DECMDS
LDD  #TH3S
STD  #BGT5
JSR  GOTO

LDA  #RDATA
LDA  #BOX5

LDRDAT6 LDAA  0,X
STA  0,Y
INX
INX
CPY  #RDATA+MAXIN
BLT  LDRDAT6

LDA  '#1'
STA  DATAIN
JSR  MAKECMD
JSR  DECMDS
LDD  #TH3S
STD  #BGT6
JSR  GOTO

LDA  #RDATA
LDA  #BOX6

* Save box2 to rdata for sounding.  *
* Sound box2  *
* Save box3 to rdata for sounding.  *
* Sound box3  *
* Save box6 to rdata for sounding.  *
* Sound box6  *
* Save box5 to rdata for sounding.  *
* Sound box5  *
ldd th3s
std hgt5
jsr gotop
ldy #rdata
ldx #box4
ldrdat4 ldaa 0,X
staa 0,Y
iny
inx
cpy #rdata+#MAXIN
blt ldrdat4

ldaa #1
staa datain
jsr makecmd
jsr docmds
ldd th3s
std hgt4
jsr gotop
ldy #rdata
ldx #box7
ldrdat7 ldaa 0,X
staa 0,Y
iny
inx
cpy #rdata+#MAXIN
blt ldrdat7

ldaa #1
staa datain
jsr makecmd
jsr docmds
ldd th3s
std hgt7
jsr gotop
ldy #rdata
ldx #box8
ldrdat8 ldaa 0,X
staa 0,Y
iny
inx
cpy #rdata+#MAXIN
blt ldrdat8

ldaa #1
staa datain
jsr makecmd
jsr docmds
ldd th3s
std hgt8
jsr gotop
ldy #rdata
ldx #box9
ldrdat9 ldaa 0,X
staa 0,Y
iny
inx
cpy #rdata+#MAXIN
blt ldrdat9

ldaa #1
staa datain
jsr makecmd
jsr docmds
ldd th3s

* Save box4 to rdata for sounding.

* Sound box4

* Save box7 to rdata for sounding.

* Sound box7

* Save box8 to rdata for sounding.

* Sound box8

* Save box9 to rdata for sounding.

* Sound box9

61
* Search all values of hgt to find the lowest value. Subtract this value from each hgt.

* Integer divide each hgt by ZDPI/4 (1/4")
psbx
pula
pulb
std hgt1
lldd hgt2
ldx tmp16
idiv
psbx
pula
pulb
std hgt2
lldd hgt3
ldx tmp16
idiv
psbx
pula
pulb
std hgt3
lldd hgt4
ldx tmp16
idiv
psbx
pula
pulb
std hgt4
lldd hgt5
ldx tmp16
idiv
psbx
pula
pulb
std hgt5
lldd hgt6
ldx tmp16
idiv
psbx
pula
pulb
std hgt6
lldd hgt7
ldx tmp16
idiv
psbx
pula
pulb
std hgt7
lldd hgt8
ldx tmp16
idiv
psbx
pula
pulb
std hgt8
lldd hgt9
ldx tmp16
idiv
psbx
pula
pulb
std hgt9
* Decide whether to:
  * (1) Lift at a box within grid.
  * (2) Make another location the center of new search pattern.
  * (3) Give up.
idaa '#1'
staa target
* Search all values of hgt to find the greatest value of hgt. Set target = the box number of the first occurrence of the greatest value.

* Save target box to rdata for pickup.
* Convert target to hex, subtract 1, multiply by MAXIN, Add boxl address, store in X.

* Print search points
* Print map
ldaa #$'0'           ;* Open claw to prepare for pickup
staa clawcmd
jsr docmds

ldaa #$'1'            ;* Pick up object
staa clawcmd
staa datain
jsr makecmd
jsr docmds
giveup jsr gotop

puly pulx pulb pula rts

**************
* gohome -- Return to home position
**************
homept fcc ' +00,+00,+000,+07.0,+03.0'

gohome ldy #$R_DATA
idx #homept
ldaa #$'1'
staa fcmd1
staa fcmd2
staa fcmd3
staa fcmdc
ldaa #$'0'
staa clawcmd
gohome1 ldaa 0,X
staa 0,Y
* jsr putacia
inx
iny
cpy #$R_DATA+#MAXIN
bit gohome1
* ldaa #$CR
* jsr putacia
jsr checkin
jsr makecmd
jsr docmds2
* jsr doout
rts

**************
* gotop -- Take claw from current position all the way to the top.
**************
gotop psha
pshb
pshx
psby
ldaa #$'1'
staa fcmd1
staa fcmd2
staa fcmd3
staa fcmdc
ldaa #$'1'
staa clawcmd
ldx #$R_DATA+#MAXIN+#SCHAR
* Note +09.0 is out of range
* and forces the claw all the way up.
ldaa #$'0'
staa 0,X
inx
ldaa #$'0'
staa 0,X
inx
* ldaa #$'9'
ldaa #$'0'
staa 0,X
* Temporarily make position 00.0
**init** -- initial process for the program. Call this function in the
first step of the main program.

```
init ldaa #$SPACE
jsr putacia
jsr putacia
ldx #REGBAS
jsr checkin
jsr makecmd
jsr docmds2
puly pulx pulb pula rts
```

**********
* grip -- Activates or deactivates claw
**********

```
grip ldx $REGBAS
ldaa clawcmd cmpa #'1' beq clawon
clawoff bclr PORTA,X #$10000000
bra zgrip
clawon best PORTA,X #$10000000
zgrip ldaa #'1'
staa ffbkc
rts
```

**********
* inacia -- reads MAXIN characters in ACIA. Stores at R_DATA.
**********

```
inacia jsr syntax
ldaa #'0'
staa datain
ldaa #'1'
staa clawcmd ldy $R_DATA
ldab ACIA bitb #$01 beq zinacia
READIT ldaa ACIA+1 anda #$ff staa 0,Y jsr putacia
iny cpy $R_DATA+MAXIN-$S01 bls GETNEXT
ldy $R_DATA bra zinacia
GETNEXT ldab ACIA bitb #$01 beq GETNEXT bra READIT
zinacia nop ldaa #$CR
jsr putacia
rts
```

**********
* init -- initial process for the program. Call this function in the
first step of the main program.

```
init ldaa #$SPACE
jsr putacia
jsr putacia
ldx #REGBAS
```

** Without these three lines,**
** approximately 10 characters  
** of nonsense print prior to **
** the first outcmd. ????????
ldy $R\_DATA
jsr onacia
jsr onsci
jsr INITPA
jsr INIVAR

***********
* INITPA -- initializes PORT A (68HC11E9)
***********
INITPA  ldx #R\_BASE
ldaa #\#0000100
staa PACTL,X
ldaa #\#00110000
staa PORTA,X
ldaa #\#00001111
staa TFLG1,X
ldaa #\#00000111
staa TMSK1,X
bset TCTL2,X #\#10101010
rts

***********
* INIVAR -- Initializes variables
***********
INIVAR  ldy #R\_DATA
ldaa #SPACE
INIVAR1 staa 0,Y
iny
cpy #R\_DATA+#MAXIN-#$01
bts INIVAR1

* ldxy #R\_DATA
ldaa \\#'0'
staa fcmd1
staa fcmd2
staa fcmd3
staa fcmdc
staa ffbk1
staa ffbk2
staa ffbk3
staa ffbkc
staa fstop
staa clawcmd
ldd #\#0000
std th1
std th2
std th3
ldaa \\#'X'
ldy #\#SCI0\_IN
inits0 staa 0,Y
iny
cpy #\#SCI0\_IN+#MAXSCI-#$01
bts inits0
ldy #\#SCI1\_IN

inits1 staa 0,Y
iny
cpy #\#SCI1\_IN+#MAXSCI-#$01
bts inits1
ldy #\#SCI2\_IN

inits2 staa 0,Y
iny
cpy #\#SCI2\_IN+#MAXSCI-#$01
bts inits2
ldaa #0
staa hgt1
staa hgt1+1
staa hgt2
staa hgt2+1
staa hgt3

* P0:000, P1:001, P2:010, P3:011
* P6:100, P5:101, P6:110, P7:111
* reset interrupt flags IC4-IC1
* Enable IC1 interrupts
* Interrupt IC1 on falling edge
staa hgt3+1
staa hgt4
staa hgt4+1
staa hgt5
staa hgt5+1
staa hgt6
staa hgt6+1
staa hgt7
staa hgt7+1
staa hgt8
staa hgt8+1
staa hgt9
staa hgt9+1

ZINIVAR rts

*************
* iscomsp -- Writes '1' to accumulator B if character in accumulator A is
  *      ',' or ' ' else writes '0'.
*************
iscomsp ldab #$1'
cmpa #$'
beq ziscom
beq ziscom
ldab #$'

ziscom rts

*************
* isdec -- Writes '1' to accumulator B if character in accumulator A is
  *      '.' else writes '0'.
*************
isdec ldab #$1'
cmpa #$'
beq zisdec
ldab #$'

zisdec rts

*************
* isint -- Writes '1' to accumulator B if character in accumulator A is
  *      '1', '2', '3', '4', '5', '6', '7', '8', '9', '0', else
  *      writes '0'.
*************
isint ldab #$1'
cmpa #$0'
beq zisint
beq zisint
cmpa #$1'
beq zisint
cmpa #$2'
beq zisint
cmpa #$3'
beq zisint
cmpa #$4'
beq zisint
cmpa #$5'
beq zisint
cmpa #$6'
beq zisint
cmpa #$7'
beq zisint
cmpa #$8'
beq zisint
cmpa #$9'
beq zisint
ldab #$0'

zisint rts

*************
* issign -- Writes '1' to accumulator B if character in accumulator A is
  *      '+' or '-' else writes '0'.
*************
**********
issign ldab #'1'
cmpa #'+
beq zissign
cmpa #'-'
beq zissign
cmpa #'0'
beq zissign
ldab #'0'
rt$*

**********
* makecmd -- Convert input string into motor commands.
**********
makecmd ldaa datain
        cmpa #'1'
        beq begmake
        jmp zmakecmd

begmake nop

* if data in no good, quit
ldx #R_DATA
dy #pitch
ldaa 0,X
staa 0,Y
inx
iny
cpy #pitch+#PCHAR
blt getp

inx
dy #roll
ldaa 0,X
staa 0,Y
inx
iny
cpy #roll+#QCHAR
blt getq

inx
dy #theta
ldaa 0,X
staa 0,Y
inx
iny
cpy #theta+#TCHAR
blt getth

inx
dy #radius
ldaa 0,X
staa 0,Y
inx
iny
cpy #radius+#RCHAR
blt getrad

inx
dy #height
ldaa 0,X
staa 0,Y
inx
iny
cpy #height+#ZCHAR

* parse received string into
* individual strings

* store string in pitch
* store string in roll
* store string in theta
* store string in radius
* store string in height

69
<table>
<thead>
<tr>
<th>Step</th>
<th>Instruction</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td><code>blt</code></td>
<td></td>
</tr>
<tr>
<td></td>
<td><code>gethght</code></td>
<td></td>
</tr>
<tr>
<td></td>
<td><code>maketh1</code></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td><code>ldab</code></td>
<td>theta+TCHAR-$01`</td>
</tr>
<tr>
<td></td>
<td><code>jsr</code></td>
<td>ASCIIHEX</td>
</tr>
<tr>
<td>3</td>
<td><code>ldaa</code></td>
<td>$00`</td>
</tr>
<tr>
<td></td>
<td><code>std</code></td>
<td><code>th1</code></td>
</tr>
<tr>
<td>4</td>
<td><code>ldab</code></td>
<td>theta+TCHAR-$02`</td>
</tr>
<tr>
<td></td>
<td><code>jsr</code></td>
<td>ASCIIHEX</td>
</tr>
<tr>
<td>5</td>
<td><code>ldaa</code></td>
<td>$10`</td>
</tr>
<tr>
<td></td>
<td><code>mul</code></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td><code>add</code></td>
<td><code>th1</code></td>
</tr>
<tr>
<td></td>
<td><code>std</code></td>
<td><code>th1</code></td>
</tr>
<tr>
<td>7</td>
<td><code>ldab</code></td>
<td>theta+TCHAR-$03`</td>
</tr>
<tr>
<td></td>
<td><code>jsr</code></td>
<td>ASCIIHEX</td>
</tr>
<tr>
<td>8</td>
<td><code>ldaa</code></td>
<td>$100`</td>
</tr>
<tr>
<td></td>
<td><code>mul</code></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td><code>add</code></td>
<td><code>th1</code></td>
</tr>
<tr>
<td></td>
<td><code>std</code></td>
<td><code>th1</code></td>
</tr>
<tr>
<td>10</td>
<td><code>cmpb</code></td>
<td><code>thetas</code> <code>&quot;-&quot;</code></td>
</tr>
<tr>
<td></td>
<td><code>beq</code></td>
<td><code>mkth1a</code></td>
</tr>
<tr>
<td>11</td>
<td><code>ldd</code></td>
<td><code>$00000</code></td>
</tr>
<tr>
<td></td>
<td><code>subd</code></td>
<td><code>th1</code></td>
</tr>
<tr>
<td>12</td>
<td><code>std</code></td>
<td><code>th1</code></td>
</tr>
<tr>
<td>13</td>
<td><code>mkth1a</code></td>
<td><code>ldaa</code> <code>'#2'</code></td>
</tr>
<tr>
<td></td>
<td><code>staa</code></td>
<td><code>fcmd1</code></td>
</tr>
<tr>
<td>14</td>
<td><code>maketh2</code></td>
<td></td>
</tr>
<tr>
<td>15</td>
<td><code>ldab</code></td>
<td>radius+RCHAR-$01`</td>
</tr>
<tr>
<td></td>
<td><code>cmpb</code></td>
<td><code>'#5'</code></td>
</tr>
<tr>
<td>16</td>
<td><code>blt</code></td>
<td><code>mkth2a</code></td>
</tr>
<tr>
<td>17</td>
<td><code>ldd</code></td>
<td><code>$RDPI</code></td>
</tr>
<tr>
<td>18</td>
<td><code>ldx</code></td>
<td></td>
</tr>
<tr>
<td>19</td>
<td><code>ldiv</code></td>
<td></td>
</tr>
<tr>
<td>20</td>
<td><code>pshx</code></td>
<td></td>
</tr>
<tr>
<td>21</td>
<td><code>pula</code></td>
<td></td>
</tr>
<tr>
<td>22</td>
<td><code>pubb</code></td>
<td></td>
</tr>
<tr>
<td>23</td>
<td><code>bra</code></td>
<td><code>mkth2b</code></td>
</tr>
<tr>
<td>24</td>
<td><code>mkth2a</code></td>
<td><code>ldd</code> <code>$0000</code></td>
</tr>
<tr>
<td>25</td>
<td><code>std</code></td>
<td><code>th2</code></td>
</tr>
<tr>
<td>26</td>
<td><code>ldab</code></td>
<td>radius+RCHAR-$03`</td>
</tr>
<tr>
<td>27</td>
<td><code>jsr</code></td>
<td>ASCIIHEX</td>
</tr>
<tr>
<td>28</td>
<td><code>ldaa</code></td>
<td><code>$RDPI</code></td>
</tr>
<tr>
<td>29</td>
<td><code>mul</code></td>
<td></td>
</tr>
<tr>
<td>30</td>
<td><code>add</code></td>
<td><code>th2</code></td>
</tr>
<tr>
<td>31</td>
<td><code>std</code></td>
<td><code>th2</code></td>
</tr>
<tr>
<td>32</td>
<td><code>ldab</code></td>
<td>radius+RCHAR-$04`</td>
</tr>
<tr>
<td>33</td>
<td><code>jsr</code></td>
<td>ASCIIHEX</td>
</tr>
<tr>
<td>34</td>
<td><code>ldaa</code></td>
<td><code>$00</code></td>
</tr>
<tr>
<td>35</td>
<td><code>pshb</code></td>
<td></td>
</tr>
<tr>
<td>36</td>
<td><code>psha</code></td>
<td></td>
</tr>
<tr>
<td>37</td>
<td><code>puly</code></td>
<td></td>
</tr>
<tr>
<td>38</td>
<td><code>mkth2c</code></td>
<td><code>cpy</code> <code>$0000</code></td>
</tr>
<tr>
<td></td>
<td><code>beq</code></td>
<td><code>mkth2d</code></td>
</tr>
<tr>
<td>39</td>
<td><code>ldx</code></td>
<td><code>#10</code></td>
</tr>
<tr>
<td>40</td>
<td><code>mkth2f</code></td>
<td><code>ldd</code> <code>$RDPI</code></td>
</tr>
<tr>
<td></td>
<td><code>add</code></td>
<td><code>th2</code></td>
</tr>
<tr>
<td>41</td>
<td><code>std</code></td>
<td><code>th2</code></td>
</tr>
<tr>
<td>42</td>
<td><code>dax</code></td>
<td></td>
</tr>
<tr>
<td>43</td>
<td><code>bne</code></td>
<td><code>mkth2f</code></td>
</tr>
<tr>
<td>44</td>
<td><code>dey</code></td>
<td></td>
</tr>
<tr>
<td>45</td>
<td><code>bra</code></td>
<td><code>mkth2c</code></td>
</tr>
<tr>
<td>46</td>
<td><code>mkth2d</code></td>
<td><code>nop</code></td>
</tr>
<tr>
<td>47</td>
<td><code>ldab</code></td>
<td><code>radius</code></td>
</tr>
<tr>
<td></td>
<td><code>cmpb</code></td>
<td><code>'#-'</code></td>
</tr>
<tr>
<td>48</td>
<td><code>beq</code></td>
<td><code>mkth2d</code></td>
</tr>
<tr>
<td>49</td>
<td><code>ldd</code></td>
<td><code>$00000</code></td>
</tr>
<tr>
<td>50</td>
<td><code>subd</code></td>
<td><code>th2</code></td>
</tr>
<tr>
<td>51</td>
<td><code>std</code></td>
<td><code>th2</code></td>
</tr>
</tbody>
</table>

* Convert theta string to th1 (hex) * 1's place
* Absolute value ok
* If "-" leave positive
* If "+" make negative
* Since motor sign opposite of coordinate system sign

* Convert radius string to th2 (hex)
* If tenths place >= 5 then
  * use 1/2 in. else truncate.

* Degrees per one inch

* If "-" leave positive
* If "+" make negative
* Since motor sign opposite of coordinate system sign
* Add count/coordinate system offset

* Set command flag

* Convert height string to th3 (hex)
* If tenthm place >= 5 then
* use 1/2 in. else truncate.

* 1's place
* Degrees per one inch

* Degrees per one inch

* If "-" make negative
* otherwise make positive
* Since motor sign is the same as
* otherwise make positive
* Since motor sign is the same as
* coordinate system sign

* Add count/coordinate system offset

* Set command flag

**********
* outcmdl -- Sends motor 1 command out sci port.
**********
outcmdl ldaa #241
jsr putsci
ldaa th1
jsr scib2h
ldaa th1+#$01
jsr scib2h
ldaa #"0"
staa  fcmd1
rts

************
* outcmd2 -- Sends motor 2 command out sci port.
************
outcmd2 ldaa #$242
jsr  putsci
ldaa th2
jsr  sci2
ldaa th2+$01
jsr  sci2
ldaa #'0'
staa  fcmd2
rts

************
* outcmd3 -- Sends motor 3 command out sci port.
************
outcmd3 ldaa #$243
jsr  putsci
ldaa th3
jsr  sci2
ldaa th3+$01
jsr  sci2
ldaa #'0'
staa  fcmd3
rts

************
* patbig -- Define centers of overall search pattern. These values
* will be used to start
* searches if the first grid search is not successful.
* cntr2:  r = r5 + 3*del_r
*          theta2 = theta5
* cntr3:  r = r5 + 3*del_r
*          theta3 = theta5 + 3*del_theta
* cntr4:  r = r5
*          theta4 = theta5 + 3*del_theta
* cntr5:  r = r5 - 3*del_r
*          theta5 = theta5 + 3*del_theta
* cntr6:  r = r5 - 3*del_r
*          theta6 = theta5
* cntr7:  r = r5 - 3*del_r
*          theta7 = theta5 - 3*del_theta
* cntr8:  r = r5
*          theta8 = theta5 - 3*del_theta
* cntr9:  r = r5 + 3*del_r
*          theta9 = theta5 - 3*del_theta
************
patbig  psba
psb
psbx
psby

* Determine the number of whole & half inches in 3*del_r.
ldd  del_r
add  del_r
add  del_r
ldx  #$2
idiv
std  half_in
psb
pula
pulb
std  whole_in

* Modify r values of cntr3, cntr4, and cntr5.
ldaa  #$00  * Read string. Convert to hex.
ldab  cntrl+15
subb  #$30
std   tmp16
ldab  cntrl+14
subb  #$30
ldaa  #$10
mul
addd  tmp16
std   tmp16
addd  whole_in  * Add whole_in to hex version of rr string
ldx   #$10
idiv
addd  #$30
stab  cntr5+15
stab  cntr4+15
stab  cntr3+15
pslx
pulb
addd  #$30
stab  cntr5+14
stab  cntr4+14
stab  cntr3+14
ldd   half_in  * if (half_in==1) then rr.x = rr.x+.5
cpd   #1
bne   nohafs1
ldaa  cntrl+17
cmpa  #$5'
bq    hafs in1
ldaa  #$5'
staa  cntr5+17
staa  cntr4+17
staa  cntr3+17
bra   nohafs1
hafs in1 ldaa  #$0'
staa  cntr5+17
staa  cntr4+17
staa  cntr3+17
ldaa  cntrl+15
adda  #1
cmpa  #$9'
bqt   carrys1
staa  cntr5+15
staa  cntr4+15
staa  cntr3+15
bra   nohafs1
carrys1 ldaa  #$0'
staa  cntr5+15
staa  cntr4+15
staa  cntr3+15
ldaa  cntrl+14
adda  #1
staa  cntr5+14
staa  cntr4+14
staa  cntr3+14
nohafs1 nop

* Modify x values of cntr7, cntr8, and cntr9
lddd  tmp16  * Load hex version of rr string
subb  whole_in  * Subtract whole_in from hex version of rr
ldx   #$10  * Convert new hex value to rr ASCII string
idiv
addd  #$30

73
stab cntr7+$15
stab cntr8+$15
stab cntr9+$15
psbh pul a
push push addd
stab cntr7+$14
staa cntr8+$14
staa cntr9+$14

ldd half_in

if (half_in=1) then rr.r = rr.r+.5

ldaa $1
hne nobafs2
ldaa cntr1+$17
cmpa #$0
beq hafsin2
ldaa #$0
staa cntr7+$17
staa cntr8+$17
staa cntr9+$17
sub $1

hafsin2 ld$5
staa cntr7+$17
staa cntr8+$17
staa cntr9+$17
subb $1
cmpa #$0
blt carrys2
staa cntr7+$15
staa cntr8+$15
staa cntr9+$15
bra nobafs2

carrys2 ld$9
staa cntr7+$15
staa cntr8+$15
staa cntr9+$15
sub $1
staa cntr7+$14
staa cntr8+$14
staa cntr9+$14

nobafr2 nop

* Modify theta values of cntr5, cntr6, and cntr7
ldaa #$00
ldab cntr1+$11
subb #$30
std tmpl6
ldab cntr1+$10
subb #$30
ldaa $10
mul
add tmpl6
std tmpl6
ldab cntr1+$9
subb #$30
ldaa $100
mul
add tmpl6
std tmpl6
subb del_theta

* Subtract 3*del_theta from hex version of theta string. Same as adding and including sign on theta. Sign ignored since always negative.
ldx  $100  * Convert hex value to theta ASCII string
idiv
psha
pshb
pshx
pula
pulb
add $30
stab cntr5+$9
stab cntr6+$9
stab cntr7+$9
pulb
pula
ldx  $10
idiv
psha
pshb
pshx
pula
pulb
add $30
stab cntr5+$10
stab cntr6+$10
stab cntr7+$10
pulb
pula
add $30
stab cntr5+$11
stab cntr6+$11
stab cntr7+$11
* Modify theta values of cntr3, cntr2, and cntr9
ldd  tmp16  * Load hex version of rr string
  add del_theta  * Add 3*del_theta from hex version of
  add del_theta  * theta string. Same as subtracting and
  add del_theta  * including sign on theta. Sign ignored
  * since always negative.
ldx  $100  * Convert hex value to theta ASCII string
idiv
psha
pshb
pshx
pula
pulb
add $30
stab cntr3+$9
stab cntr2+$9
stab cntr9+$9
pulb
pula
ldx  $10
idiv
psha
pshb
pshx
pula
pulb
add $30
stab cntr3+$10
stab cntr2+$10
stab cntr9+$10
pulb
pula
add $30

75
stub cntr3+$11
stub cntr2+$11
stub cntr9+$11
jsr wrcntr  * Write out the calculated grid centers.
puly pulx pulb pula rts

************
* pattern  -- Modify box strings to spread pattern. Looking down from the
* boom and towards boom end, search pattern looks like
*  
*          box1   box2   box3
*  
*          box4   box5   box6
*  
*          box7   box8   box9
*  
* Top row:        r = r5 + del_r
* Bottom row:     r = r5 - del_r
* Left Column:    theta = theta5 + del_theta
* Right Column:   theta = theta5 - del_theta

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

pshx pshy

* Determine the number of whole & half inches that r must be modified
ldd del_r
ldx #1
idiv
std half_in
pshx pulb
std whole_in

* Modify r values of box1, box2, and box3.
ldaa #$00  * Read string. Convert to hex.
ldab box5+$15
subb #$30
std tmp16
ldab box5+$14
subb #$30
ldaa #10
mul
addd tmp16
std tmp16
addd whole_in  * Add whole_in to hex version of rr string
ldx #10
idiv
addd #$30
stub box1+$15
stub box2+$15
stub box3+$15
pshx pulb
pula pulb
addd #$30
stub box1+$14
stub box2+$14
stub box3+$14
ldd half_in  * if (half_in=1) then rr.r = rr.r+.5

76
cpd $1
bne nohalf1
ldaa box5+#17
cmpa '5'
beq halfin1
ldaa '5'
sta box1+#17
sta box2+#17
sta box3+#17
bra nohalf1

halfin1 ldaa #'0'
sta box1+#17
sta box2+#17
sta box3+#17
lda box5+#15
adda $1
cmpa #'9'
bgt carry1
sta box1+#15
sta box2+#15
sta box3+#15
bra nohalf1
carry1 ldaa #'0'
sta box1+#15
sta box2+#15
sta box3+#15
lda box5+#14
adda $1
sta box1+#14
sta box2+#14
sta box3+#14
nohalf1 nop

* Modify r values of box7, box8, and box9

ldd tmp16 * Load hex version of rr_string

subd whole_in * Subtract whole_in from hex version of rr

ldx $10 * Convert new hex to rr ASCII string

ldiv

add $30
stab box7+#15
stab box8+#15
stab box9+#15
pshx
pula
pulb
add $30
stab box7+#14
stab box8+#14
stab box9+#14

ldd half_in * if (half_in=1) then rr.r = rr.r+.5

cpd $1
bne nohalf2
ldaa box5+#17
cmpa #'0'
beq halfin2
lda box5+#17
lda box5+#17
sta box9+#17
bra nohalf2

halfin2 ldaa '5'
sta box7+#17
sta box8+#17
sta box9+#17
lda box5+#15
suba $1

77
cmpa  #'0'
blt  carry2
staa  box7+#15
staa  box8+#15
staa  box9+#15
bra  nohalf2

nohalf2  nop

carry2  ldaa  #'9'
staa  box7+#15
staa  box8+#15
staa  box9+#15
ldaa  box5+#14
suba  #1
staa  box7+#14
staa  box8+#14
staa  box9+#14

* Modify theta values of box1, box4, and box7

ldaa  #$00
ldab  box5+#11
subb  #$30
std  tmpl6
ldab  box5+#10
subb  #$30
ldaa  #10
mul
add  tmpl6
std  tmpl6
ldab  box5+#9
subb  #$30
ldaa  #100
mul
add tmpl6
std  tmpl6

* Subtract del_theta from hex version of theta string. Same as adding and including sign on theta. Sign ignored since always negative.

* Convert hex value to theta ASCII string
* Modify theta values of box3, box6, and box9

```assembly
ldd tmp16  * Load hex version of rr string
add del_theta  * Add del_theta from hex version of theta string. Same as subtracting and
               * including sign on theta. Sign ignored
               * since always negative

ldx $100  * Convert hex value to theta ASCII string
```

```assembly
ldx $10
idiv psha psb pshx pula pulb
add $30
stab box3+$9
stab box6+$9
stab box9+$9
pulb pula
ldx $10
idiv psha psb pshx pula pulb
add $30
stab box3+$10
stab box6+$10
stab box9+$10
pulb pula
add $30
stab box3+$11
stab box6+$11
stab box9+$11
```

```assembly
dlx #11
```

```assembly
jsr wrbox  * Print search box coordinates to screen
```

```assembly
puly pulx pulb pula rts
```

**********

* putaway -- Move claw to drop location and open.
**********

dropt fcc '00,00,000,010.5,-03.5'

putaway nop

*! new stuff to take claw to top before approaching drop point
```

```assembly
ldx #R_DATA
```

```assembly
*debl ldaa 0,X
```

```assembly
* inx * cpx #R_DATA+#MAXIN
```

```assembly
* blt debl
```

```assembly
* ldaa #1'  * Take claw to top
```

```assembly
* staa fcmd1
```

```assembly
* staa fcmd2
```

```assembly
* staa fcmd3
```

```assembly
* staa fcmdc
```

```assembly
* ldaa #1'
```

```assembly
* staa clawcmd
```

```assembly
* ldex #R_DATA+#MAXIN-ZCHAR  * Note +09.0 is out of range
```

79
* ldaa $' '  
* staa 0,X  
* inx  
* ldaa $'0'  
* staa 0,X  
* inx  
* ldaa $'9'  
* staa 0,X  
* inx  
* ldaa $'0'  
* staa 0,X  
* jsr checkin  
* jsr makecmd  
* jsr docmds2  
* 
* ! end new stuff  
ldy #$R_DATA  
ldx #$droppt  
ldaa #$'1'  
staa fcmd1  
staa fcmd2  
staa fcmd3  
staa fcmdc  
ldaa #$'0'  
staa clawcmd  
ptaway2 ldaa 0,X  
staa 0,Y  
inx  
iny  
cpy #$R_DATA+#MAXIN  
blt ptaway2  
jsr checkin  
jsr makecmd  
jsr docmds  
* jsr doout  
rts  

**********  
* PRNTDAT -- OUTPUT RELEVANT DATA TO SCREEN  
**********  
PRNTDAT psha  
pshb  
pashx  
pshy  
* ldaa #$'0'  
* jsr putacia  
ldx #$REGBAS  
ldy #$sci0_in  
prthl ldaa 0,Y  
* jsr putacia  
inw  
cpy #$sci0_in+#MAXSCI-$2  
bne prthlz  
* ldaa #$'h'  
* jsr putacia  
ldaa #$SPACE  
jsr putacia  
prthlz nop  
*p cpy #$sci0_in+#MAXSCI-$501  
* mode info removed for plot  
bls prthl  
ldaa #$SPACE  
jsr putacia  
jsr putacia
* ldaa #$'0'          * added for plot
* jsr putacia         * added for plot
ldy #$e1_in
prth2 ldaa 0,Y
jsr putacia
iny
cpy #$e1_in+$MAXSCI-#2
bne prth2z
* ldaa #$'h'          * added for plot
* jsr putacia
ldaa #$SPACE
jsr putacia
prth2z nop
*p cpy #$e1_in+$MAXSCI-$#01   * mode info removed for plot
bls prth2

ldaa #$SPACE
jsr putacia
jsr putacia

* ldaa #$'0'          * added for plot
* jsr putacia         * added for plot
ldy #$e2_in
prth3 ldaa 0,Y
jsr putacia
iny
cpy #$e2_in+$MAXSCI-#2
bne prth3z
* ldaa #$'h'          * added for plot
* jsr putacia
ldaa #$SPACE
jsr putacia
prth3z nop
*p cpy #$e2_in+$MAXSCI-$#01   * mode info removed for plot
bls prth3

prclw ldab PORTA,X
bmi itson
itsoff ldaa #$'0'
jsr putacia
* ldaa #$'0'
* jsr putacia
* ldaa #$'p'
* jsr putacia
* ldaa #$'e'
* jsr putacia
* ldaa #$'n'
* jsr putacia
* ldaa #$SPACE
* jsr putacia
* jsr putacia
bra ZPRNTDT
itson ldaa #$'1'
jsr putacia
* ldaa #$'c'
* jsr putacia
* ldaa #$'1'
* jsr putacia
* ldaa #$'o'
* jsr putacia
* ldaa #$'s'
* jsr putacia
* ldaa #$'e'
* jsr putacia
* ldaa #$'d'

81
Jmr putacia

ZPRNTDT ldaa $CR
    jsr putacia

puly
pulx
pub
pula
rts

************
* showmap -- print depth map to screen (3x3 grid)
************
showmap psaha
    psahb
    psahx
    psahy

    ldaa bgt1
    jsr bin2hex
    ldaa bgt1+l
    jsr bin2hex
    ldaa #$SPACE
    jsr putacia
    ldaa bgt2
    jsr bin2hex
    ldaa bgt2+l
    jsr bin2hex
    ldaa #$SPACE
    jsr putacia
    ldaa bgt3
    jsr bin2hex
    ldaa bgt3+l
    jsr bin2hex
    ldaa #$CR
    jsr putacia
    ldaa bgt4
    jsr bin2hex
    ldaa bgt4+l
    jsr bin2hex
    ldaa #$SPACE
    jsr putacia
    ldaa bgt5
    jsr bin2hex
    ldaa bgt5+l
    jsr bin2hex
    ldaa #$SPACE
    jsr putacia
    ldaa bgt6
    jsr bin2hex
    ldaa bgt6+l
    jsr bin2hex
    ldaa #$CR
    jsr putacia
    ldaa bgt7
    jsr bin2hex
    ldaa bgt7+l
    jsr bin2hex
    ldaa #$SPACE
    jsr putacia
    ldaa bgt8
    jsr bin2hex
    ldaa bgt8+l
    jsr bin2hex
    ldaa #$SPACE
    jsr putacia
    ldaa bgt9
    jsr bin2hex
    jsr bin2hex
idaa bgt9+1
jsr bin2hex
ldaa #CR
jsr putacia
ldaa target
jsr putacia
ldaa #CR
jsr putacia

pulx
pulb
pula
rts

**********
* syntax -- print input command syntax to screen
**********
syntax nop
ldaa '#A'
jsr putacia
ldaa '#P'
jsr putacia
ldaa '#P'
jsr putacia
ldaa '#,,' jsr putacia
ldaa '#,,' jsr putacia
ldaa '#A'
jsr putacia
ldaa '#T'
jsr putacia
ldaa '#,,' jsr putacia
ldaa '#Z'
jsr putacia
ldaa '#Z'
jsr putacia
ldaa '#Z'
jsr putacia
ldaa '#Z'
jsr putacia
ldaa '#Z'
jsr putacia
ldaa '#Z'
jsr putacia
ldaa '#Z'
jsr putacia
ldaa #CR
jsr putacia
ldaa #CR
jsr putacia
ldaa #CR
jsr putacia

83
**********
* th3a2th3s() - Converts 4 ascii bytes in th3a to 2 hex bytes in th3s.
**********
th3a2th3s nop
 psha
 ldaa th3a
 jsr HEXBIN
 ldaa th3a+1
 jsr HEXBIN
 ldaa SHFTREG+1
 sta th3s
 ldaa th3a+2
 jsr HEXBIN
 ldaa th3a+3
 jsr HEXBIN
 ldaa SHFTREG+1
 sta th3s+1
 pula
 rts

**********
* th32hghts() - Converts 2 hex bytes in th3 to 4 ascii bytes in hghts (hht.h).
**********
th32hghts nop
 rts

**********
* wrbox -- print command lines for nine search points.
**********
wrbox psha
 pshb
 pshx
 pshy
 ldy #box1
 wrbox1 ldaa 0,Y
 jsr putacia
 iny
 cpy #box1+#MAXIN
 blt wrbox1
 lda #CR
 jsr putacia

 ldy #box2
 wrbox2 ldaa 0,Y
 jsr putacia
 iny
 cpy #box2+#MAXIN
 blt wrbox2
 lda #CR
 jsr putacia

 ldy #box3
 wrbox3 ldaa 0,Y
 jsr putacia
 iny
 cpy #box3+#MAXIN
 blt wrbox3
 lda #CR
 jsr putacia

 ldy #box4
 wrbox4 ldaa 0,Y
 jsr putacia
 iny
 cpy #box4+#MAXIN

84
blt wrbox4
ldaa #CR
jsr putacia

ldy #box5
ldaa 0,Y
jsr putacia
iny
cpy #box5+#MAXIN
blt wrbox5
ldaa #CR
jsr putacia

ldy #box6
ldaa 0,Y
jsr putacia
iny
cpy #box6+#MAXIN
blt wrbox6
ldaa #CR
jsr putacia

ldy #box7
ldaa 0,Y
jsr putacia
iny
cpy #box7+#MAXIN
blt wrbox7
ldaa #CR
jsr putacia

ldy #box8
ldaa 0,Y
jsr putacia
iny
cpy #box8+#MAXIN
blt wrbox8
ldaa #CR
jsr putacia

ldy #box9
ldaa 0,Y
jsr putacia
iny
cpy #box9+#MAXIN
blt wrbox9
ldaa #CR
jsr putacia

puly
pulx
pulb
pula
rts

*******
* wrctr -- print command lines for nine search grid centers.  
*******

wrcntr psha
pshb
pshx
pshy

ldy #cntrl1
wrcntrl1 ldaa 0,Y
jsr putacia
iny
cpy #cntrl1+#MAXIN
blt wr cntr1
ldaa $CR
jsr putacion

ldy $cntr2
wr cntr2 ld aa 0,Y
jsr putacion
iny
cpy $cntr2+$MAXIN
blt wr cntr2
ldaa $CR
jsr putacion

ldy $cntr3
wr cntr3 ld aa 0,Y
jsr putacion
iny
cpy $cntr3+$MAXIN
blt wr cntr3
ldaa $CR
jsr putacion

ldy $cntr4
wr cntr4 ld aa 0,Y
jsr putacion
iny
cpy $cntr4+$MAXIN
blt wr cntr4
ldaa $CR
jsr putacion

ldy $cntr5
wr cntr5 ld aa 0,Y
jsr putacion
iny
cpy $cntr5+$MAXIN
blt wr cntr5
ldaa $CR
jsr putacion

ldy $cntr6
wr cntr6 ld aa 0,Y
jsr putacion
iny
cpy $cntr6+$MAXIN
blt wr cntr6
ldaa $CR
jsr putacion

ldy $cntr7
wr cntr7 ld aa 0,Y
jsr putacion
iny
cpy $cntr7+$MAXIN
blt wr cntr7
ldaa $CR
jsr putacion

ldy $cntr8
wr cntr8 ld aa 0,Y
jsr putacion
iny
cpy $cntr8+$MAXIN
blt wr cntr8
ldaa $CR
jsr putacion

ldy $cntr9
wr cntr9 ld aa 0,Y

86
THE INTERRUPT SERVICE ROUTINES
**********
* ACIA_ISR -- Interrupt service routine occurs when ACIA input received.
**********

ACIA_ISR
nop
jsr inacia
jsr checkin
jsr makemcm
lda #datain
cmps #'1'
bne zacia
jsr PRNTDAT

zacia rti

**********
* SCI0_ISR -- Interrupt service routine occurs when SCI0 input received.
**********

SCI0_ISR
nop
* ldaa #'A'
  * jsr putacia
  ldy #REGBAS
  bclr PORTA,X #%01110000
  * read trigger character & discard
  * jsr getsci
  * jsr putacia
getsci0
dey
beq ZSCI0
ldb SCBR,X
beq getsci0
* ldaa SCBR,X
  * jsr putacia
  cmpa #$FF0
  bne ZSCI0
  bne getsci0

GETEM0
ldy #sci0_in
jsr getsci
cmpa #$FF0
beq GETEM0
staa 0,Y
* jsr putacia
iny
cpy #sci0_in+#MAXSCI-$01
bls SCI0
* ldaa #SPACE
  * jsr putacia
  * ldaa #0'
  * jsr putacia
  * ldaa #CR
  * jsr putacia
  ldaa #'1'
staa ffbkl

ZSCI0
ldaa #$00000001
staa TPLG1,X
lda PORTA,X
ora #$01110000
staa PORTA,X
rti

**********
* SCII_ISR -- Interrupt service routine occurs when SCII input received.
**********

SCII_ISR
nop
* ldaa #'B'
  * jsr putacia
  ldx #REGBAS
  bclr PORTA,X #%01110000
  * read trigger character & discard
scri
jsr getsci

• jsr putacia
  ldy #$FFFF
getsci1 dey
  beq ZSCI1
  ldab SCSR,X
  bitb #$20
  beq getsci1
  ldab SCDR,X
  jsr putacia
  cmpa #$F0
  bne ZSCI1
* Y = #$FFFF
* Y = Y - 1
* if Y=0 then return
* if (character not received) then
  goto getsci1
* read received character
  * if (char != F0) then return
  GETEM1 ldy #sci1_in
  jsr getsci1
  cmpa #$F0
  beq GETEM1
  staa 0,Y
  jsr putacia
  iny
cpy #sci1_in #$MAXSCI-$#01
  bne sci1
  ldaa #$SPACE
  jsr putacia
  * Y = Y - 1
  * if Y=0 then return
  * if (character not received) then
    * goto getsci1
  * read received character
  * echo characters, space, 0, CR
  ldaa #$'1'
  jsr putacia
  ldaa #$CR
  jsr putacia
  ldaa #$'1'
  staa ffbk2
  ldaa #00000010
  staa TVLS1,X
  ldaa PORTA,X ##01010000
  jsr getsci1
  jsr putacia
  ldy #$FFFF
getsci2 dey
  beq ZSCI2
  ldab SCSR,X
  bitb #$20
  beq getsci2
  ldab SCDR,X
  jsr putacia
  cmpa #$F0
  bne ZSCI2
* Y = #$FFFF
* Y = Y - 1
* if Y=0 then return
* if (character not received) then
  * goto getsci2
  * read received character
  * if (char != F0) then return
  GETEM2 ldy #sci2_in
  jsr getsci2
  cmpa #$F0
  beq GETEM2
  staa 0,Y
  jsr putacia
  iny
cpy #sci2_in #$MAXSCI-$#01
***********
* SCI2_ISR -- Interrupt service routine occurs when SCI2 input received.
***********
SCI2_ISR nop
  ldaa #$'C'
  jsr putacia
  ldx #REGBas
  bcir PORTA,X ##01010000
  jsr getsci2
  jsr putacia
  ldy #$FFFF
getsci2 dey
  beq ZSCI2
  ldab SCSR,X
  bitb #$20
  beq getsci2
  ldab SCDR,X
  jsr putacia
  cmpa #$F0
  bne ZSCI2
* Y = #$FFFF
* Y = Y - 1
* if Y=0 then return
* if (character not received) then
  * goto getsci2
  * read received character
  * if (char != F0) then return
  * read and store MOVSCI characters
  * echo characters, space, 0, CR
  ldaa PORTA,X ##01110000
  staa PORTA,X
  rtl
  ** we • •
  SCI2_ISR -- Interrupt service routine occurs when SCI2 input received.

89
************

* **SLAK_ISR** -- Interrupt service routine occurs when PA3 goes low.
* This occurs whenever cable tension goes slack.

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

```
SLAK_ISR ldx $REGBAS
    bclr PORTA,X $%00010000 aslak       * Continue only if PA3 is low.
    jmp zslak

aslak bclr TMSK1,X $%00010000       * Disable ICl4 interrupt
    ldx $243       * write to SCI to stop motor 3
    jsr putsci
    ldx #'S'       * read trigger character & discard
    jsr putsci
    ldx #'T'
    jsr putsci
    ldx #'O'
    jsr putsci
    ldx #'P'
    jsr putsci

* Wait for reply. Update sci2_in.

ldx $REGBAS
    bclr PORTA,X $%01010000
    jsr getsci
    ldy #$FFFF
    dec y
    beq GETSLK3
    ldab SCMR,X
    bitb #$20
    beq getslk2
    ldx SCMR,X
    cmpa #$F0
    beq GETSLK3
    bmi getslk2

GETSLK3 ldy #sci2_in
    jsr getsci
    cmpa #$F0
    beq GETSLK3
    staa 0,Y
    iny
    cpy #sci2_in+$MAXSCI-$#01
    bls slik2

ZSLK2 ldx $%00000100
    staa TVFLG1,X
    ldx PORTA,X
    ora $%01110000
    staa PORTA,X

* above lines taken from sci2_isr
```

90
** Convert received position in sci2_in (ASCII) to desired th3 (ASCII)

* Strip th3a from sci2_in

```
ldaa sci2_in
staa th3a
ldaa sci2_in+#1
staa th3a+#1
ldaa sci2_in+#2
staa th3a+#2
ldaa sci2_in+#3
staa th3a+#3
```

* Convert th3a to th3s

```
jsr th3a2th3s
ldd th3s
std tmp16
```

* Increase height by .5" to 1" to restore tension.

```
ldd th3s
subd #$00
std tmp16
ldd tmp16
ble negth3s
ldaa #'
staa stphgt
bra th3cont
negth3s
ldaa #'-'
staa stphgt
ldd #$0000
subd tmp16
std tmp16
```

```
th3cont
nop
ldx #$ZDPI
ldd tmp16
idiv
pshb
```

```
ldd #0
addd #$ZDPI
std tmp16
dex
bne stpl
```

```
pulb
ldaa #2
mul
cpd #$ZDPI
bge anglge
angllt
nop
ldaa stphgt
cmpa #'
beq case3
bra case2
Endif
```

```
anglge
nop
ldaa stphgt
cmpa #'
beq case4
bra case1
Endif
```

```
casel
nop
ldd tmp16
addd #$ZDPI
std tmp16
ldd #$ZDPI
ldx #2
```

```
tmp16 = tmp16 + 1.5*ZDPI
```

91
```
idiv
psbx
pula
pulb
add d tmp16
std tmp16
bra tmp2

case2
nop
ldd tmp16
add d #ZDPI
std tmp16
bra tmp2

case3
nop
ldd #ZDPI
ldx $2
idiv
psbx
pula
pulb
subd tmp16
std tmp16
ldd $0
subd tmp16
std tmp16
bra tmp2

case4
nop
bra tmp2

/* Convert angle from Z coordinate system to absolute motor angle by adding angle to ZZERO if positive and subtracting angle from ZZERO if negative. */

tmp16 = tmp16 + 1.0*ZDPI

/* Add an additional inch to height so that clay can be pinned instead of scooped. */

ldd th3
add d #ZDPI
std th3

ldd #1:
staa fstop
staar cmd3

/* set stop flag */

ldd #400001000
staar TPLG1,X

/* reset interrupt flag IC4 */

zslak
nop
ldaa
staar

rti

******************************************************************************
*
* THE MAIN PROGRAM
*
******************************************************************************

START
ldz $stack
jsr init
jsr gohome

zmain
ldaa datain
cmpa #1:
beq mna
ldaa ffbk1
cmpa #1:
beq mna
ldaa ffbk2
```
cmpa $'1'
beq mna
ldaa ffbk3
cmpa $'1'
beq mna
bra zmain
mna
bra main1

main1
ldaa datain
cmpa $'1'
beq main2
jsr dout
jmp zmain

main2
nop
jsr docmds
jsr findit
jsr putaway
jsr gohome
jmp zmain
jmp zmain

* unsolicited feedback received
D. Motor Controller Program Listing
PORTA pins:
* 0: low bit in 2 bit CPU address
  * Read from DIP switch 1.
* 1: hi bit in 2 bit CPU address
  * Read from DIP switch 1.
* 2: Q input from 74LS74. Driven hi
  * by index pulse. Driven lo
  * by reset CLR (lo-hi on PB7)
* 3: Step
  * (5210)
* 4: Direction (5210)
* 5: Write Input (LS7166)
* 6: Read Input (LS7166)
* 7: Control/Data Input (LS7166)

PORTB pins:
* 7: lo-hi clears 74LS74 (CLR1)

7/18/94 -- RMB
* Added routine to seek index (home position) and reset counter.
* Reads PA2 and steps is not high.
* When hi, toggles CLR on 74LS74 (PB7).

10/18/94 -- RMB (mtr2.asm)
* Added SCI Interrupt Service Routine
to accept incoming command only if
fa or fi is first character.

10/19/94 -- RMB (mtr3.asm)
* Added small, medium, and large step option

11/15/94 -- RMB (mtr4.asm)
* Repaired home seek on power up. Routine disturbed by s/m/l option.

11/16/94 -- RMB (mtr5.asm)
* Modified input command format to a one byte address
  and four bytes representing
  the ascii version of a hexadecimal number of steps.
  During the SCI_ISR, the command is check to ensure it is valid.
  A flag is set to show that a new command has been received.
  During the main loop, this command is interpreted and sent to
  the motor. Each time a command is executed, a ASCII string is output
  consisting of 8 (master address, "f0"), six characters representing
  the hex value of the count (512 counts / 360 degrees), and four
  characters representing the issued command.

11/28/94 -- RMB (mtr6.asm)
* Make input command and output count in degrees. Appropriate
  conversions must take place in code (400 steps/360 degrees and
  360 degrees/512 counts). Position output is limited to a two
  byte hex number (represented by four ASCII characters). Maximum
  and minimum values of theta are stored as lim_min and lim_max.
  Theta_d is forced to remain within this range.

11/29/94 -- RMB (mtr7.asm)
* Controller can operate closed loop (using encoder feedback) or
  open loop (using estimated theta). On power-up, mode is closed
  loop. Mode (o or c) is echoed with theta and command. Cause
  (1: nonchanging count, 2: nonzero at startup, 3: unreasonable
  variation between actual and estimated count)

12/8/94 -- RMB (mtr8.asm)
* Pause after two address output to allow master
hardware time to select port.

12/19/94 -- RMB (mtr9.asm)
Modified sci_isr to allow stop command (address,S,**). Modified
STEPIT routine to check for new command before every step. If new
command is received, routine resets. If new command is STOP,
theta_d is set to theta. Modified estimated theta (in STEPIT) to
allow for an interrupt during motion.

COMSIZE equ $0004
R_DATA equ $0100
CR equ $0D
LF equ $0A

org R_DATA
rdata rmb 4
tdata rmb 1
byte2 rmb 1
byte1 rmb 1
byte0 rmb 1
theta_d rmb 2
dxflag rmb 1
fstop rmb 1
theta rmb 2
delta_th rmb 2
mode rmb 1
modeflag rmb 1
theta_e rmb 2
theta_o rmb 2
cnterr rmb 2
temp rmb 2
tmpl6 rmb 2
steps rmb 2
SHFTREG rmb 2
TMP1 rmb 1
STACKAREA rmb 30
STACKTOP rmb 1

org $b700
waitcnt rmb 2
ADDRESS rmb 1
lim_min rmb 2
lim_max rmb 2
MAXERR rmb 2

org $b600
jmp $d000
org $b604
jmp SCI_ISR
org waitcnt
fdb $0bff
org ADDRESS
fcb $f3
org lim_min
fdb $1ff0
org lim_max
fdb $0000
org MAXERR
fbd $0010
org $fff6
fbd $bf00
org $fffe
fbd $bf04

* EPROM begins at $d000.
org $d000

96
jmp STARTUP

INIT
ldx $REGBAS
jsr INITA
jsr INITOP
jsr ONSCI
jsr GOHOM
jsr INITVAR
cpi
rts

*******
*INITVAR -- Initialize variables
*******

INITVAR ldaa $0
sta rdata
sta rdata+1
lda #0
sta dflag
sta fstop
sta modeflag
lda #0
sta mode
lda #0
sta theta_e
sta theta_o
rts

*******
*INITA -- MAKE PINS 3 & 7 OF PORTA OUTPUTS
*******

INITA ldaa %10001000
sta FACCX,X
rts

* INITIALIZE THE OPTICAL ENCODER
* COUNTER CHIP (LS7166)

INITOP jsr RSTCNTR
jsr SETQR
jsr SETICR
rts

* RESET COUNTER TO ZERO ON STARTUP

RSTCNTR psha
pshb
pshx
pshy
ldx $REGBAS
lda %11111111
sta DDRC,X
lda %00000000
sta PORTC,X
jsr WRREG
puly
pulx
pulb
pula
rts

* TOGGLE WRITE BIT TO WRITE BYTE ON
* PORTC TO LS7166.

WRREG psha
pshb
pshx
pshy
ldx $REGBAS
lda PORTA,X
rts

97
ora $11100000
staa PORTA,X
anda $11011111
staa PORTA,X
ora $11100000
staa PORTA,X
puly pulx pulb pula rts

* SET QUADRATURE REGISTER FOR X1
* OPERATION
SETQR psha
pshb pshx pshy
ldx $REGBA
ldaa $11111111
staa DDRC,X
ldaa $11111101
staa PORTC,X
jsr WRREG
puly pulx pulb pula rts

* SET INPUT CONTROL REGISTER
* ENABLE INPUTS A & B
SETICR psha
pshb pshx pshy
ldx $REGBA
ldaa $11111111
staa DDRC,X
ldaa $01010000
staa PORTC,X
jsr WRREG
puly pulx pulb pula rts

* turn SCI on. 9600 baud.
ONSCI psha
pshb pshx pshy
ldx $REGBA
ldaa #BAUD_9600
staa BAUD,X
ldaa $00001000 * wakeup by address mark (MSB=1)
staa SCCR1,X
ldaa $00101110
staa SCCR2,X * enable SCI transmit & receive (wake-up mode)
puly pulx pulb pula rts
* STEP MOTOR BACKWARD UNTIL OPTO INDEX
* GOES HIGH. FORWARD ONE STEP. RESET
* COUNTER. RESET 74LS74.

GOHOME pha
    phsb
    phx
    phy
    lda $REGVAS
* * toggle CLR1 (PB7) on 74LS74 to set
* * Q (PA2) low
    lda PORTB,X
    anda #%01111111
    sta PORTB,X
    ora #%10000000
    sta PORTB,X
* * check for high on PA2
* * (implies that index has
* * pulsed).
GOHOME1 ldaa PORTA,X
    anda #%00000000
    bne RESETEM
* backward one step since not yet at index
    jsr MFWD1 * M0 mtr1: MREV1 mtr2: MREV1 mtr3: MFWD1
* pause before next step necessary so that motor has time to respond
    jsr SLODOWN
    bra GOHOME1

RESETEM nop
* Clear 74LS74 (set Q low) step FWD (REV) one step so that zero count is
* first step with low Q.
    jsr MREV1 * M0 mtr1: MFWD1 mtr2: MFWD1 mtr3: MREV1
    jsr SLODOWN
* reset 74LS74 (Q is lo)
    lda PORTB,X
    anda #%01111111
    sta PORTB,X
    ora #%10000000
    sta PORTB,X
    jsr RSTCNTR * reset counter to 0 (LS7166)
    jsr RD CNTR
    ldd byte1
    cpd $0001
    bgt OPEN2 * else
    cpd $FFFF
    bit OPEN2
    * endif
    bra CLOSED2

OPEN2 ldaa #'o'
    staa mode
    ldaa #'2'
    staa modeflag
    bra RESETZ * mode = 'o'

CLOSED2 ldaa #'c'
    staa mode
    ldaa #'0'
    staa modeflag
    bra RESETZ * mode = 'c'

RESETZ pulx
    pulx
    pulb
    pul
    rts

* loop counts down from waitcnt
* to kill time between step
* commands
SLODOWN psha
    pshb
    lad ad waitcnt
SLO1 subd #$0001
    beq SLO1
    pulb
    pula
    rts

* Send a char out of SCI.
OUTSCI psha
    pshb
    pshx
    pshy
    ldx #$REGBAS
OUTSCI1 ldax SCSR,X
    bits #$80
    beq OUTSCI1 * loop if not
    * ready/ still
    * xmitting.
    ldax tdata
    staa SCDR,X * send char
    pulx
    pulb
    pulx
    pulb
OUTSCIIX rts

* SET UP THE LS7166 TO READ THE
* COUNTER REGISTER ON
* THE LS7166.
RDCNTR psha
    pshb
    pshx
    pshy
    ldx #$REGBAS
    ldax #11111111
    staa DDRC,X
    ldax #$00000011
    staa PORTC,X
    jsr WRREG
    ldax #$00000000
    staa DDRC,X
    jsr RDDATA
    pulx
    pulb
    pula
    rts

* READ THE THREE BYTE COUNTER
* REGISTER ON THE LS7166
RDDATA psha
    pshb
    pshx
    pshy
    ldx #$REGBAS
    ldax PORTA,X
    anda #$01111111
    ora #$01100000
    staa PORTA,X
    anda #$00111111
    staa PORTC,X
    ldab PORTA,X
stab byte0
ora #%01100000
staa PORTA,X
anda #%00111111
staa PORTA,X
ldab PORTC,X
stab byte1
ora #%01100000
staa PORTA,X
anda #%00111111
staa PORTA,X
ldab PORTC,X
stab byte1
ora #%01100000
staa PORTA,X
puly
pulx
pulb
pula
rts

*********
* PRCNT -- Convert the six byte counter value to a two byte value (degrees)
* and store at theta:theta+1
*********
PRCNT
psha
pshb
pshx
pshy
ldx #REGBAS

ldaa byte1
jsr TOASCII
pshb
staa tdata
jsr OUTSCI
pula
staa tdata
jsr OUTSCI

ldaa byte0
jsr TOASCII
pshb
staa tdata
jsr OUTSCI
pula
staa tdata
jsr OUTSCI

PRCNTX
puly
pulx
pulb
pula
rts

*********
* PRTHETA -- Convert the six byte counter value to a two byte value (degrees)
* and store at theta:theta+1
*********
PRTHETA
psha
pshb
pshx
pshy
ldx #REGBAS
jsr DIV2DEG

ldaa theta
**PRTHETX**

puly
pulx
pulb
pula
rts

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

* PRMODE -- Print mode and modeflag

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

**PRMODE**
psha
psbb
psbx
psby

ldaa mode
staa tdata
jsr OUTSCI
ldaa modeflag
staa tdata
jsr OUTSCI

puly
pulx
pulb
pula
rts

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

* PREST -- Print estimated theta

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

**PREST**
psha
psbb
psbx
psby

ldaa theta_e
jsr TOASCII
psbb
staa tdata
jsr OUTSCI
pula
staa tdata
jsr OUTSCI

ldaa theta_e+1
jsr TOASCII
psbb
staa tdata
jsr OUTSCI
pula
staa tdata
jsr OUTSCI

puly
pulx
pulb
pula
rts

*******
* 8-bit binary in A -> 2 ascii digits in A:B
*******
TOASCII tab
rora
rora
rora
anda #$0F
adda #$30
cmpa #$39
ble TASC1
adda #$7
TASC1 andb #$0F
addb #$30
cmpb #$39
ble TASCX
addb #$7
TASCX rts

* WRITE A SPACE TO THE SCI
WRSPACE psha
pshb
ldaa #$20
staa tdata
jsr OUTSCI
pulb
pula
rts

*******
* STEPIT -- Calculate and send motor command.
*******
STEPIT NOP
* Save old accumulator values
psha
pshb
pshx
pshy
* Reset steps
ldd #$0000
std steps

* Calculate new estimate for theta using theta_e and steps (number of
* steps since last update.
STEPIT1 nop
ldd steps
cpd #$0000
blt netback
bra netfwd
netback ldd #$0000
subd steps
jsr stp2deg
subd theta_e
std theta_e
ldd #$0000
subd theta_e
std theta_e
netfwd ldd steps
jsr step2deg
add theta_e
std theta_e
bra newest
newest ldd $0000
std steps

* Determine mode of operation
jsr RDCNTR
jsr DIV2DEG

ldaa mode
cmpa '#o'
beq CONT1
ldd theta
subd theta_e
std cnterr
cpd $0000
bge MODECHK
ldd $0000
subd cnterr

MODECHK cpd MAXERR
bgt OPEN3
bra CLOSED3

OPEN3 ld aa '#o'
sta mode
ldaa '#3'
sta modeflag
bra CONT1

CLOSED3 ld aa '#c'
sta mode
ldaa '#0'
sta modeflag
bra CONT1

* Determine number and direction of steps
CONT1 ld aa fstop
cmpa '#1'
beq CONT2
ldaa '#0'
sta fstop
ldaa rdata
cmpa '#s'
bne CONT2

ldaa mode
cmpa '#o'
beq olstop
ldd theta
bra storit

olstop ldd theta_e
storit std theta_d

* Calculate delta theta
CONT2 nop

ldaa mode
cmpa '#c'
beq CLOSED
ldd theta_d
subd theta_e
std delta_th
bra MOVEON

* read counter value into byte2,byte1,byte0
* convert counter value to degrees (theta)

* if open loop mode goto CONT1
* cnterr = theta - theta_e
* cnterr = abs(cnterr)
* if cnterr>MAXERR goto OPEN3
* jump to CLOSED3

* if mode=closed
theta_d = theta
* else
theta_d = theta_e
endif

* if mode = closed then
* else
delta_th = theta_d - theta
delta_th = theta_d - theta_e
endif
CLOSED

ldd theta_d
subd theta
std delta_th

* Use delta theta to determine motor direction

MOVCON

pshb
psha
puly

* Y = delta_th

cpy #$FFFFFF
bit JGOREV
* if delta_th < -1 GOREV

cpy #$00001
bgt GOFWD
* if delta_th > 1 GOFWD

jmp STEPI/TX
* branch to STEPI/TX (Good enough!)

JGOREV

jmp GOREV

* Move motor forward if necessary

GOFWD

ldd theta
std theta_o

cpy #$00B4
ble GOFWD1
ldy #$00B4

GOFWD1

nop
pshy
pula
pulb

ldaa #$0A
mul
ldx #$09
idiv

pshx
puly

* Y = X (steps)

GOFWD2

ldaa #$'1'

CMPA fstop

BNE fwda

JMP STEPI/T1

FWDa

ldd steps

ADDD #$01

STD steps

JSR MFWD1

* Take one step forward

JSR SLODOWN

* Pause between steps

dey

cpy #$0000

BNE GOFWD2

* If Y <> 0 jump to GOFWD2

ldaa mode

CMPA #$'0'

BEQ CONT3

LD THETA_O

JSR RDCNTR

JSR DIV2DEG

* read counter value intobyte2,byte1,byte0

* convert counter value to degrees (theta)

CPD THETA

BEQ FOPEN1

JMP STEPI/T1

FOPEN1

LDAA #$'0'

STAAMode

LDAA #$'1'

STAAModeflag

CONT3

JMP STEPI/T1

* Move motor backward if necessary

GOREV

ldd theta

STD theta_o

ldd #$0000

subd delta_th

PSHB

PSHA

PULY

* Y = D (Y = -delta_th)

cpy #$00B4

105
ble GOREV1
ldy #$00B4

GOREV1
nop
pshy
pula
pulb
ldaa #$0A
mul
ldx #$09
ldiv
pshx
puly

* B = LSB of Y (MSB = $00)
* A = $0A (10)
* D = B*$0A
* X = $09 (9)
* X = LSB*10/9
* Y = X (steps)

GOREV2
ldaa #'1'
cmpa fstop
bne reva
jmp STEPITI

rea
ldd steps
subd #$01
std steps
jsr MREV1
jsr SLODOWN
dey
cpy #$0000
bne GOREV2

* Take one step backward
* Pause between steps
* Y = Y - 1
* If Y <> 0 jump to GOREV2

ldaa mode
cmpa #'o'
beq CONT4

ldd theta_o
cmp theta
beq ROPEN1
jmp STEPITI

* If open loop mode goto CONT4

ROPEN1
ldaa #'o'
staa mode
ldaa #'1'
staa modeflag
jmp STEPITI

* Jump to STEPITI

CONT4

* Jump to STEPITI

* * Recall old accumulator values

STEPIX
puly
pulx
pulb
pula
rts

* Issue command for one step backward

MREV1
psha
pshb
pshx
pshy
ldx #REGBAS
ldab PORTA,X
andb #$11100111
stab PORTA,X
orab #$00001000
stab PORTA,X
puly
pulx
pulb
pula
rts

* Issue command for one step forward

MFWD1
psha
pshb
psbx
psby
ldx @REGBAS
ldab PORTA,X
orab #%00100000
andb #%11110111
stab PORTA,X
orab #%000010000
stab PORTA,X
puly
pulx
pulb
pula
rts

***************
* HEXBIN(a) - Convert the ASCII character in a
* to binary and shift into shftreg. Returns value
* in TMPI incremented if a is not hex.
***************
HEXBIN PSHA
PSHB
PSHX
psby
JSR UPCASE
CMPA #$'0'
BLT HEXNOT
CMPA #$'9'
BLE HEXXMB
CMPA #$'A'
BLT HEXNOT
CMPA #$'F'
BGT HEXNOT
ADDA #$9
HEXXMB ANDA #$0F
STAA I,X
BRA HT_XRTS
INC TMPI
HZXRTS puly
PULX
PULB
PULA
rts

***************
* UPCASE(a) - If the contents of a is alpha,
* returns a converted to uppercase.
***************
UPCASE
CMPA #$'a'
BLT CASE1
CMPA #$'z'
BGT CASE1
SUBA #$20
CASE1 RTS

***************
* DIV2DEG -- Converts Count (in divisions) to theta (in degrees)
***************
DIV2DEG psha
psbh
psbx
psby
* Push Y onto stack
107
ldaa  TMP1
psba

ldaa  #$00
staa  TMP1  * TMP1 = 0
ldy  #$0000
ldd  byte1
cpd  #$0000
bge  NOCHNG  * If byte >= 0, jump to NOCHNG
ldd  #$0000
subd  byte1  * D = -byte1:byte0
psba
ldaa  #$1
staa  TMP1  * TMP1 = 1
pula

NOCHNG  nop

* Determine how many times D is divisible by $100.  $100 div = $B4 deg.

<table>
<thead>
<tr>
<th>CNT1</th>
<th>cpd</th>
<th>$0100</th>
</tr>
</thead>
<tbody>
<tr>
<td>bne</td>
<td>CNT2</td>
<td></td>
</tr>
<tr>
<td>subd</td>
<td>$0100</td>
<td></td>
</tr>
<tr>
<td>iny</td>
<td></td>
<td></td>
</tr>
<tr>
<td>bra</td>
<td>CNT1</td>
<td></td>
</tr>
</tbody>
</table>

| CNT2 | ld a  | #$2D |
| mul  |       |       |
| ldx  | #$0040|       |
| idiv |       |       |
| ps x |       |       |
| pul a|       |       |
| pul b|       |       |
| std  | theta |       |
| psh y|       |       |
| pul a|       |       |
| pul b|       |       |
| ldaa | #$B4  |       |
| mul  |       |       |
| addd | theta |       |
| std  | theta |       |
| cmpa | #1    |       |
| bne  | NOCHNG2 |      |
| ldd  | #$0000|       |
| subd | theta |       |
| std  | theta |       |

NOCHNG2  nop

| pula  |
| staa  | TMP1 |
| puly  |
| pulx  |
| pulb  |
| pula  |
| rts   |

***************
* stp2deg  -- Converts steps in D (400/rev) to degrees (360/rev) in D.
***************

stp2deg  pshx
 pshy

<table>
<thead>
<tr>
<th>std</th>
<th>tmp16</th>
</tr>
</thead>
<tbody>
<tr>
<td>s2d1</td>
<td>ldd</td>
</tr>
<tr>
<td>ldy</td>
<td>#9</td>
</tr>
<tr>
<td>mulit</td>
<td>addd</td>
</tr>
<tr>
<td>dey</td>
<td></td>
</tr>
<tr>
<td>bne</td>
<td>mulit</td>
</tr>
<tr>
<td>std</td>
<td>tmp16</td>
</tr>
</tbody>
</table>
****SCI Interrupt Service Routine****
* SCI_ISR Idx #REGBASE
ldaa SCSR,X
ldaa SCDR,X
cmpa #$fa
beq isr2
cmpa ADDRESS
beq isr2
jmp isr4

isr2
ldy #$rdata
brclr SCSR,X #%00100000 isr3
staa 0,Y
iny
cpy #$COMSIZE+#R_DATA
blo isr3

* if $rdata:0000 = 'S****' then
* fstop = '1'
* rti
* end
ldaa rdata
cmpa #$'S'
bne nostop
ldaa #$'1'
staa fstop
jmp isr4

nostop
nop

* if all $rdata byte 0-9 or A-F then
* set flag
* else
* $rdata = 0000
* end
* return
ldaa #000
staa TMP1
ldy #$rdata
ldaa 0,Y
jsr HEXBIN
ldaa TMP1
bnz isr4
ldaa 1,Y
jsr HEXBIN
ldaa SHFTREG+1
staa theta_d
ldaa 2,Y
jsr HEXBIN
ldaa TMP1
bnz isr4

**********
* SCI Interrupt Service Routine
* If first byte is fa or ADDRESS,
* dxflag is set to $31 and
* remaining bytes are stored in rdata
**********

ldx $10
idiv
stx tmp16
push
pula
pulb
std tmp16
puly
pulx
rts

*s2dz
* Pull Y off stack
*pulx
* Pull X off stack

* if first byte is fa or ADDRESS,
* dxflag is set to $31 and
* remaining bytes are stored in rdata

* SCI_ISR Idx #REGBASE
ldaa SCSR,X
ldaa SCDR,X
cmpa #$fa
beq isr2
cmpa ADDRESS
beq isr2
jmp isr4

isr2
ldy #$rdata
brclr SCSR,X #%00100000 isr3
staa 0,Y
iny
cpy #$COMSIZE+#R_DATA
blo isr3

* if $rdata:0000 = 'S****' then
* fstop = '1'
* rti
* end
ldaa rdata
cmpa #$'S'
bne nostop
ldaa #$'1'
staa fstop
jmp isr4

nostop
nop

* if all $rdata byte 0-9 or A-F then
* set flag
* else
* $rdata = 0000
* end
* return
ldaa #000
staa TMP1
ldy #$rdata
ldaa 0,Y
jsr HEXBIN
ldaa TMP1
bnz isr4
ldaa 1,Y
jsr HEXBIN
ldaa SHFTREG+1
staa theta_d
ldaa 2,Y
jsr HEXBIN
ldaa TMP1
bnz isr4

**********
* SCI Interrupt Service Routine
* If first byte is fa or ADDRESS,
* dxflag is set to $31 and
* remaining bytes are stored in rdata
**********

ldx $10
idiv
stx tmp16
push
pula
pulb
std tmp16
puly
pulx
rts

*s2dz
* Pull Y off stack
*pulx
* Pull X off stack

* if first byte is fa or ADDRESS,
* dxflag is set to $31 and
* remaining bytes are stored in rdata

* SCI_ISR Idx #REGBASE
ldaa SCSR,X
ldaa SCDR,X
cmpa #$fa
beq isr2
cmpa ADDRESS
beq isr2
jmp isr4

isr2
ldy #$rdata
brclr SCSR,X #%00100000 isr3
staa 0,Y
iny
cpy #$COMSIZE+#R_DATA
blo isr3

* if $rdata:0000 = 'S****' then
* fstop = '1'
* rti
* end
ldaa rdata
cmpa #$'S'
bne nostop
ldaa #$'1'
staa fstop
jmp isr4

nostop
nop

* if all $rdata byte 0-9 or A-F then
* set flag
* else
* $rdata = 0000
* end
* return
ldaa #000
staa TMP1
ldy #$rdata
ldaa 0,Y
jsr HEXBIN
ldaa TMP1
bnz isr4
ldaa 1,Y
jsr HEXBIN
ldaa SHFTREG+1
staa theta_d
ldaa 2,Y
jsr HEXBIN
ldaa TMP1
bnz isr4
lda $3,Y
jsr HEBSM
lda SHAFTREG+1
staa theta_d+1
lda $31
staa dxflag

* set good data received flag

lda theta_d
cpd lim_min
bge LIM1
lda lim_min
std theta_d

* if theta_d < lim_min then
* theta_d = lim_min
* endif

LIM1 cpd lim_max
ble LIM2
lda lim_max
std theta_d

* if theta_d > lim_max then
* theta_d = lim_max
* endif

LIM2 nop

isr4 nop
beet SCCR2.X #%00000010
* RWU == 1 (MCU in wakeup mode)
zs_isr rti
* return from interrupt

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

* STARTUP -- MAIN

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

STARTUP lds #STACKTOP
jsr INIT

lda $CR
staa tdata
jsr OUTSCI

LOOP0 jsr RDCNTR
lda #$f0
staa tdata
jsr OUTSCI
lda #$f0
staa tdata
jsr OUTSCI

ldy #$ff
* pause to allow master to select port

pausel day
bne pausel

lda mode
cmpa '#c'
beq CCLLOP
jsr PREST
bra GOLOOP

CCLLOP jsr PRTTHETA
GOLOOP jsr PRMODE

* Does not affect master. Looks better on screen.

* Write command to SCI.

* Check for received command

LOOP1 lda dxflag
cmpa $31
bne LOOP1
jsr STEPIT
lda $50
staa dxflag

ZLOOP0 bra LOOP0
E. Using Pcbug11 to Program the Motorola 68HC11E9
S68HC11EVBU Customer:

Both the standard and student EVBUs come with an MC68HC11E9FN1 MCU installed on the board and the BUFFALO monitor program stored in the MCU-internal ROM. But the student EVBU kit also contains a blank MC68HC711E9 MCU and PCbug11, a PC-based monitor program. A complete description of the PCbug11 is provided in the PCbug11 User's Manual, M68PCBUG11/D1. While a detailed description of the BUFFALO monitor is available in the M68HC11EVBU Universal Evaluation Board User's Manual, M68HC11EVBU/AD1.

You may install the blank MC68HC711E9 MCU in the EVBU socket at location U3. After installing the 711E9 MCU on the EVBU it may be programmed using PCbug11. Either BUFFALO or a user-developed program can be stored in the 711E9 MCU-internal EPROM. Step-by-step EPROM programming instructions are provided in this letter or refer to page 4-12 of the PCbug11 user's manual for additional information.

WHAT IS PCbug11?

PCbug11 is a software package for easy access to and simple experimentation with M68HC11 microcontroller unit (MCU) devices. PCbug11 lets you program any member of the M68HC11 MCU family and examine the behavior of internal peripherals under specific conditions. In addition, you may run your own programs on the MCU; breakpoint processing and trace processing are available.

To configure the EVBU to use PCbug11:

1. Remove the jumper from J7, and place it across J3. Moving the jumper to J3 grounds the MODB pin and at reset places the HC11 in BOOTSTRAP mode.

NOTE

Refer to Figure 2-1 of the M68HC11EVBU Universal Evaluation Board User's Manual, M68HC11EVBU/AD1, for jumper header and connector locations.

2. Connect the EVBU to your PC serial port via a user-supplied 25-pin cable. The PC serial port can be either COM1 or COM2. The cable must be a Hayes-compatible modem cable and is available at most electronic supply stores.

3. Apply power to the EVBU.
4. To start PCbug11 from the command line:

```
PCbug11 -E port=1<CR>   Where 1/0 is COM2, use port=2.
<CR> is the symbol for carriage return.
```

5. The registers should be displayed on the screen, and a >> prompt in the window at the bottom of the screen.

6. With PCbug11 version 3.24A, enter on the PC keyboard:

```
CONTROL BASE HEX<CR>
```

This defines the keyboard input default as hexadecimal. By doing this, you do not have to add the $ to inputs.

This should get you started with PCbug11. Because the TALKER code used in this example resides in RAM, you are limited to the amount of free space that you can use for variables. It may be useful to put the TALKER into EPROM (it takes about 200 bytes) and leave most of your user space free. For more detail on the TALKER refer to paragraph 4.4 of the PCbug11 manual.
PROGRAMMING EEPROM

Files to be programmed into the 711E9 MCU-intenal EEPROM must be in S-record format. The S-record format is explained in Appendix A of the M68HC11EVBU Universal Evaluation Board User's Manual, M68HC11EVBU/AD1.

NOTE

The S-record to be downloaded into the 711E9 MCU-intenal EEPROM must be ORGed at address $B600.

Enter on the PC keyboard:

EEPROM $B600 $B7FF<CR>  
This lets PCbug11 know that these addresses are EEPROM and that it should use a EEPROM algorithm.

MS $1035 00<CR>  
This clears the block protect register (B PROT) and lets you program the EEPROM section of the 711E9 MCU.

LOADS filename<CR>  
This loads an S-record format file into the EEPROM section of the 711E9 MCU.

VERF filename<CR>  
This verifies that the S-record format file was successfully loaded into EEPROM.
PROGRAMMING EPROM

Files to be programmed into the 711E9 MCU-internal EPROM must be in S-record format. The S-record format is explained in Appendix A of the M68HC11EVBU Universal Evaluation Board User's Manual, M68HC11EVBU/AD1.

NOTE

The S-record to be downloaded into the 711E9 MCU-internal EPROM must be ORGed at address $D000.

To program the MCU-internal EPROM enter on the PC keyboard:

EPROM $D000 $FFFF<CR> This lets PCbug11 know that these addresses are EPROM and that it should use a EPROM algorithm.

Apply +12Vdc to the XIRQ pin To program the MCU-internal EPROM, +12Vdc must be applied to the XIRQ pin of the 711E9 MCU. Attach a +12Vdc power supply to the MCU I/O port connector P4, pin-18. A 100Ω resister must be installed in series with the +12Vdc power supply and P4, pin-18.

CAUTION

Do not apply a +12Vdc programming voltage power source when the main VDD (+5Vdc) power is off; doing so will damage the EVBU integrated circuits. Always turn on the main VDD (+5Vdc) power before the +12Vdc programming voltage is applied.

LOADS filename<CR> This loads an S-record format file into the EPROM section of the 711E9 MCU.

VERF filename<CR> This verifies that the S-record format file was successfully loaded into EPROM.

---

MOTOROLA

1-4

UNIVERSAL EVALUATION BOARD CUSTOMER LETTER

S68HC11EVBU/L1
**PCbug11 HINTS**

- The EPROM and EEPROM commands must be entered before you can program EPROM and EEPROM. This sets up PCbug11 EPROM and EEPROM programming routines.

- Don't forget to clear the BPROT register before trying to modify EEPROM locations.

- Initially, you should work on your routines in EEPROM. Since you can trace through EEPROM like RAM, it is best to try them out there before committing to EPROM. When tracing the EEPROM use the memory set (MS) command to modify the block protect register to 00 (MS $1035 00) and the EEPROM command (EEPROM $B600 $B7FF).

- Be sure to set your stack pointer where it will not interfere with the PCbug11 stack pointer. The TALKER program starts at $0000 in RAM, with the first free byte at $0100. The PCbug11 stack pointer is set to $01FF. Set your stack pointer at least 20 bytes ($01EB) lower than this.

- If a COM fault occurs while entering commands on the command line:

  1. Check the cable between the EVBU and the PC. If the connection is okay, try issuing the control timeout command (CONTROL TIMEOUT 10000). This gives the MCU more time to respond (needed when PCbug11 is running on a fast PC).

  2. Make sure the transmit pin for the PC connects to the receive pin of the MCU. The transmit pin will have approximately 9 to 12 volts on it. The receive pin will only have a few millivolts if any.

  3. Remove the MCU from the socket and check the pins for damage. If the pins are shorted, straighten the pins and carefully reinsert the MCU.

- If you are using an XT-PC and the display locks up, try issuing a MODE CO80 at the DOS prompt.

---