CC E & TECHNOLOGY:

AN ARCHITECTURE FOR A ROBOT HIERARCHICAL CONTROL SYSTEM

NBS Special Publication 500-23
U.S. DEPARTMENT OF COMMERCE
National Bureau of Standards

NATIONAL BUREAU OF STANDARDS

The National Bureau of Standards[1] was established by an act of Congress March 3, 1901. The Bureau's overall goal is to strengthen and advance the Nation's science and technology and facilitate their effective application for public benefit. To this end, the Bureau conducts research and provides: (1) a basis for the Nation's physical measurement system, (2) scientific and technological services for industry and government, (3) a technical basis for equity in trade, and (4) technical services to promote public safety. The Bureau consists of the Institute for Basic Standards, the Institute for Materials Research, the Institute for Applied Technology, the Institute for Computer Sciences and Technology, the Office for Information Programs, and the Office of Experimental Technology Incentives Program.

THE INSTITUTE FOR BASIC STANDARDS provides the central basis within the United States of a complete and consistent system of physical measurement; coordinates that system with measurement systems of other nations; and furnishes essential services leading to accurate and uniform physical measurements throughout the Nation's scientific community, industry, and commerce. The Institute consists of the Office of Measurement Services, and the following center and divisions:

Applied Mathematics — Electricity — Mechanics — Heat — Optical Physics — Center for Radiation Research — Laboratory Astrophysics[2] — Cryogenics[2] — Electromagnetics[2] — Time and Frequency[2].

THE INSTITUTE FOR MATERIALS RESEARCH conducts materials research leading to improved methods of measurement, standards, and data on the properties of well-characterized materials needed by industry, commerce, educational institutions, and Government; provides advisory and research services to other Government agencies; and develops, produces, and distributes standard reference materials. The Institute consists of the Office of Standard Reference Materials, the Office of Air and Water Measurement, and the following divisions:

Analytical Chemistry — Polymers — Metallurgy — Inorganic Materials — Reactor Radiation — Physical Chemistry.

THE INSTITUTE FOR APPLIED TECHNOLOGY provides technical services developing and promoting the use of available technology; cooperates with public and private organizations in developing technological standards, codes, and test methods; and provides technical advice services, and information to Government agencies and the public. The Institute consists of the following divisions and centers:

Standards Application and Analysis — Electronic Technology — Center for Consumer Product Technology: Product Systems Analysis; Product Engineering — Center for Building Technology: Structures, Materials, and Safety; Building Environment; Technical Evaluation and Application — Center for Fire Research: Fire Science; Fire Safety Engineering.

THE INSTITUTE FOR COMPUTER SCIENCES AND TECHNOLOGY conducts research and provides technical services designed to aid Government agencies in improving cost effectiveness in the conduct of their programs through the selection, acquisition, and effective utilization of automatic data processing equipment; and serves as the principal focus wthin the executive branch for the development of Federal standards for automatic data processing equipment, techniques, and computer languages. The Institute consist of the following divisions:

Computer Services — Systems and Software — Computer Systems Engineering — Information Technology.

THE OFFICE OF EXPERIMENTAL TECHNOLOGY INCENTIVES PROGRAM seeks to affect public policy and process to facilitate technological change in the private sector by examining and experimenting with Government policies and practices in order to identify and remove Government-related barriers and to correct inherent market imperfections that impede the innovation process.

THE OFFICE FOR INFORMATION PROGRAMS promotes optimum dissemination and accessibility of scientific information generated within NBS; promotes the development of the National Standard Reference Data System and a system of information analysis centers dealing with the broader aspects of the National Measurement System; provides appropriate services to ensure that the NBS staff has optimum accessibility to the scientific information of the world. The Office consists of the following organizational units:

Office of Standard Reference Data — Office of Information Activities — Office of Technical Publications — Library — Office of International Standards — Office of International Relations.

---

[1] Headquarters and Laboratories at Gaithersburg, Maryland, unless otherwise noted; mailing address Washington, D.C. 20234.
[2] Located at Boulder, Colorado 80302.

COMPUTER SCIENCE & TECHNOLOGY:

An Architecture for a Robot Hierarchical Control System

Anthony J. Barbera

Institute for Computer Sciences and Technology
National Bureau of Standards
Washington, D.C. 20234

U.S. DEPARTMENT OF COMMERCE, Juanita M. Kreps, Secretary

Dr. Sidney Harman, Under Secretary
Jordan J. Baruch, Assistant Secretary for Science and Technology

NATIONAL BUREAU OF STANDARDS, Ernest Ambler, Acting Director

Issued December 1977

Reports on Computer Science and Technology

The National Bureau of Standards has a special responsibility within the Federal Government for computer science and technology activities. The programs of the NBS Institute for Computer Sciences and Technology are designed to provide ADP standards, guidelines, and technical advisory services to improve the effectiveness of computer utilization in the Federal sector, and to perform appropriate research and development efforts as foundation for such activities and programs. This publication series will report these NBS efforts to the Federal computer community as well as to interested specialists in the academic and private sectors. Those wishing to receive notices of publications in this series should complete and return the form at the end of this publication.

National Bureau of Standards Special Publication 500-23
Nat. Bur. Stand. (U.S.), Spec. Publ. 500-23, 227 pages (Dec. 1977)
CODEN: XNBSAV

Library of Congress Cataloging in Publication Data
Barbera, Anthony J.
An architecture for a robot hierarchical control system.
(Computer science & technology) (NBS special publication ; 500-23)
Supt. of Docs. no.: C13.10:500-23
1. Robots, Industrial. 1. Title. II. Series. III. Series: United States. National Bureau of Standards. Special publication ; 500-23.
QC100.U57 no. 500-23 [T59.4] 602'.1s 629.8'92 77-17960

U.S. GOVERNMENT PRINTING OFFICE
WASHINGTON: D.C.

For sale by the Superintendent of Documents, U.S. Government Printing Office, Washington, D.C. 20402
Price $4.25 - Stock No. 003-003-01874-1

TABLE OF CONTENTS

FIGURES

TABLES

PROGRAM DOCUMENTATION

Control Hierarchy
(Module #1)

Program Module
(Module #2)

Location Module
(Module #3)

ACKNOWLEDGEMENTS

I wish to express my thanks to Drs. James Albus and John Evans for their invaluable help, advice and suggestions. Many of the ideas presented in this report were a result of many hours of discussion with them.

I wish to thank Ellen Lowenfeld for her able assistance in developing the computer programs.

I am indebted to Debbie Ingram for the uncountable hours she spent in the typing and preparation of this report.

I. CONTROL SYSTEM ARCHITECTURE

I. CONTROL SYSTEM ARCHITECTURE

AN ARCHITECTURE FOR A ROBOT HIERARCHICAL CONTROL SYSTEM

Anthony J. Barbera

ABSTRACT

Complex automation systems, such as industrial robots, require a computer-based control system for the effective utilization of this advanced technology. This report describes such a control system developed at the National Bureau of Standards. The approach has been to partition the control system into a hierarchy of different functional levels. This has proven to be a powerful technique in obtaining sensor-controlled robot behavior at a minimum cost of programming time and computer size. Further, this partitioning has greatly simplified the implementation of additional functions and sensors. This report discusses the control system, its implementation and use, and provides a documented listing of all of the control programs.

Key Words: Adaptive; automation; computer; control; goal-oriented; hierarchical control; robot; sensors.

I. CONTROL SYSTEM ARCHITECTURE

1. Introduction

The National Bureau of Standards program in automation focuses NBS resources on developing a basic understanding of the technology of computer based automation and then develops those standards and guidelines that will stimulate the diffusion of this technology to enhance productivity in both Government and industry.

Specifically this program attempts:

1) to provide standards for the interfaces between modular components of computer-aided manufacturing systems,

2) to provide standards for the computer control languages used to program automation systems,

3) to provide performance measures for specification and procurement of robots and numerically controlled machine tools, and

4) to carry out research in dynamic measurement and computer control for computer based automation systems.

It is work in this last area, the development of dynamic sensors and computer control techniques that has led to the control system architecture for robots described in this report.

1.1 Industrial Robot Control Systems

Industrial robots are proving themselves to be flexible, general purpose automation systems that will contribute significantly to the development of automatic factories. These industrial robots utilize essentially the same control mechanisms that were developed for numerically controlled (NC) machine tools. That is, the motions of an industrial robot are determined by numbers that are stored in the memory of a computer or on a tape or some other storage device. Each degree of freedom (each axis) of the robot has a position servo system that drives the joints to the values commanded by the stored numbers of the control program.

Robots have presently found use in a number of industries in such diverse operations as removing parts from presses, spray painting, spot welding automobile bodies and loading and unloading parts from machine tools. All of the applications of these robots have two characteristics in common. First, these operations involve a rather high volume production where changes to the robots' programs do not have to be made often. At present, industrial robots are not being used in a batch type of environment (i.e. where parts are made in small lots or batches and control programs must be quickly changed) because of the time and difficulty required to program the robot to do a new task. Second, these operations are also characterized by a highly constrained work environment, either due to the nature of the work (e.g. unloading work pieces from a press) or by the installation or redesign of positioning equipment (e.g. installing very accurate indexing transfer lines to maintain repeatability in the positioning of automobile bodies on a spot welding assembly line). Although present industrial robots have a high degree of positional repeatability, they cannot in any way sense slight misalignments of parts in their environment. This lack of sensory feedback creates the requirement that the position of the objects they are to work with must be accurately maintained to match the position locations stored in the robot's program.

1.2 PROBLEM

Thus the situation can be summarized as follows:

> There exists a sophisticated piece of general purpose manufacturing hardware - the industrial robot - whose effective and widespread use is seriously hampered by the difficulty and amount of time necessary to communicate even simple operations to it, and by the requirement of a tightly constrained work environment.

1.3 SOLUTION

The solution to this problem is the development of a higher level control system that will make it faster and easier to program the robot and that can interact with sensory data to modify the robot's motions in real time to cope with misalignments in its environment.

1.4 ADDITIONAL CONTROL SYSTEM REQUIREMENTS

Several additional requirements are identified here that are felt to be important for promoting the effective use of computer controlled robots within the environment of developing CAD/CAM systems (Computer-Aided Design/Computer-Aided Manufacturing).

1) the control system should be as general purpose as possible so that each type of task does not require a different unique control system (e.g. the control system that causes the robot to load parts in and out of a vice on a machine tool, should also allow a robot to spot weld automobile bodies, spray paint bath tubs, or drill holes in an aircraft wing panel).

2) The control system should be modular and partitioned to make higher control levels independent from specific robot designs. This allows the same higher levels of the control system to be universally used to direct any robot. The system should be designed to facilitate the addition of new modules at any level. This will enhance flexibility in new applications.

3) The control system should allow for the creation of robot application programs off-line. These programs should be independent of an individual robot in much the same way that the part program created by an APT programmer is independent of a particular machine tool. Obviously, a robot or machine tool with three degrees of freedom cannot execute a program requiring six degrees of freedom; however, to the extent that robots or machine tools are functionally equivalent, the same application programs should be useable.

4) The control system should allow for the entry of location points in the robot's work space from other data bases in a computer-aided manufacturing system, such as the CLDATA file from APT. In addition, these location points should be defined in a relative manner so that the same set of location points can be post-processed to specify these locations for a particular robot. This is a specific facet of the general requirement that the control system should be as compatible as possible with the Integrated Computer Aided Manufacturing concept, i.e. the use of standard interfaces between modules of well-defined functions that can interact with common data bases.

1.5 Research Control Systems

A number of groups have developed control systems (1, 2, 3, 4, 5) to solve the basic problem stated above. These higher level control systems rely on a computer to provide the information processing necessary for real time control of the robot in accord with its incoming sensory feedback data. These systems have some form of a higher level language, usually resembling a general purpose computer language like FORTRAN. This provides a communication interface to the robot so that the programmer can specify, off-line, a complex task which may require the use of sensory feedback. However, none of these control systems meets all of the additional requirements stated above.

2. THE NBS ROBOT CONTROL SYSTEM

The National Bureau of Standards (NBS), with its unique view of standards, measurement science, numerical control machining, and computer applications, together with input from manufacturing industries, has developed an architecture for a hierarchical control (6, 7) system. In carrying out this work, NBS has adopted the strategy of modular design, well-defined interfaces, and integratability into total CAD/CAM systems. The architecture of this control system, its present implementation, and future development are the subject of this report.

2.1 The Three Modules of the NBS Control System

The hierarchical control system for the execution of complex tasks involving sensory feedback makes up module #1 of the complete NBS control system shown in Figure 10.

Input to the highest level in the hierarchy of module #1 comes from module #2. In module #2 a sequence of elemental moves ("GOTO" statements) is produced. These commands are interpreted by module #1 as an executable program. This program is a procedural description of a task, and as such can be used to instruct any module #1 control hierarchy to control a robot to carry out this specified task

Module #3, the location module, is used to record the coordinate values for all the location names used in the program module. These are X, Y, Z values in a standard coordinate system. This location table is transformed by a postprocessor in module #1 to a table of coordinate values that define these points in a particular robot's work space.

2.1.1 Hierarchical Control Module

The control system architecture (Figure 1) is based on a hierarchical structure, where simple primitive operations are executed at the lowest levels. A complex task command enters the hierarchy at the highest level and results in the generation of a sequence of simple primitive commands to the next lower level. Each primitive, in turn, produces a sequence of a joint position commands to the lowest level to accomplish the task. The partitioning of the control system into this hierarchy of control levels does much to simplify the control problem and to allow complex-looking behavior to be commanded simply and quickly using a small computing system. Each control level uses incoming data (both higher level commands as well as sensory feedback) to branch to the appropriate subroutine calls of the next lower control level. Additional primitive operations or the implementation of another type of sensor require only the addition of a new subroutine that will accomplish this function.

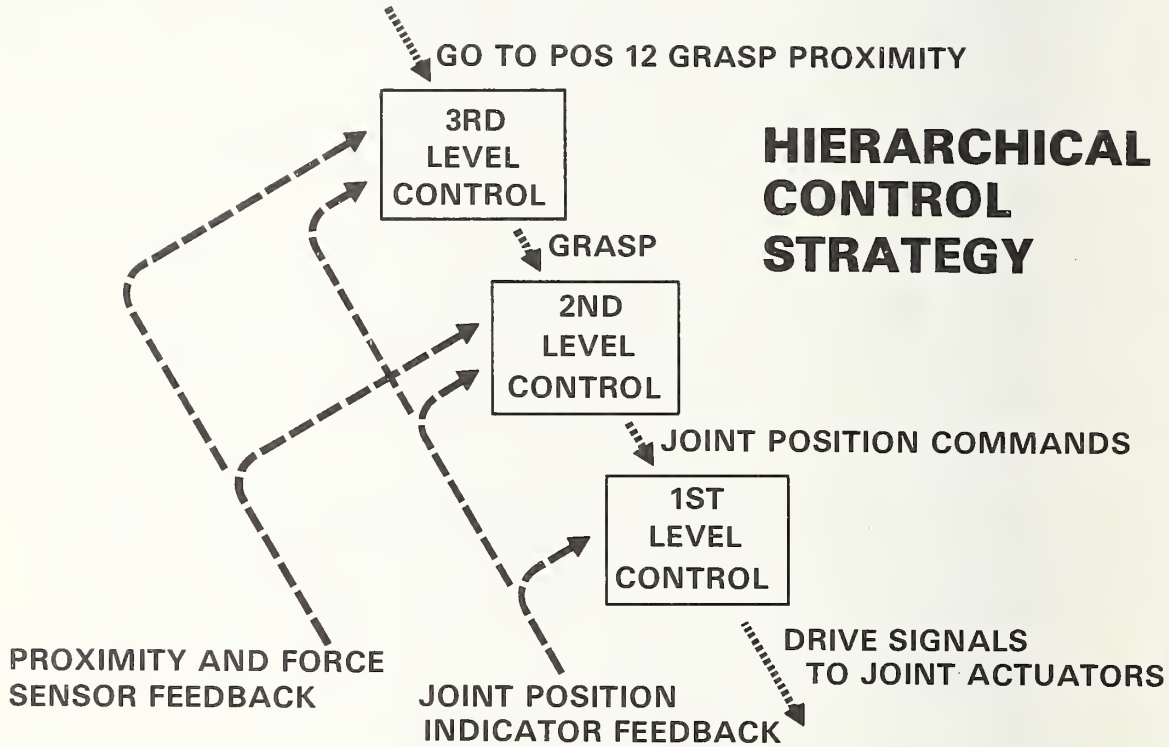


Figure 1

This chart shows the three levels in the control hierarchy. Each level receives commands from the next higher level and responds by generating ordered sequences of simpler commands to the next lower level. Sensory feedback is used to close control loops where appropriate.

The elemental move command GOTO POSITION 12 GRASP PROXIMITY causes the third level to generate a sequence of primitive commands (INTERPOLATE TO POSITION 12; MOVE X, Y, Z; SEARCH (with proximity sensors); BALANCE (with proximity sensors); GRASP). As a result of these primitive function commands interacting with the sensory feedback, the second level generates the correct sequence of joint position commands to the first level. The first level servos the joints to these positions by generating the necessary drive signals to the actuators.

The test bed for evaluating this control system has been a six axis research manipulator (Figure 2) which is controlled by one of the minicomputers in the Institute for Computer Sciences and Technology's Experimental Computer Facility at NBS.

Two types of manipulative tasks were designed to evaluate the performance of the different programming techniques at each level in the hierarchy.

The first task is a simple transfer operation (Figure 3). It requires the arm to move to a particular location, pick up an object, and place it at another defined location.

The second task is an insertion operation (Figure 4). A peg is moved to a hole and inserted. The diameter of the peg is 1.3 cm. The diameter of the hole is 1.5 cm. The insertion part of this task requires the hand to follow a straight line motion for a distance of about 7.6 cm.

These two tasks are meant to simulate the type of transfer and simple assembly operations performed by industrial robots.

Infrared proximity sensors (Figure 5) developed at NBS will be used to demonstrate how the control system can interact with sensory feedback data in real time. The sensors themselves are the subject of another report.

2.1.1-1 First Level of the Control Hierarchy

The lowest level in the hierarchy (Figure 1) is where servo control functions are computed. This is the level at which most industrial robots in use today are programmed and controlled. The input commands are joint positions which are compared to the feedback from the joint position indicators. If these values are different, a drive signal is generated to move each joint until the position error is nulled.

The control system at this level is shown schematically in Figure 6. During teaching or programming, a hand controlled unit is used that allows the operator to control the motion of each individual joint, using rate control. When the joints of the robot are in a desired configuration, the programmer presses a "record" button which stores the positional value of all of the joints in the memory of the robot. During playback, these position values are recalled from memory and compared with the actual measured positions of the joints. If there are any discrepancies, the servo system sends a command to the actuator to move the joint until the error disappears.

Most of the robots being sold today have controls of this type. These controls may be specially hardwired units, or they may be based on minicomputers or microcomputers, but the control concepts are the same. A typical robot may be able to store 100 program steps or more.

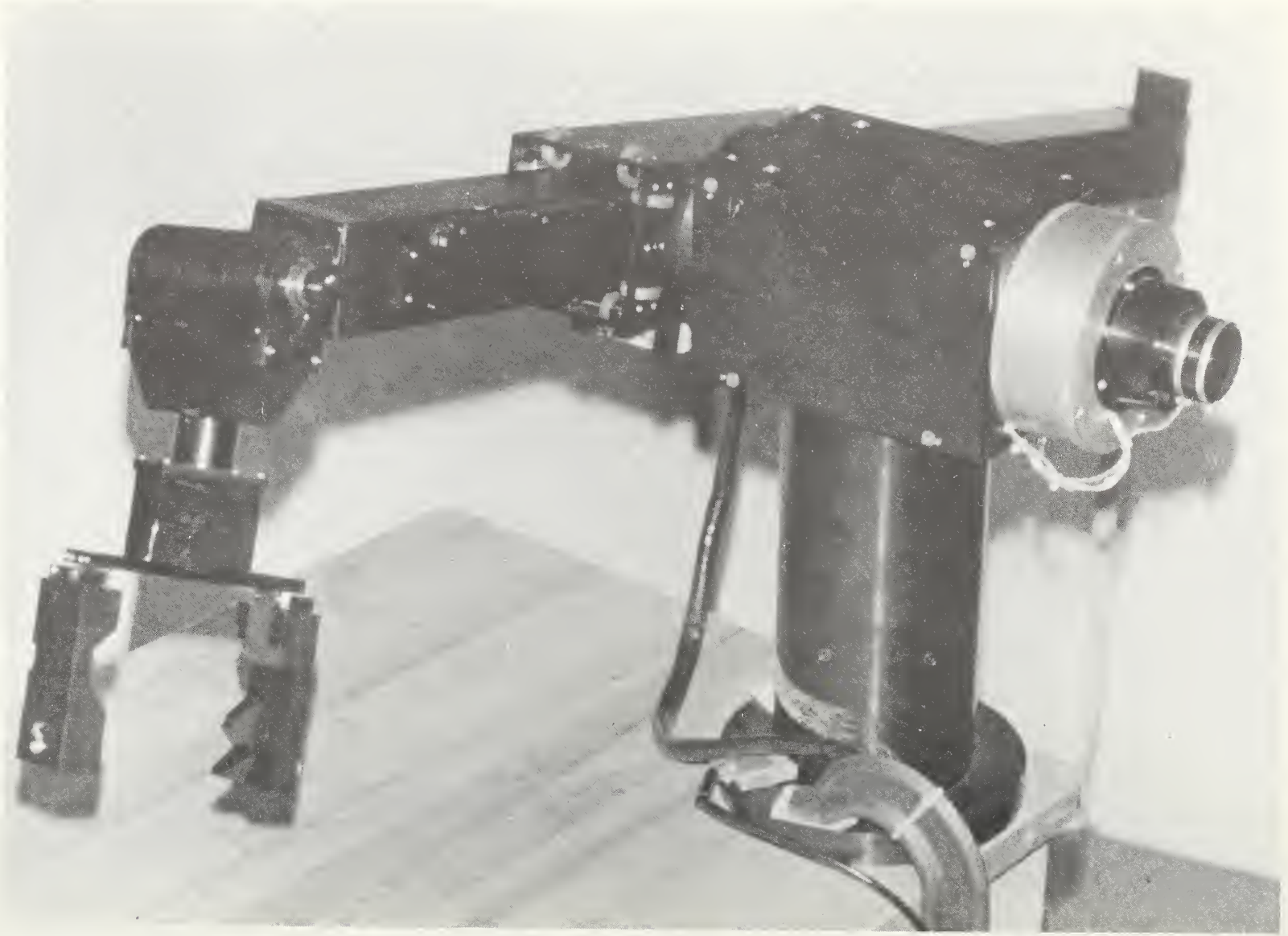


Figure 2 - Picture of 6-Axis Research Manipulator

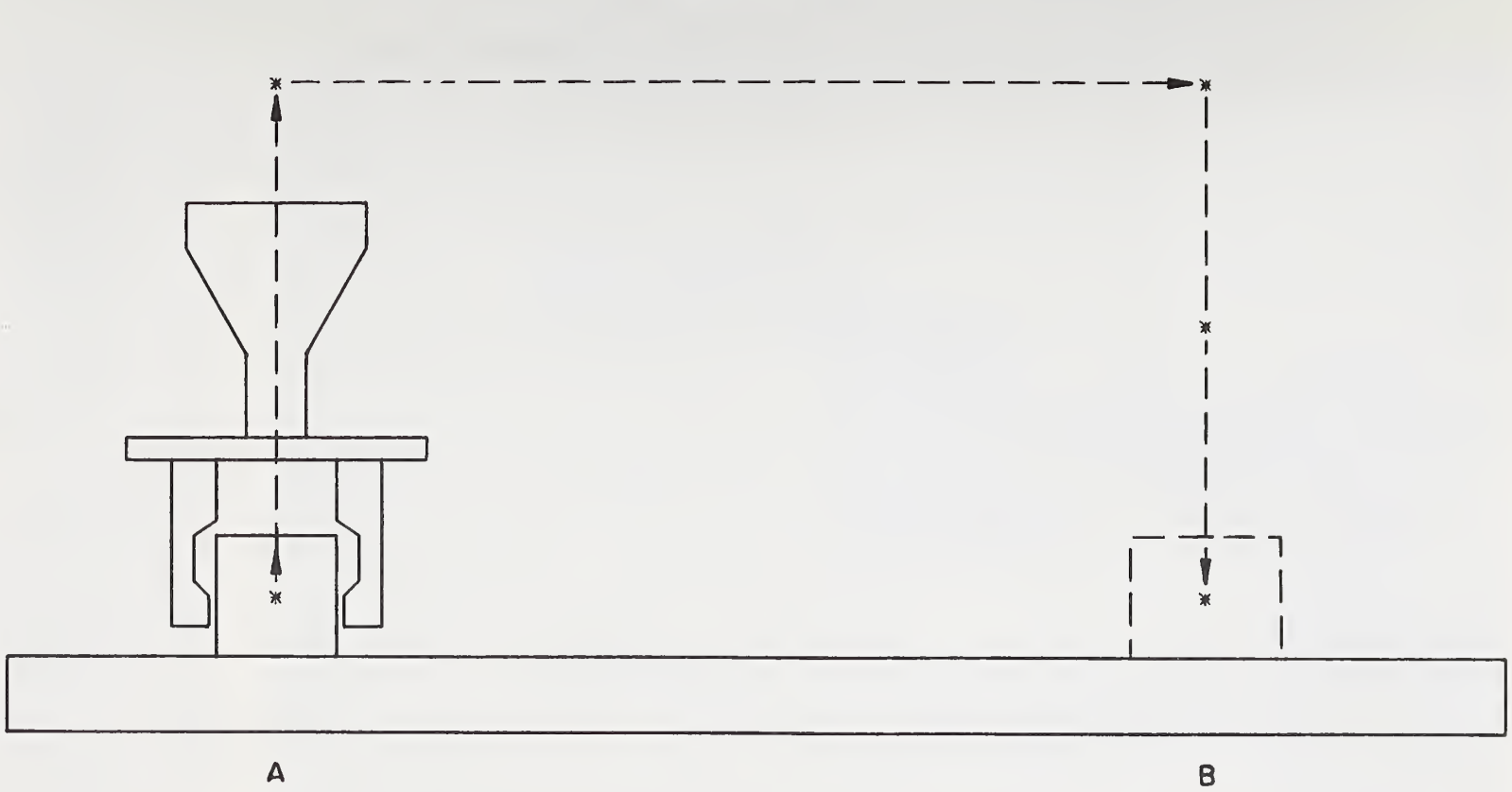


Figure 3 - This schematic represents the motion required for a transfer task. The hand has to pick up an object from one location, move to a point above this point to provide a reasonable lifting path, move to a point over the destination to provide an approach path to the destination and finally set the object down at this destination point. Thus, for this transfer task, four points would be recorded. The beginning and end locations had to be recorded within 3mm of the specified position.

INSERTION TASK

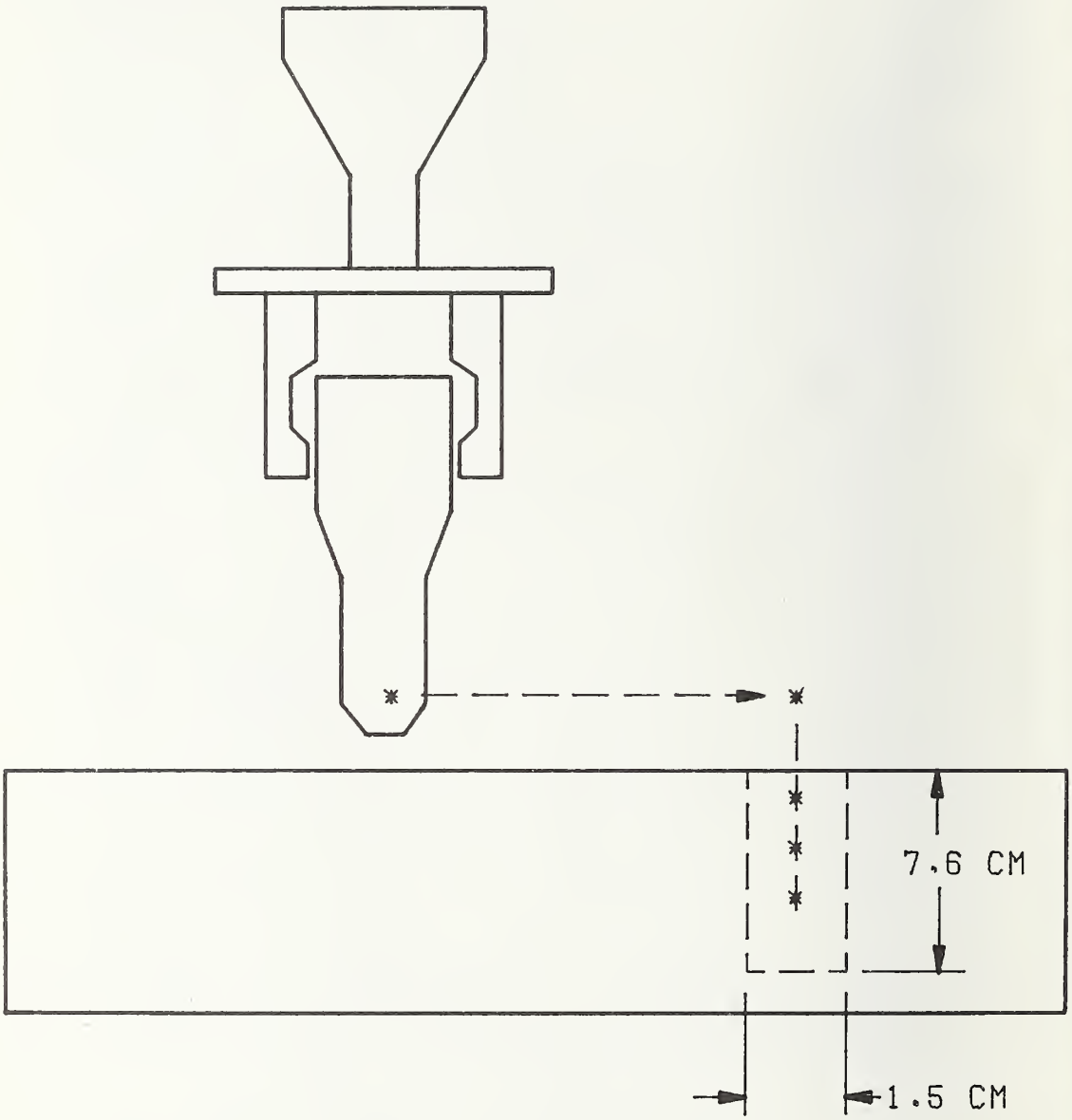


Figure 4 - This schematic represents the motion required for an insertion task. The hand is to move to a point over the hole, follow a straight line path into the hole, which required recording two additional points between the beginning of the approach path and the destination point.

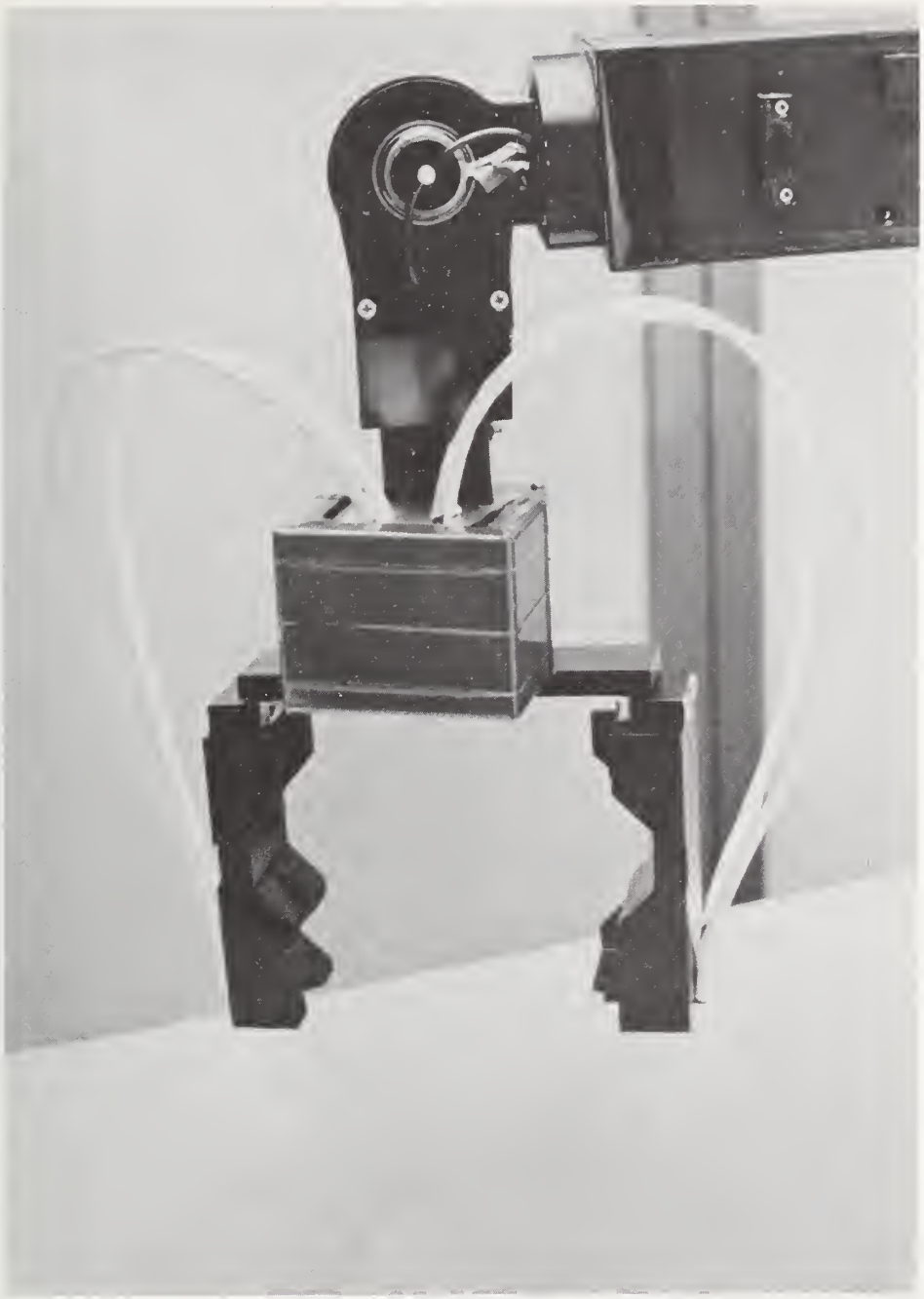


Figure 5 - Proximity Sensor affixed to a robot's gripper enables a robot to locate and grasp objects that are not precisely positioned. Infrared radiation, produced by light-emitting diodes and carried by fiber-optic bundles, is projected downward in two beams. Radiation reflected from the target enters a parallel set of fiber-optic bundles. Strength of reflected radiation acts as a feedback signal that informs robot how close gripper is to its target. Experimental system in photograph is under development at the National Bureau of Standards.

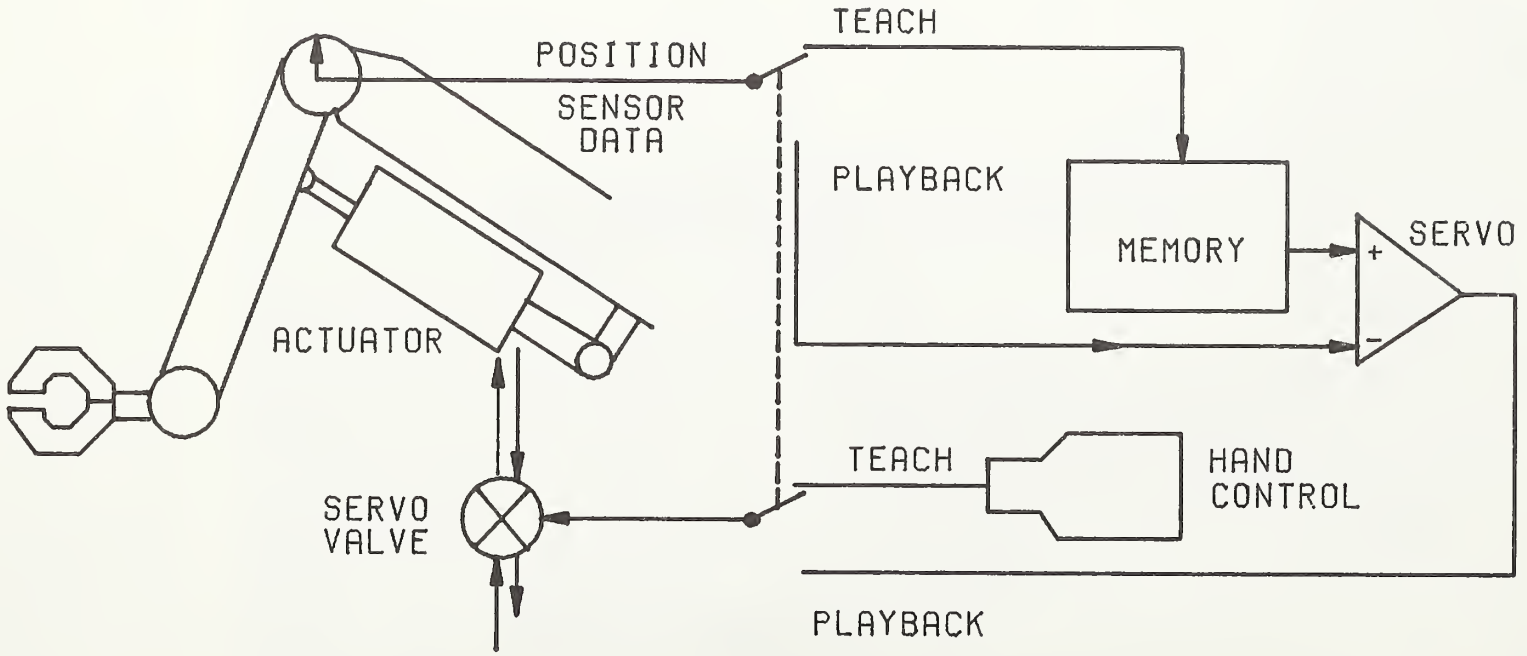


Figure 6 - Illustration of a typical first level control system used on industrial robots. The robot is led through a series of points by means of a teaching box which allows each of the joints to be moved by the operator. These points are recorded on a memory storage system. To repeat the operation, this switch is thrown to cause these recorded points to dirve the robot.

Some controls allow branching from one set of program steps to another, depending on external conditions. This feature is useful, for example, in spot welding of automobile bodies, where the control program must be changed to accommodate a mix of body styles on a typical automobile assembly line.

Some tasks require the path of the robot hand to follow a straight line (such as inserting a cutter into the spindle of a machine tool).

Programming a straight line motion with the rate control box is a particularly time consuming job since most industrial robots have one or more rotary joints that cause the hand to travel along a circular arc instead of a straight line (Figure 9).

Thus, a simple insertion task requires the operator to continually readjust a number of joints in order to program the hand to move along a straight line.

The transfer and insertion tasks described above were programmed by three different operators using a rate control box. Their times for the first level of control are shown in Figure 7 and 8.

This first level of control does not require the use of a computer. However, a computer is necessary to implement the second and third levels of this hierarchical control system. These higher control levels are required for increased capabilities in speed and ease of programming, and for real time interaction of the robot with the environment through the use of sensory feedback.

2.1.1-2 Second Level of the Control Hierarchy

The commands to the second level of the control system are calls to primitive function subroutines to be executed. These low level primitives are the basic, general purpose, operations that can be sequenced together to accomplish more complicated tasks. They are called, one at a time, by the different input commands such as GRASP, or RELEASE, or MOVE X, Y, Z, etc. A command call like GRASP will, together with whatever feedback is appropriate for this primitive, cause the second level to generate the correct sequence of joint position outputs to the next lower level (servo level) to accomplish this operation.

Programming at this second level is much enhanced over the first level since coordinate transformations are now possible with the computer. Inputs to the robot, to move it to the desired positions and orientations to be recorded, can now be in the form of values in an external coordinate system. Thus, the arm can be commanded in terms of X, Y, Z coordinate space through the use of a joystick. Moving the joystick in a desired direction provides inputs into the computer that are the delta X, Y, Z coordinate offsets from the present position. This X, Y, Z command becomes the input to the second level. The

BOX TRANSFER TASK

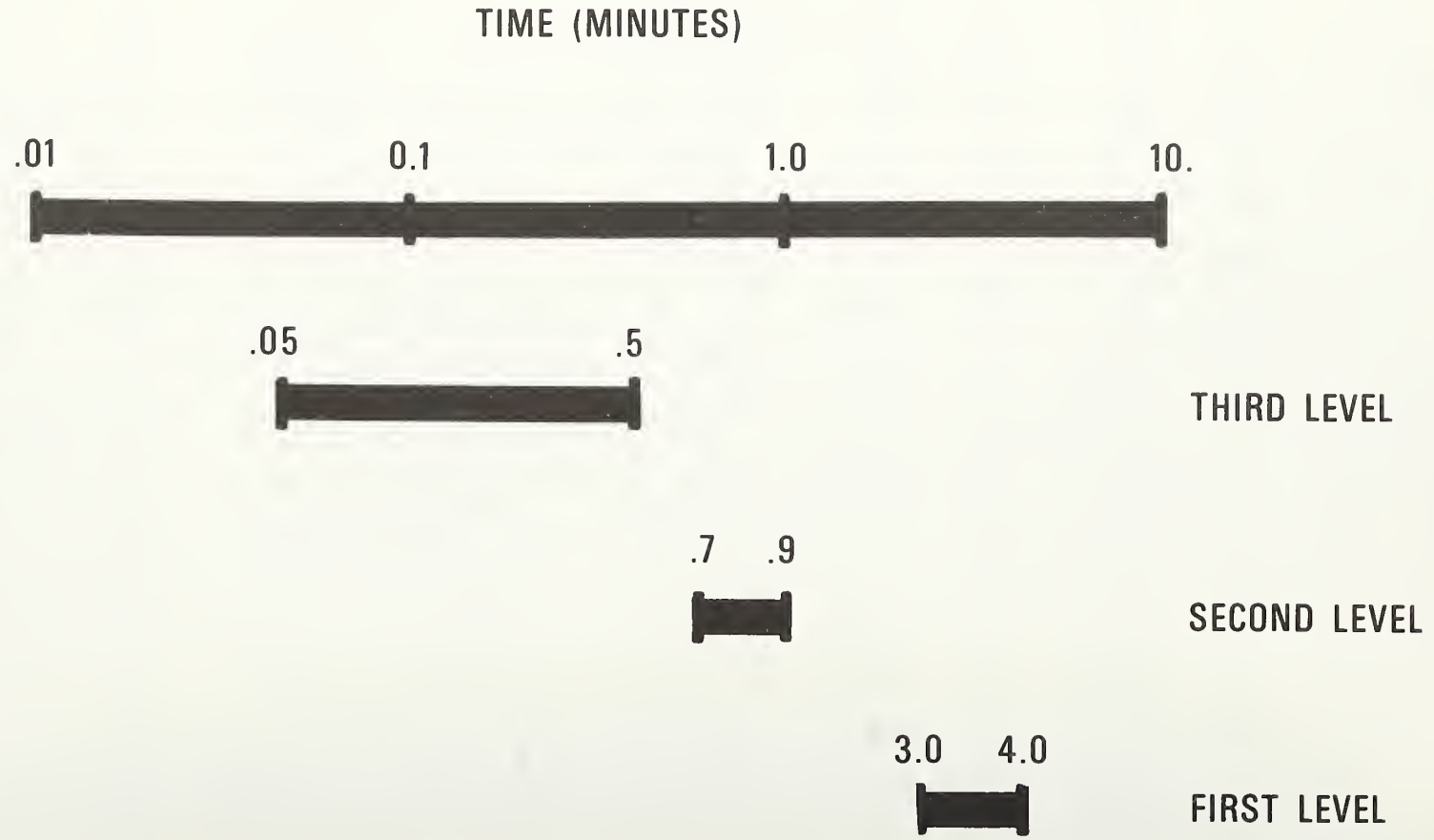


Figure 7 - Range of average times of three different operators to program a single transfer task. Programming was done with a rate control box at the first level, a joystick at the second level, and elemental moves typed into a computer at the third level.

INSERTION TASK

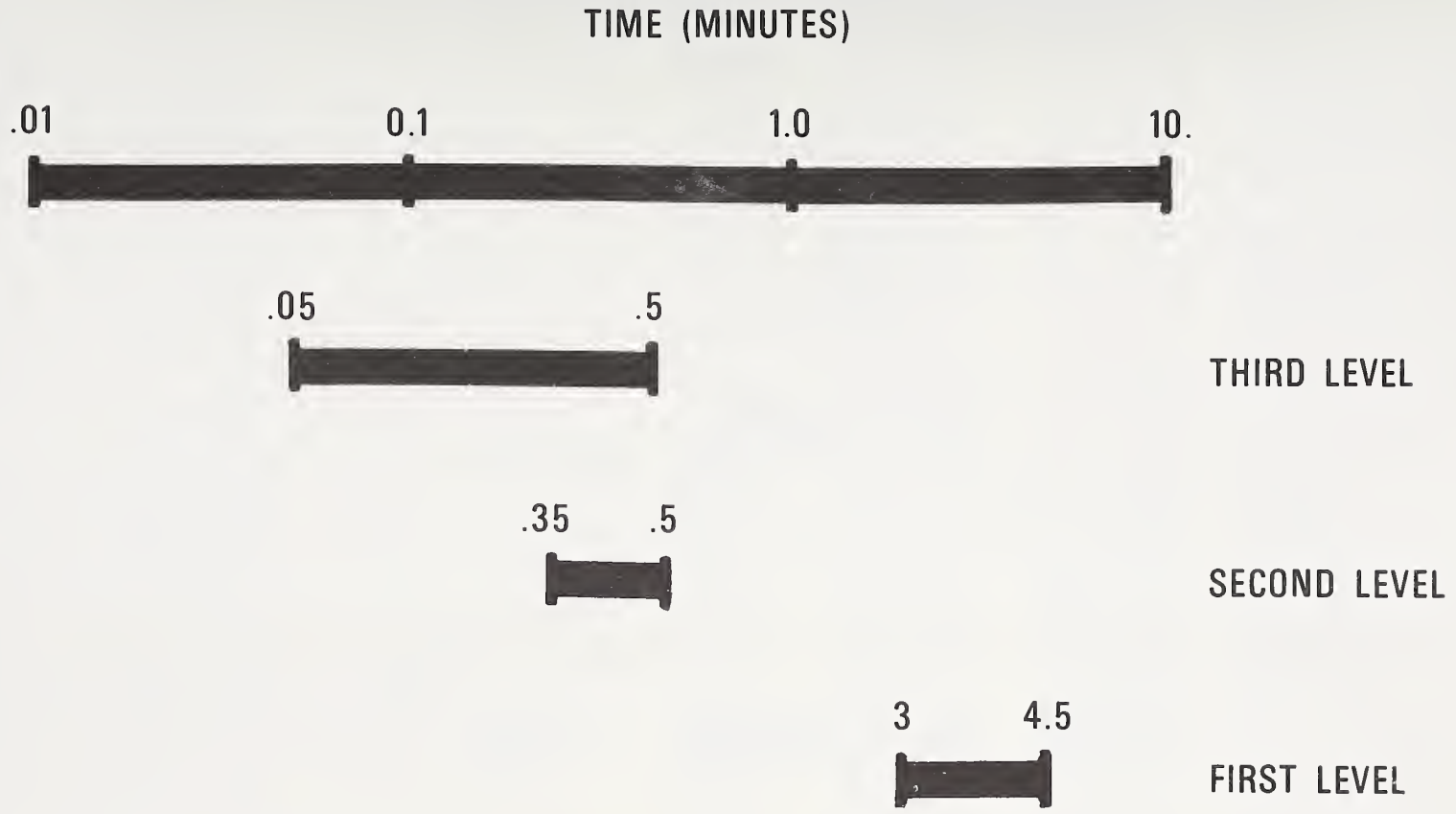


Figure 8 - Range of average times of three different operators to program a single insertion task. Programming was done with a rate control box at the first level, a joystick at the second level, and elemental moves typed into a computer at the third level.

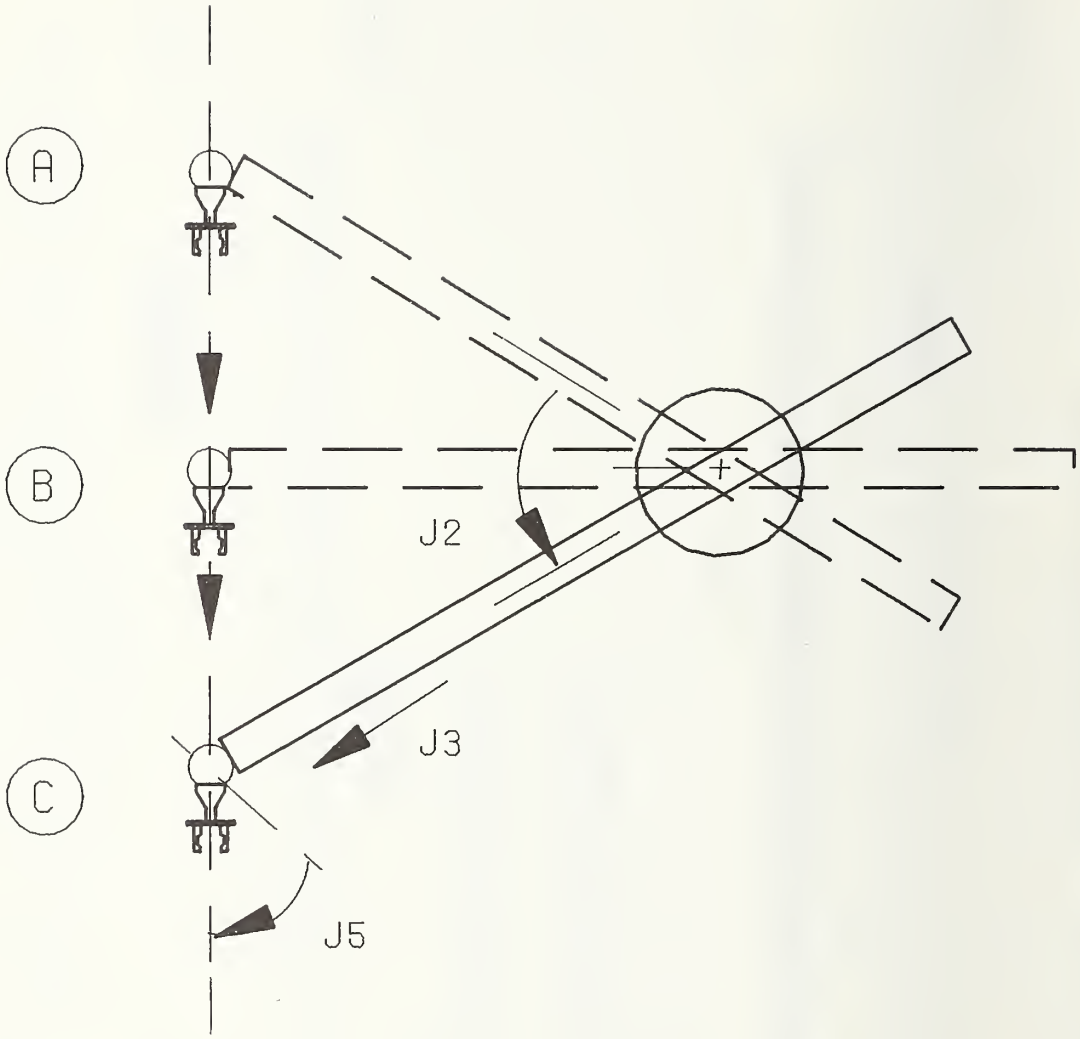


Figure 9 - This figure illustrates the required coordination of a number of joints in order to obtain a typical straight line motion of the hand. For each incremental move along the straight line path, joints 2 (elevation), 3 (boom), and 5 (wrist flex) must be adjusted.

coordinate transformation routine is then called to calculate all of the joint motions required to cause the robot's hand to move along the commanded straight line. The operator is one level removed from the servo system and, therefore, no longer has to worry about moving the individual joints. This is the power of a hierarchy. As higher levels are added, the input commands become simpler and more procedure oriented, while the sequences of the detailed operations required to accomplish the tasks are generated by the lower levels in response to these commands.

If the transfer and insertion tasks are programmed at the second level, using a joystick, there is a significant reduction in programming time (Figures 7 & 8). In addition, the programming becomes easier and less tedious since the operator no longer has to concern himself with what series of joint motions is required to move the arm to the new location.

In addition, the coordinate transformation routine makes it possible for the control system to interact with sensory data. Most sensors provide information that will require the robot to move along vectors in the sensor-based coordinate system, not in the joint coordinate system of the robot.

The sensor generated commands for motions of the arm in terms of the sensor's coordinate system are transformed into the proper joint coordinate values. Thus, causing real time dynamic interaction of the robot with its environment through sensor controlled movement.

2.1.1-3 Third Level of the Control Hierarchy

The third level in the control hierarchy receives its input commands in the form of elemental move commands. The elemental move is a basic unit building block in the description of a task. It is in the form of a motion and an operation. Most, if not all tasks, can be broken down into a sequence of these elemental move commands, where the hand of the robot executes some trajectory through space to a destination point and performs some function. Of course, the trajectory or the function may default to a null value. These elemental moves are programmed by the operator in the form of "GOTO" statements. An example of an elemental move command would be "GOTO PALLET (04), GRASP." This command, along with any appropriate sensory data, would generate a sequence of calls to the second level to execute the required primitive functions.

At this third level, the operator is programming in a much more task procedural language as opposed to the robot joint position language of the first level. The joint positions of the robot that define the location PALLET (04) still have to be recorded in a table of points. However, these points can be entered under joystick control or as the X, Y, Z coordinates of the location. Once a location is stored under

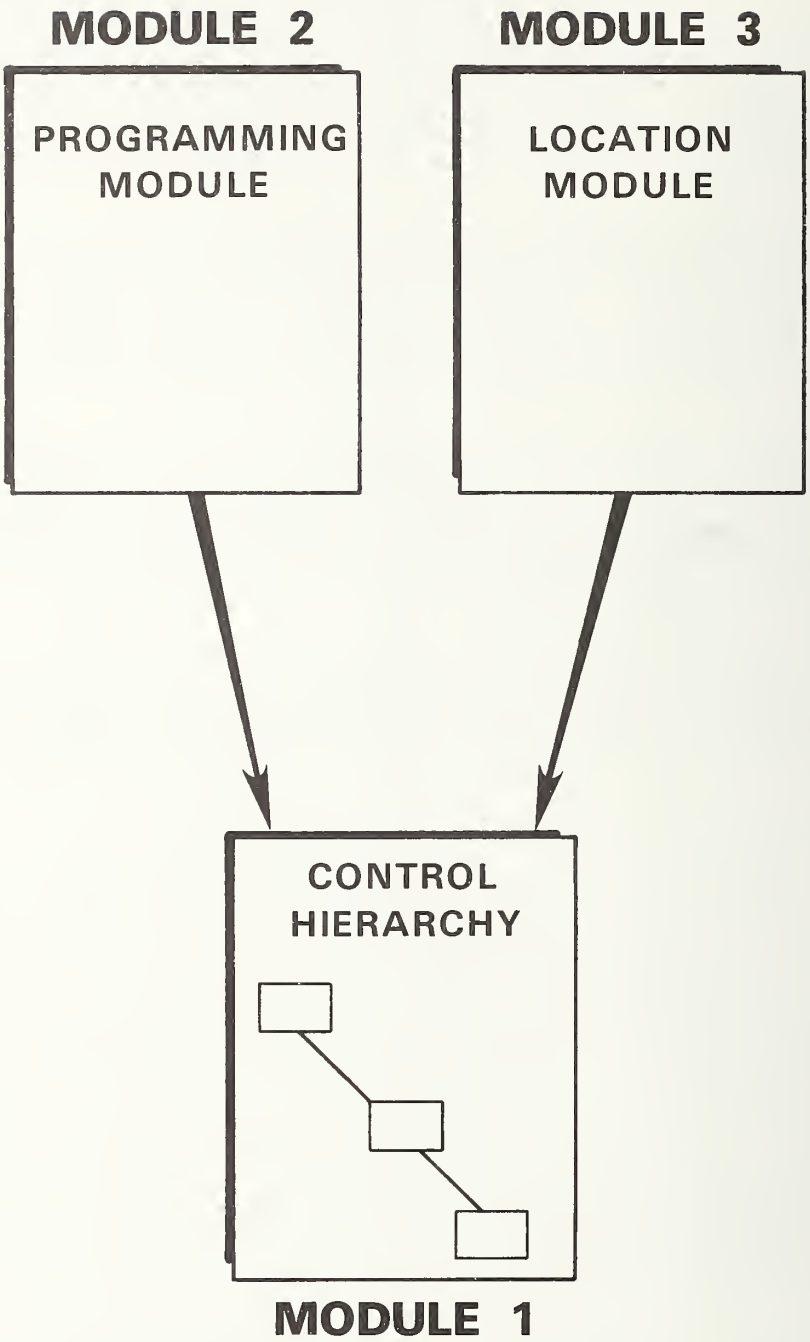


Figure 10 - The three modules of the control system in their relation to each other and the robot.

some arbitrary name (like PALLET (04)), it can be used in any number of elemental move statements. Of course, the stored locations can be programmed in any sequence, not just the order in which they were entered.

Recording and programming the transfer and insertion tasks at the third level shows a further reduction in programming time (Figure 7 & 8). The programming is even easier for the operator to accomplish since he is giving elemental move assignments which describe the task in much the same way as it would be described by one person to another. Large reductions in recording and programming time occur when arrays of locations are used as will be explained later in the report.

2.1.2 Program Module

The application program is entered in the program module (module #2) with the control system editor. The application program takes the form of a sequence of GOTO statements using arbitrarily named locations such as PALLET (04), NEUTRAL, VISE, etc. Each statement (elemental move) may also include a list of desired functions such as GRASP, INSERT, etc. These elemental move commands might be entered through a computer terminal by the operator, by an APT part programmer at the same time that he is writing the part programs for a NC tool, or through a function button programming box on the shop floor.

Thus, the program module produces a procedural description of the tasks to be performed in terms which are as independent as possible of the particular robot that may perform it.

2.1.3 Location Table Module

The values in the location table provide a relative description of the positions of all of the location points. These values can be postprocessed to specify the location points in the coordinate space of any particular industrial robot that is sufficiently flexible to carry out the task (i.e., in most cases, a six-axis robot). The concept is to allow these relative locations to be entered by the programmer as X, Y, Z coordinate locations at the same time he is creating the program module, or to be processed from some data base such as a CLDATA file of an APT program. The CLDATA file (the X, Y, Z and surface normal angle data) has a certain level of machine independence and can be postprocessed to provide the location points for a particular robot as well as for a particular machine tool. Another source of these location points is from a particular robot trained in its own workspace. These specific location points can be converted into standard coordinates for use by another robot through an inverse postprocessor.

Thus, an attempt has been made to separate the description of the task as much as possible from the particular robot that might carry out its operation.

The control system interfaces to the particular robot through that robot's own coordinate transformation subroutine. The coordinate transformation routine can be used with a post processor to generate the robot- specific location table from a robot-independent location table. It is also used during execution of the program for real time transformation between external or sensor-based coordinate systems and the robot's joint coordinate system.

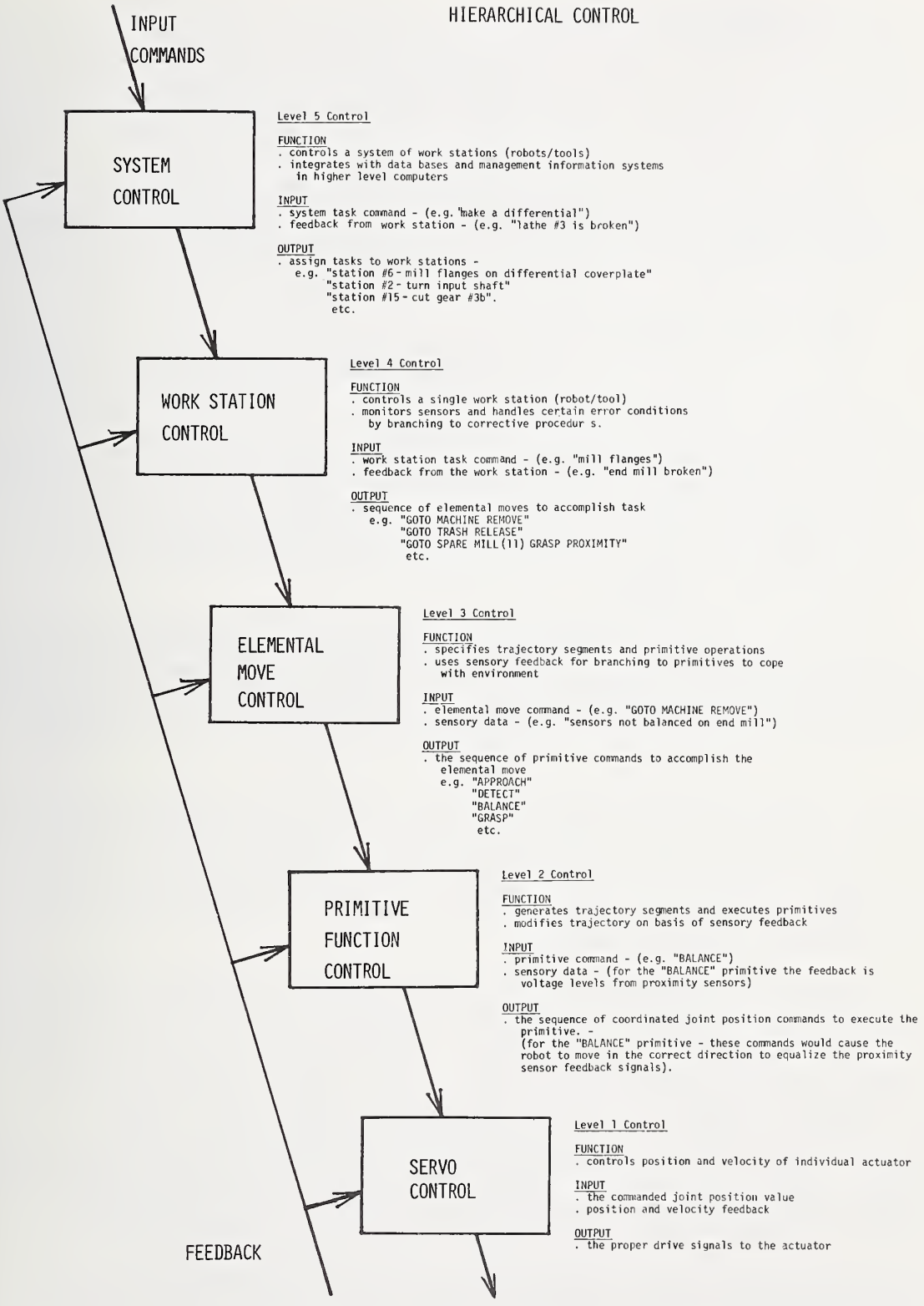


Figure 11 - The 5 Level Control Hierarchy

3. THE NEXT LEVELS OF CONTROL

Ongoing work at the Bureau includes the addition of the fourth and fifth levels of the control hierarchy (Figure 11), and the completion of the modular design, its standard interfaces and the postprocessors to provide the general purpose control system described above.

The next sections will discuss the basic conceptual functions of the fourth and fifth levels of control and an overall view of the complete control system to be integrated into the computer-aided manufacturing operation.

3.1 Fourth Level Control

This level of control takes care of the complete operation of a robot in its associated work station. Its input is in the form of a task to be completed, such as "SPOT WELD A CAR BODY" or "CUT 50 GEARS OF TYPE #36" etc. Sensory feedback data comes not only from the robot's sensors, but also from sensors throughout the work station. These work station sensors provide the additional feedback required to allow the robot to cope with all of the error conditions that are within its capability to correct. This reduces the need for external supervision and intervention to a minimum.

The input task command, together with the sensory feedback from the robot and the workstation, result in the fourth level sending out sequence of elemental move commands to the third level. Different prerecorded sequences of elemental move commands can be decided upon as a result of the particular input task and sensory feedback.

As an example, consider the task to make a particular gear from a pallet of gear blanks. To accomplish this task command, the fourth level will send the sequence of elemental moves to the third level to cause the robot to load and unload the parts from the pallet to the vise on a machine tool table, and to change cutters in the machine tool for the different cutting operations.

This sequence of elemental moves will be the main execution program. In addition, a number of sequences of elemental moves can be programmed and named to be used as subroutines. These will be sent to the third level when certain conditions arise. For example, suppose one of the cutters breaks while in the machine tool. A sensor on the tool or on the robot can report this data back to the fourth level. This condition will cause a branch to a preprogrammed sequence of elemental moves. This sequence will command the robot to remove the broken cutter from the tool and replace it with a new cutter. The program then returns control to the proper point in the execution program.

Thus, the fourth level control system has the responsibility of accomplishing tasks for an entire work station. This level receives

sufficient sensory data to cope with problems that might hinder the execution of that task, and outputs the preprogrammed sequences of elemental moves to correct these situations.

3.2 Fifth Level Control

The fifth level of control has the responsibility of accomplishing a project that might involve assigning a number of tasks to a number of different work stations; or scheduling a number of tasks to the same work station. Its feedback might consist of one of its fourth level control stations reporting back that a task has been completed, or that a machine tool is inoperative. This fifth level would respond by issuing a new task to the particular work station or rerouting materials to another work station and assigning it the task that the disabled station could no longer accomplish.

The fifth level interfaces to higher level computer-aided manufacturing and management information systems' data bases to provide them with information (processed feedback) about the work being done on the factory floor, maintenance and repair situations, productivity, efficiency of machine utilization, etc. Figure 12 gives an illustration of the processing of information both up and down the hierarchy.

These five levels of control carry out the responsibility of the actual manufacturing or production in a factory. They should integrate into total CAD/CAM systems as well as have the capability of a stand alone system at any level depending on the resources of the manufacturing unit involved.

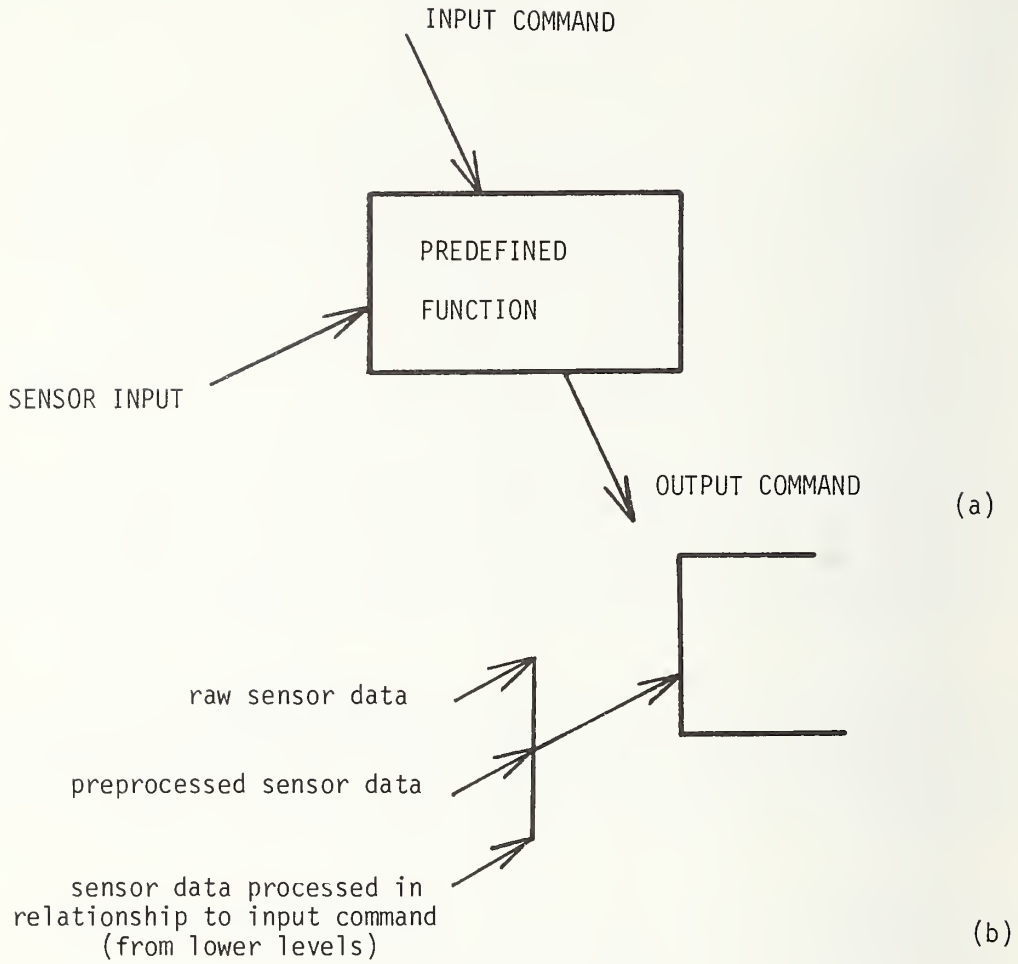


Figure 12 (a) & (b)

Each level in the hierarchy has information flowing into and out of it. The output being a function of the inputs (a).

Each level's sensory input is composed of three types (b). There is raw sensor data such as the voltage levels from proximity sensors indicating relative reflected intensity.

There is preprocessed sensor data such as might come from a vision system where sophisticated data manipulation and pattern recognition is performed by some sensor unit and its output is a sensor input to the control system.

The other sensor feedback is information from the lower levels of the hierarchy that is a reporting of their effectiveness in completing their input task. This takes the form of an interpretation of their sensor data in light of the input command. These are additional outputs from each level, not as commands to the next lower level, but as inputs to the higher levels. As with the other outputs they are also predetermined functions.

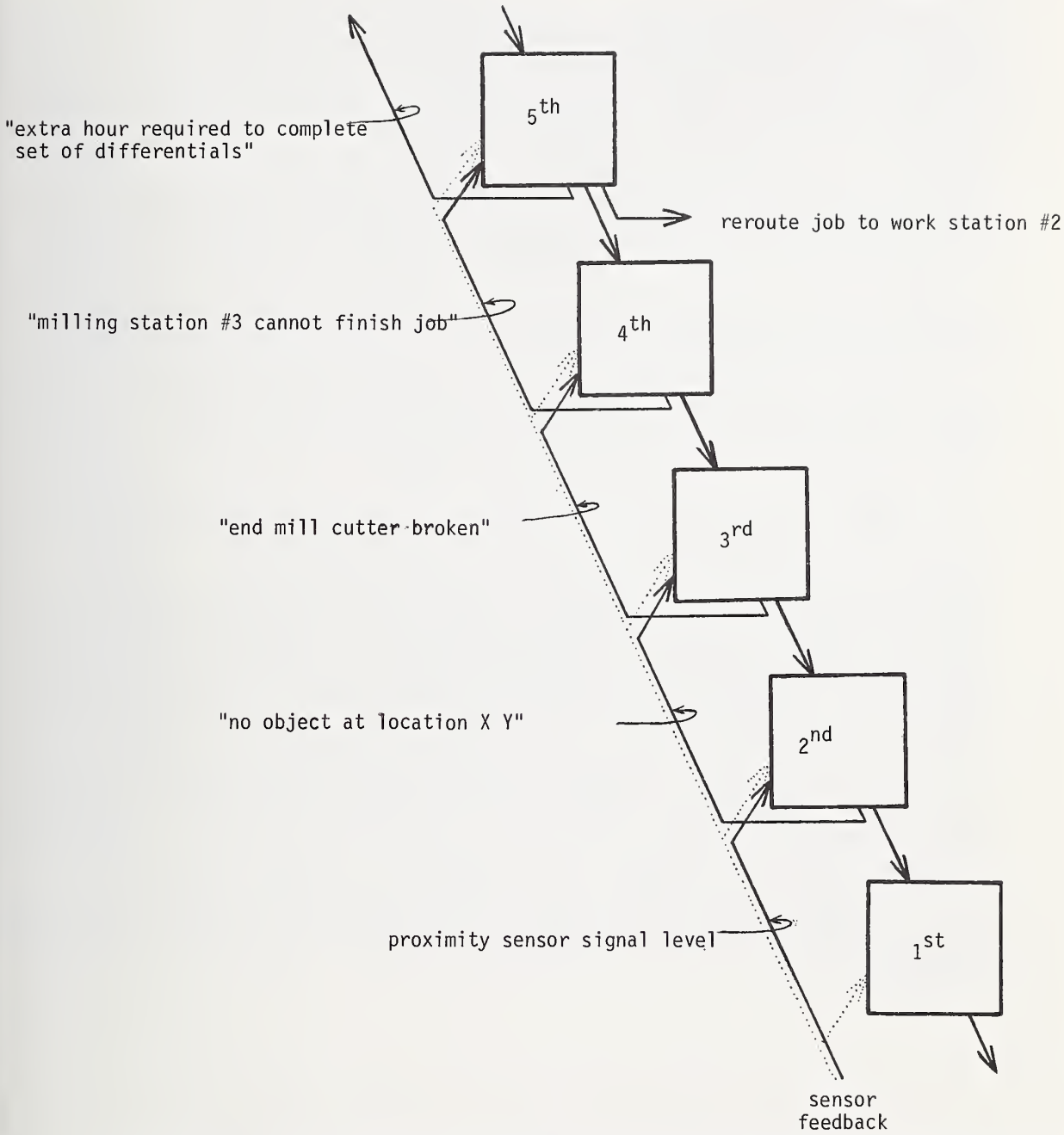


Figure 12 (c) - Thus, each level not only generates information in a descending path to the lower levels but also processes and reports information back up the hierarchy. At each higher level, this reporting takes on a more and more sophisticated form such that the fifth level may report to a management information system that an extra hour will be required to produce a set of differentials while the original raw sensor data to the second level might have been a proximity sensor signal level.

4. MODULAR DESIGN

Modular design offers several very important advantages. It provides a partitioning of the system into identifiable units, each of which is comprehensible and lends itself more easily to a solution. This modularization also aids in the separation of the control system into those sections that are of a general nature and therefore robot-independent, and those sections that are specific to the individual robot (e.g. the coordinate transformation routine, the servo system etc.). This allows the creation of a general purpose, universal control system that can run any robot if the robot-dependent modules are supplied.

Modularization encourages the use of standard interfaces which aids in the set up of the functional organization of the system and simplifies the information links between the modules themselves, and between these modules and the other components of the CAD/CAM system. It clearly defines where the responsibility for each function lies, which simplifies the writing and debugging of code.

The concept of modularization has been extended to the level of partitioning the control system module, the program module, and the location table module into a number of simple subroutines. In the control system, each primitive operation (like GRASP) is a separate subroutine which is called when the appropriate condition occurs. Therefore, once the control system architecture is supplied, the addition of new primitive functions is merely the addition of the new subroutines. Interaction with sensors is handled in the same way.

A sensor subroutine is written and added into the system. Whenever the system is commanded to interact with the sensor's feedback, a call to its subroutine is made. This usually includes calls to the coordinate transformation routine to provide the necessary joint position commands to cause motion of the robot in the sensor's coordinate system. In this way, sensors of any type can be implemented and the control system made to interact with their feedback by the addition of simple subroutines. This makes the control system flexible and easily modifiable for new applications.

5. NBS CONTROL SYSTEM PHILOSOPHY

This control system has been designed as a completely deterministic hierarchy of input - output patterns. The hierarchical architecture has provided a separation of responsibilities into different function levels. Each level becomes a group of preprogrammed function generators that respond to a limited set of input states with a defined set of output states.

If a response to a certain set of input conditions is not programmed, the control system can neither decide on the proper output nor "learn" the correct response. All of the intelligence to cope with the environment must come from human intervention in the form of preprogrammed functions. The location points and procedure must be specified for each task. A human operator must construct the appropriate subroutines to recognize and respond to the input patterns.

However, within this set of defined input states, this deterministic system exhibits goal directed, adaptive behavior, responding to sensory feedback to modify the robot's motions in real time. In this way, the assigned task is accomplished in spite of perturbations in the environment.

This use of a hierarchical system of simple deterministic functions has demonstrated large increases in capabilities as each new level is added. At each higher level, input patterns that correspond to more complex task assignments and more sophisticated processed sensory data, are used to generate sequences of simpler commands to the next lower level. The degree of "intelligent" behavior that can eventually be exhibited by a control system based on this concept is an important area of continuing research and development.

6. SUMMARY

The partitioning of the control system into simple modules and the interaction of these modules in a hierarchical fashion has proven to be a powerful technique in obtaining sophisticated sensor-controlled robot behavior at a minimum cost of programming time and computer size. This architecture also provides a flexible framework for the incorporation of additional functions and sensors.

II. USER'S GUIDE

II. USER'S GUIDE

II. USER'S GUIDE

1. EXECUTING APPLICATIONS PROGRAMS (Module #1 - Control Hierarchy)

The compiling and linking of subroutines into an executable load module are the only operator interface to the host computer's operating system. All other functions are maintained internally to the control system programs.

Real time control over the execution of the different control system modules is accomplished by a set of function switches. These switches are polled at periodic intervals (approximately every 20 milliseconds). Branches to different subroutine calls are made on the basis of these switch values. Thus, the operator can interact with the control system in real time. He can, for example, stop the arm in the middle of a trajectory modify the application program, and/or change a location point using either the joystick control or entering the X, Y, Z coordinate values, and then continue the execution of the new application program either at the point it was interrupted or at any other designated step in the program.

1.1 Control Hierarchy - Operator Interactions

See Table 1 for a listing of the panel switches and their interactions with the control system.

1.2 Initializing Program

Once execution begins (i.e. execution of the compiled and linked subroutines on the user's computer) the control system initially reads in from the storage unit the assigned application program and location table and prints the following messages on the terminal:

```
---SW 28 UP--- INTERRUPTS RUN PROGRAM
---SW 29 UP--- TEACH LOCATION TABLE
---SW 30 UP--- ENTER PROGRAM SEQUENCE
---SW 31 UP--- RECORD LOCATION TABLE ON DISC
*** TYPE [CR] *** TO CONTINUE
```

This identifies the functions of switches 28 thru 31. If an existing application program and location table is available, then, by typing in a carriage return, this application program will be initiated. After the entry of a carriage return the next message is sent to the terminal:

```
---SW 35 DOWN--- AND
*** TYPE [CR] *** TO INITIALIZE THE SERVOS WITH THE STARTING LOCATION
```

This reminds the operator to be sure that switch 35 is in the down position. This switch will start the execution of the program. When

Table 1

Control Hierarchy - Operator Interactions

Input Channel Number	Function	Description
26	Velocity Control	Voltage readings from a potentiometer used to control relative velocity of arm by specifying the number of interpolation points to be used.
28	Interrupt (abort)	When set, immediately stops execution and provides the operator with the option of editing program or location table or recording them on disc.
29	Enter Location	When set, this switch calls up the location module to allow entering of points in the location table. When reset, causes an exit from location module.
30	Edit Program	When set, this switch calls up the program module to allow editing of program table. When reset, causes an exit from program module.
31	Store on disc	When set, causes the location table to be stored on disc.
34	Pause	Temporarily suspends all execution so long as it is set. When reset, execution continues.
35	Start, Repeat	After initialization procedure, this switch is set to begin execution. The program will continue to repeat as long as this swtich remains set. If reset, it stops the program after execution of last line of program table.

Table 1 cont.

Input Channel Number	Function	Description
36	Straight Line Velocity	Voltage readings from a potentiometer used to control the velocity of the straight line portions of the trajectories.

the carriage return is typed in, the program sends the joint position values of the starting location to the servo system.

```
TURN ON SERVOS
---SW 35 UP--- TO EXECUTE PROGRAM.
---SW 35 REMAINS UP--- FOR REPEATED EXECUTION.
```

1.3 Executing Program

The servo system can now be turned on since a known set of joint position values has been commanded. The program cycles here, testing switch 35 until this switch is flipped up. When this switch is in the up position, the application program is executed. Switch 35 will not be polled again until after the last executable line of the application program. If the switch is still up at this time, the application program is repeated. If it is down, the control system cycles here, continuously polling switch 35.

Once switch 35 is flipped to the up position, the application program is executed without any further messages being printed on the terminal except error conditions.

1.4 Interrupting Execution

At any time, the interrupt (abort) switch (switch 28) can be set. This causes an immediate halt in the executing program, stopping the arm at its present position. The following message is printed on the terminal:

```
YOU ARE AT PROGRAM STEP 11
11  GOTO BOX(03) GRASP PROXIMITY VEL(50) SEND(0) WAIT(0)
---SW 29 UP--- TEACH LOCATION TABLE
---SW 30 UP--- ENTER PROGRAM SEQUENCE
---SW 31 UP---RECORD ON DISC
*** TYPE [CR] *** TO CONTINUE
```

In this instance, the control system was executing line 11 of the application program when interrupted. The elemental move statement (GOTO BOX (03) GRASP PROXIMITY) that is in line number 11 is also printed out.

1.5 Continuing After Interrupt

After the desired changes have been made, a carriage return is entered and the system responds with:

```
WHICH PROGRAM STEP DO YOU WANT?
```

The operator can respond with any program line number, and execution will begin at that point in the program. If a zero or carriage return is entered, the initialization phase is carried out and the system awaits switch 35 to be set to begin executing the program.

Table 2

Program Module - Operator Interactions

Input Channel Number	Function	Description
28	Interrupt (abort)	When set immediately stops execution of the program and allows the program module to be called.
30	Edit Program	When set, calls up the program module to allow editing of the program table. When reset, causes an exit from the program module.

Table 3

Editor Commands

Command Symbol	Function
Insert (XX)*	Specifies the program table line number XX where new line(s) are to be inserted.
Print (XX-YY)	Specifies the first (XX) and last (YY) line of the section of the program table to be printed. Once print command given, a minus sign will cause previous line to be printed, a carriage return will cause the next line to be printed.
Delete (XX-YY)	Specifies the first (XX) and last (YY) line of the section of the program table to be deleted. This section will be printed out as it is deleted.
Start NNN	Once the insert command has been called, this command enters the name (NNN) of the starting location in the program.
Loop	Specifies that all of the following elemental move inputs between this command and the command FINISH are to be repeated the number of times required by the indexed values of the location names used.
Finish	Causes the termination of the LOOP sequence.
Name	Causes the names of the indexed locations to be printed out on terminal with the option to change them.
Avoid XX $I_1 I_2 \cdots I_{10}$ YY	Causes the insertion of up to 10 (I_{10}) additional intermediate location points between the specified location points XX and YY. Used to create special trajectories using detailed path segments.

* All editor commands default to just their first letter for faster editing.

Table 4

Elemental Move Functions

Command Symbol	Function
Goto XX(YY-ZZ)*	Specifies a destination point (XX) for this trajectory. If it is an indexed location, the value of the particular index (YY) is specified. If it is used in a loop, then the range of indexed values(YY-ZZ) is specified.
Grasp	A grasp command - the fingers close on an object with a predefined amount of force.
Release	A release command - the fingers open completely.
Grasp Proximity	Center hand over object using the proximity sensors, then pick it up.
Release Proximity	Locate an object with proximity sensors and place the held object on top of it.
Unstack	Locate top of a stack with the proximity sensors, then pick up the top object.
Touch	Locate the top of an object of undefined height using proximity sensors.
Detect	Locate an object using the proximity sensors.
Balance	Center hand over an object using the proximity sensors.
Line	Move to the destination point through a straight line trajectory rather than an interpolated trajectory.
Send (XX)	Cause channel number XX to output a +5 voltage level.
Wait (XX)	Suspend execution until channel number XX receives +5 voltage signal.

* All elemental move commands default to just their first letter for faster entry.

2. ENTERING APPLICATION PROGRAMS (Module #2 - Program Module)

If a new program is to be entered or the present one modified, then switch 30 is set and a carriage return typed.

2.1 Program Module - Operator Interactions

See Table #2 for a summary of the operator interactions through the switch panel.

2.2 Indexed Location Name Entry

When the editor is called, the names that have been assigned indexed values are printed out on the terminal.

```
THESE ARE THE CURRENT INDEXED LOCATIONS
BOX STACK
DO YOU WANT TO ENTER NEW INDEXED NAMES?
```

If the answer is yes, then the program responds with:

```
ENTER INDEXED NAMES
(E.G. PALLET CUTTER STACK)
```

As an example, consider a pallet with an array of parts to be machined by a milling tool. The program to be entered should cause the robot to load and unload these parts from the vise, insert the correct end mill cutter in the tool at the beginning and remove it at the end, and air brush off the chips after each operation. For this task, PALLET will be set up with indexed locations corresponding to the array of parts on the pallet. Therefore, in answer to the request for indexed names made above, the operator enters the name PALLET:

```
% PALLET*
```

The system responds with the message:

```
EDITOR IS NOW AVAILABLE FOR PROGRAM ENTRY
```

---

*All operator entered commands will be preceded by a percent sign (%).

2.3 Editor Commands

The editor that has been written for this control system is line oriented and has commands to insert, delete or print specified line numbers.

2.3.1 Table of Editor Commands

See Table #3 for summary of all of the editor commands.

2.3.2 Elemental Move Functions

See Table #4 for a description of all of the presently implemented function commands for writing an application program.

2.3.3 INSERT and LOOP Commands

To enter the application program that will accomplish the machining task described above, the following commands are typed in:

```
% INSERT  01
% START NEUTRAL
% GOTO ENDMILL GRASP
% GOTO TOOL RELEASE
% LOOP
% GOTO PALLET (01 - 06) GRASP PROX
% GOTO VISE RELEASE
% GOTO NEUTRAL SEND (03)  WAIT (13)
% GOTO AIRHOSE GRASP
% GOTO CHIP VELOCITY (10)
% GOTO AIRHOSE RELEASE
% GOTO VISE GRASP
% GOTO PALLET (01 - 06) RELEASE
% FINISH
% GOTO TOOL GRASP
% GOTO ENDMILL RELEASE
% GOTO NEUTRAL SEND (04)
```

The above program, which is 14 statements long, will cause the robot to execute the following actions - start at some initial location named "neutral" - move from this location to the location of the end mill cutter - pick up the end mill cutter and insert it into the spindle of the milling tool - move over to the first work piece on the pallet, locate it with proximity sensors, pick it up and put it in the vise - go back to its neutral safe position -send a voltage level out on channel three to cause the machine tool to begin cutting - wait until the interlock signal from the machine tool on channel 13 goes high, indicating that the machining operation is finished - go to the location of the air hose - pick it up and blow off the chips by taking the air hose to a position named "chip" which is above the vise - replace the air hose - go to the vise and remove the machined part -

return the part to the pallet - locate and pick up the next part - repeat all of the above procedures for all six parts - then remove the end mill cutter from the tool and replace in the tool rack - go back to the neutral position and send out a signal on output channel four indicating that the job is finished.

The advantage of indexing names is obvious here, where the task has to be described only one time for one part. The "loop" command will cause all of the statements between it and the "finish" command to be repeated, for each of the six pallet positions.

2.3.4 PRINT Command

The edit program immediately generates the complete sequence of elemental move commands for all of the indexed positions. Thus, if a print command is given to display lines one thru 20, the following would occur:

```
%  PRINT 01 - 20
   START N*
01 GOTO  E           GRASP          VEL(50)   SEND(0)    WAIT(0)
02 GOTO  T           RELEASE        VEL(50)   SEND(0)    WAIT(0)
03 GOTO  PALLET(01)  GRASP PROX     VEL(50)   SEND(0)    WAIT(0)
04 GOTO  V           RELEASE        VEL(50)   SEND(0)    WAIT(0)
05 GOTO  N                          VEL(50)   SEND(03)   WAIT(13)
06 GOTO  A           GRASP          VEL(50)   SEND(0)    WAIT(0)
07 GOTO  C                          VEL(10)   SEND(0)    WAIT(0)
08 GOTO  A           RELEASE        VEL(50)   SEND(0)    WAIT(0)
09 GOTO  V           GRASP          VEL(50)   SEND(0)    WAIT(0)
10 GOTO  PALLET(01)  RELEASE        VEL(50)   SEND(0)    WAIT(0)
11 GOTO  PALLET(02)  GRASP PROX     VEL(50)   SEND(0)    WAIT(0)
12 GOTO  V           RELEASE        VEL(50)   SEND(0)    WAIT(0)
13 GOTO  N                          VEL(50)   SEND(03)   WAIT(13)
14 GOTO  A           GRASP          VEL(50)   SEND(0)    WAIT(0)
15 GOTO  C                          VEL(10)   SEND(0)    WAIT(0)
16 GOTO  A           RELEASE        VEL(50)   SEND(0)    WAIT(0)
17 GOTO  V           GRASP          VEL(50)   SEND(0)    WAIT(0)
18 GOTO  PALLET(02)  RELEASE        VEL(50)   SEND(0)    WAIT(0)
19 GOTO  PALLET(03)  GRASP PROX     VEL(50)   SEND(0)    WAIT(0)
20 GOTO  V           RELEASE        VEL(50)   SEND(0)    WAIT(0)
```

As can be seen, the program has been expended to its full sequence of elemental moves. The channel numbers for the interlocking commands "SEND" and "WAIT" are listed for each program line. A velocity value is also listed for each line. This is the maximum velocity that the hand is to reach for that particular move. If it is not specified during programming then a default value of 50 cm/sec is entered in the program sequence.

---

*All location names other than indexed names, default to their first letter.

2.3.5 AVOID Command

The program entered above is a minimum description of the task to be completed. There might, however, be peculiarities about the work environment that will prevent the trajectories from being executed as described. For example, suppose there is a building support column between the "NEUTRAL" position and the "VISE" position. To prevent the arm from colliding with the column, the following entry can be made:

```
% AVOID VISE POINT NEUTRAL
```

From the single entry of this one command, the entire application program will be scanned for all instances where the trajectory to be executed is between the vise and the neutral position. At every one of these points in the program, the additional elemental move - "GOTO POINT" - will be inserted. Thus, if a print out of lines three to seven is called for:

```
%   PRINT 03 - 07
03  GOTO PALLET(01)     GRASP PROX      VEL(50)   SEND(0)   WAIT(0)
04  GOTO V              RELEASE         VEL(50)   SEND(0)   WAIT(0)
05  GOTO P                              VEL(50)   SEND(0)   WAIT(0)
06  GOTO N                              VEL(50)   SEND(03)  WAIT(13)
07  GOTO A              GRASP           VEL(50)   SEND(0)   WAIT(0)
```

2.3.6 DELETE Command

Once a program is written it can easily be modified by the "DELETE" and "INSERT" editor commands. As an example, suppose this pallet of parts was to go to a work station to have a number of holes drilled in the parts. The only difference in the description of the task is that now a drill bit instead of an end mill cutter is to be inserted into the machine tool.

To accomplish this program change, line one will be deleted and a new line will be inserted. This new line will require the robot to go to a position named "DRILL" to pick up the drill bit to be used for this machining operation. To delete line one, the following command is given:

```
%  DELETE 01
01 GOTO    E     GRASP          VEL(50)        SEND(0)   WAIT(0)
```

When a delete command is given, the system deletes the specified line(s) and prints it (them) out on the terminal. The remainder of the program is closed up around the deleted line so that a print out of lines one to three would look like this:

```
% PRINT 01 - 03
   START     NEUTRAL
01 GOTO      T           RELEASE        VEL(50)   SEND(0)   WAIT(0)
02 GOTO      PALLET(01)  GRASP PROX     VEL(50)   SEND(0)   WAIT(0)
03 GOTO      V           RELEASE        VEL(50)   SEND(0)   WAIT(0)
```

To enter a new line, the "INSERT" command is given:

```
% INSERT 01
   START NEUTRAL
01 GOTO  T        RELEASE        VEL(50)     SEND(0)     WAIT(0)
```

The "INSERT" command causes the line presently at the specified line number to be printed out. Any new line(s) inserted here will be placed in the program in front of the displayed line. The new line can now be entered:

```
% GOTO DRILL GRASP
```

If a print out of lines one to three is called for:

```
% PRINT 01 - 03
   START     NEUTRAL
01 GOTO      D           GRASP          VEL(50)   SEND(0)   WAIT(0)
02 GOTO      T           RELEASE        VEL(50)   SEND(0)   WAIT(0)
03 GOTO      PALLET(01)  GRASP PROX     VEL(50)   SEND(0)   WAIT(0)
```

The new line has been inserted and the other program lines moved back to accommodate it. This change is also to be made at the end of the program where the drill bit is removed from the machine tool and returned to its holder.

Thus, by two "DELETE" and "INSERT" commands, the program has been modified to cause the next machining operation to be accomplished on this pallet of parts.

2.3.7 RECORD Command

Once the application program is written, it can be stored on disc for future use. The editor command "RECORD" accomplishes this:

```
% RECORD
   THE PROGRAM HAS BEEN STORED ON DISC
```

2.4 New Functions

If there are additional functions that are required, for example, a special insertion technique is required to place a cutter into a spindle, then these special motions and perhaps the use of touch or force sensing to adjust the positioning of the cutter as it is

inserted into the spindle, can be defined in a new FORTRAN subroutine or combination of existing subroutines. Then this subroutine(s) can be specified by a single function call like INSERT so that a command to insert a cutter into a tool might be programmed as:

```
%  GOTO TOOL INSERT
```

The addition of these extra functions and their programming code words is a simple modification with this control system and will be explained in some detail in the next chapter.

2.5 Exit Program Module

To exit the program module, switch 30 is flipped down and a carriage return is typed in. The control system responds with the following list of options:

```
---SW 29 UP---  TEACH LOCATION TABLE
---SW 30 UP---  ENTER PROGRAM SEQUENCE
---SW 31 UP---  RECORD LOCATION TABLE ON DISC
*** TYPE [CR] ***  TO CONTINUE
```

The operator can now set switch 29 and type a carriage return to call up the location table module with its associated routines.

Table 5

Location Module - Operator Interactions

Input Channel Number	Function	Description
28	Interrupt (abort)	Stops execution of program so location module can be called.
29	Enter Location	Causes location module to be called up so location points can be entered. Resetting this switch causes an exit from the location module.
31	Coordinate Printout	While under joystick control, this switch causes X, Y, Z coordinate values of the present location of the arm to be printed out. Resetting switch returns control of joystick.
32	Joystick Control	Causes the signals from the joystick box to control the motions of the arm. Resetting this switch causes the present location of the arm to be stored in the location table.
34	X, Y, Z Entry	Allows location point to be entered as an X, Y, Z coordinate position.

3. ENTERING LOCATION POINTS (Module #3 - Location Module)

Once the program is written, it only remains that those location names referenced in the program have their corresponding position values stored in the location table. This section explains how the user is to enter these location points and what data the system will request to completely describe these points.

3.1 Location Module - Table of Operator Interactions

See Table #5 for a summary of the operator's interactions through the function switch panel.

3.2 Mode of Entry

As in the programming module, the names of the indexed locations are printed on the terminal and the option to change them is given.

The system then prints out the following message to indicate which panel switches are to be used for the different modes of location point entry:

```
---SW 32 UP---  FOR JOYSTICK CONTROL
---SW 34 UP---  FOR XYZ ENTRY
---SW 32 & 34 DOWN---  FOR MANUAL ENTRY
*** TYPE +1 ***  TO RECORD AN ARRAY OF INDEXED LOCATIONS
*** TYPE [CR] ***  TO RECORD LOCATIONS ONE AT A TIME
```

3.3 Array Entry

To record the array of location points for the six positions on the pallet in the previous example and to use the joystick to maneuver the arm to the proper positions, requires panel switch 32 to be set and a +1 to be typed on the terminal.

The system responds with a series of requests. The operator's responses will exactly specify the array of locations to be recorded:

```
ENTER LOCATION NAME AND INDEX NUMBERS
E.G. PALLET (01-20)
```

The array to be specified is the six positions on the pallet as shown in Figure 13a, so the following is entered:

```
%  PALLET (01-06)

   ENTER ARRAY DIMENSIONS
   (E.G. 4,5)
```

The array of parts on the pallet, as shown in Figure 13a, is a 3 by 2 array therefore the entry is:

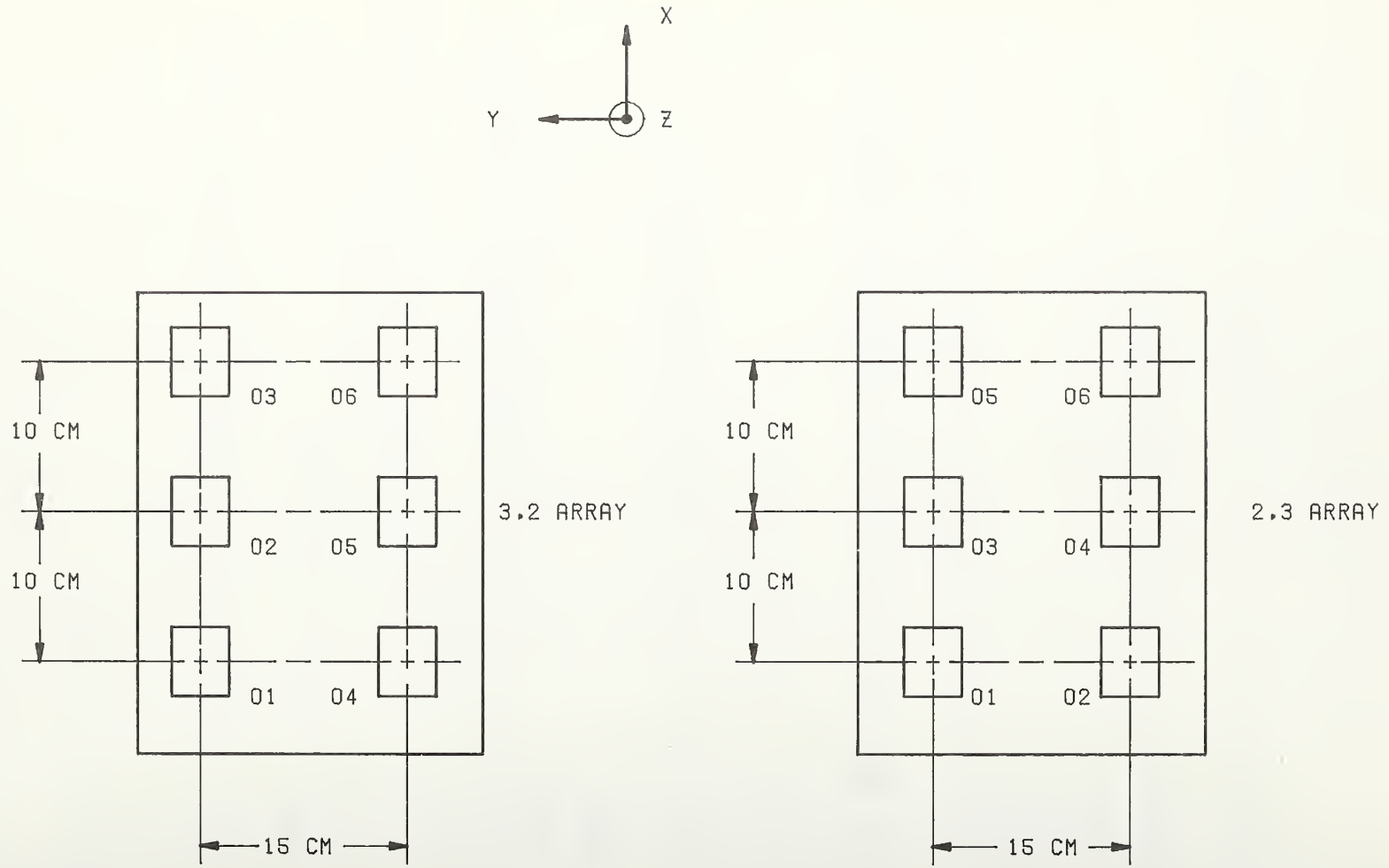


Figure 13 - The two alternate ways an array can be specified for entry. The first is designated a 3,2 array. The first dimension is always along the row of ascending indexed values starting at 01 (01-02-03). This same array could also be specified as a 2,3 array, in which case the control system would assume the indexed values of the location prints to be as pictured in B.

% 3,2

ENTRY X,Y,Z DELTAS (DECIMAL CM) BETWEEN ARRAY POSITIONS ALONG FIRST DIMENSION.

The first dimension is along the sequential ascending index values (i.e. along the row 01-02-03), therefore the delta X, Y, Z offsets will be +10. cm in the X, 0.0 cm in the Y and 0.0 cm in the Z (given that the pallet lies in an X-Y plane) and will be entered as:

% 10.,0,0

ENTER X,Y,Z DELTAS (DECIMAL CM) BETWEEN ARRAY POSITIONS ALONG SECOND DIMENSION.

% 0,-15.,0

Control is now turned over to the joystick box to move the arm to the correct position for PALLET(01). The following message is also sent out to the terminal to provide further information.

TO RETURN TO THE TEACH LOCATION MODE FLIP SW 29 TO THE DOWN POSITION.
FLIP SW 31 UP TO DISPLAY XYZ VALUES.

At any time, while the arm is under joystick control, switch 31 can be flipped up for a print out on the terminal of the the x,y,z coordinate values of the present location of the hand. This information can be used by the operator for future entry of locations in terms of their X, Y, Z coordinate values. Once the arm has been positioned at the location PALLET(01) using joystick control, switch 32 is flipped down. This causes the system to record this position as PALLET(01).

The remaining five positions in the array are computed from the position of this location and the array dimension data supplied above.

As mentioned earlier in the report, it is with the recording and programming of arrays of locations that large savings in time can be obtained.

3.4 Approach Path

The next request from the location module is:

ENTER DELTA X,Y,Z (DECIMAL CM) TO START OF APPROACH PATH

This data is required for the specification of the approach path to be used for this location. The approach path concept simplifies much of the trajectory calculation. It is illustrated in Figure 14. Location "PALLET(01)" has the coordinate values X_L, Y_L, Z_L. The direction in

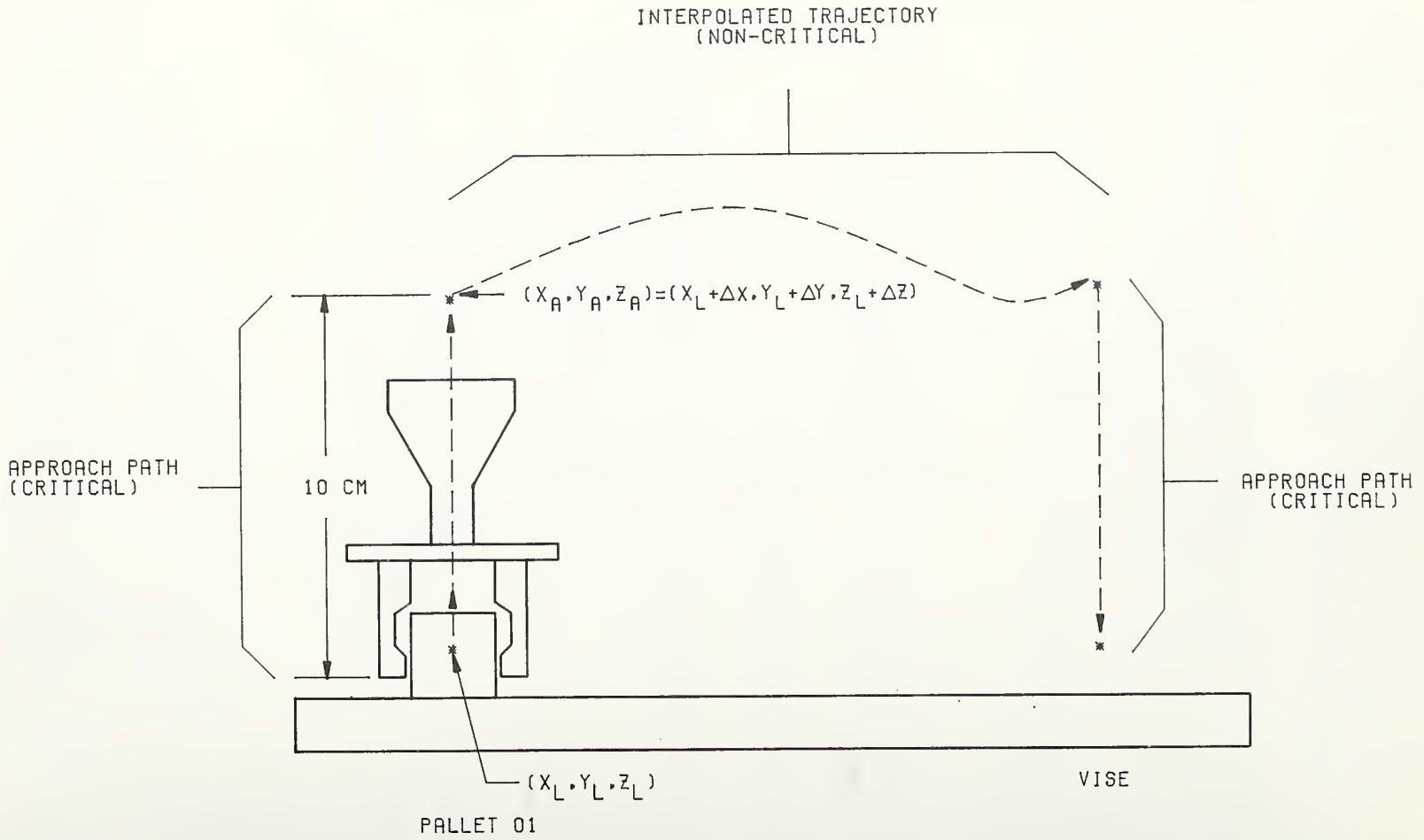


Figure 14 - The control system partitions the trajectories into 3 segments - 2 approach paths and an interpolated trajectory. The end points are recorded and stored in terms of both X, Y, Z coordinate values (as X_L, Y_L, Z_L) and joint coordinate values. The approach path point is entered by the operator as delta X, Y, Z coordinate offset values (ΔX, ΔY, ΔZ) from the end point.

which the hand approaches this point is critical. If the object shown is sitting on top of a table, then it would be disastrous to approach it from below.

The operator designates the delta X, delta Y, delta Z offsets from the location point to the start of the desired approach path (for the example in Figure 14 delta X=0, delta Y=0 delta Z=10cm). These values are stored in the location table. During execution, the hand moves quickly through space to the start of the approach path. It decelerates at this point and follows a straight line path from here to the actual location point. During this approach path maneuver, the orientation of the hand in space is maintained.

The non-critical portion of the trajectory, from one approach path to the other, can be executed at high speed by simple interpolation between all the joint values without regard for close tolerances on this motion. The use of an approach path, therefore, modularizes the trajectory into segments that can be easily executed by simple subroutines.

The operator's response to the above request for the delta offset values would be:

```
% 0,0,10.
*** TYPE +1 *** IF HAND WILL STOP AT THIS LOCATION
*** TYPE [CR] *** IF HAND DOES NOT STOP HERE.
```

This information is requested to provide a flag for each location. If the flag indicates that the hand is to stop at that location then an acceleration profile is created for the trajectory to this point. If the flag indicates that the hand does not stop at that location, then no acceleration profile is created and the trajectory passes through that point at the velocity specified in the elemental move command.

This completes the data required to totally describe a location. The operator is now given the opportunity of entering another location point or displaying the location table itself on the terminal for his inspection:

```
*** TYPE[CR] *** FOR ANOTHER POINT
*** TYPE +1 *** TO DISPLAY TABLE
```

3.5 Exit Location Module

To exit from the teach-location portion of the control system, switch 29 is flipped down and a carriage return is entered.

III. CONTROL SYSTEM IMPLEMENTATION

III. CONTROL SYSTEM IMPLEMENTATION

III. CONTROL SYSTEM IMPLEMENTATION

This chapter will concern itself with the manner in which information is processed by the subroutines to accomplish all of the responsibilities of the control system. Included here will also be an example of the procedure used to incorporate a new subroutine into the control system. The system documented here, is presently implemented on a 16 bit word length minicomputer, running under DOS (Disc Operating System), with 32K words of core and a 1.2 million byte disk storage system. All of the subroutines (except the I/O drivers) have been written in FORTRAN, compiled on this computer and their object modules linked to form a single run-time load module. If a prerecorded robot application program and table of location points are to be used, then these two files are assigned to the load module at run time. These files are read in from the disc by the subroutine RDMOD.

The load module contains all of the control system modules - the program module, the location module and the control system hierarchy including all of the support subroutines.

The operator communicates with the control system through a CRT terminal, the previously described switch panel and the joystick box.

1. CONTROL SYSTEM OPERATION

The architecture of this control system is based on a non-intelligent, deterministic system. Essentially, all data that is an input to the control system (control flags, location table pointers, sensory data, etc.) is treated as a form of a pointer (or address) to a file location, subroutine call, or section of code. The control system has been partitioned into different control levels which have been further partitioned into simple general purpose subroutines. A high level command input is in the form of pointers and flags which cause control to branch to the appropriate sequences of subroutine calls. Incoming sensory feedback data is typically tested against a threshold level. The section of code to be executed is determined by whether the sensory data is above or below the threshold value. Thus, this hierarchical system executes real time control over the robot by testing and branching to sequences of simple subroutines.

1.1 Control Hierarchy (Module #1)

The control hierarchy receives input commands as individual lines (elemental moves) from the program table. Each line is an input to the third level ("EMOVE") to generate the correct sequence of calls to the second (primitive) level (see Figure 15 for a diagram of the sub-routines in the hierarchy). The pointers to the location table address the proper position values of the end points of the trajectories.

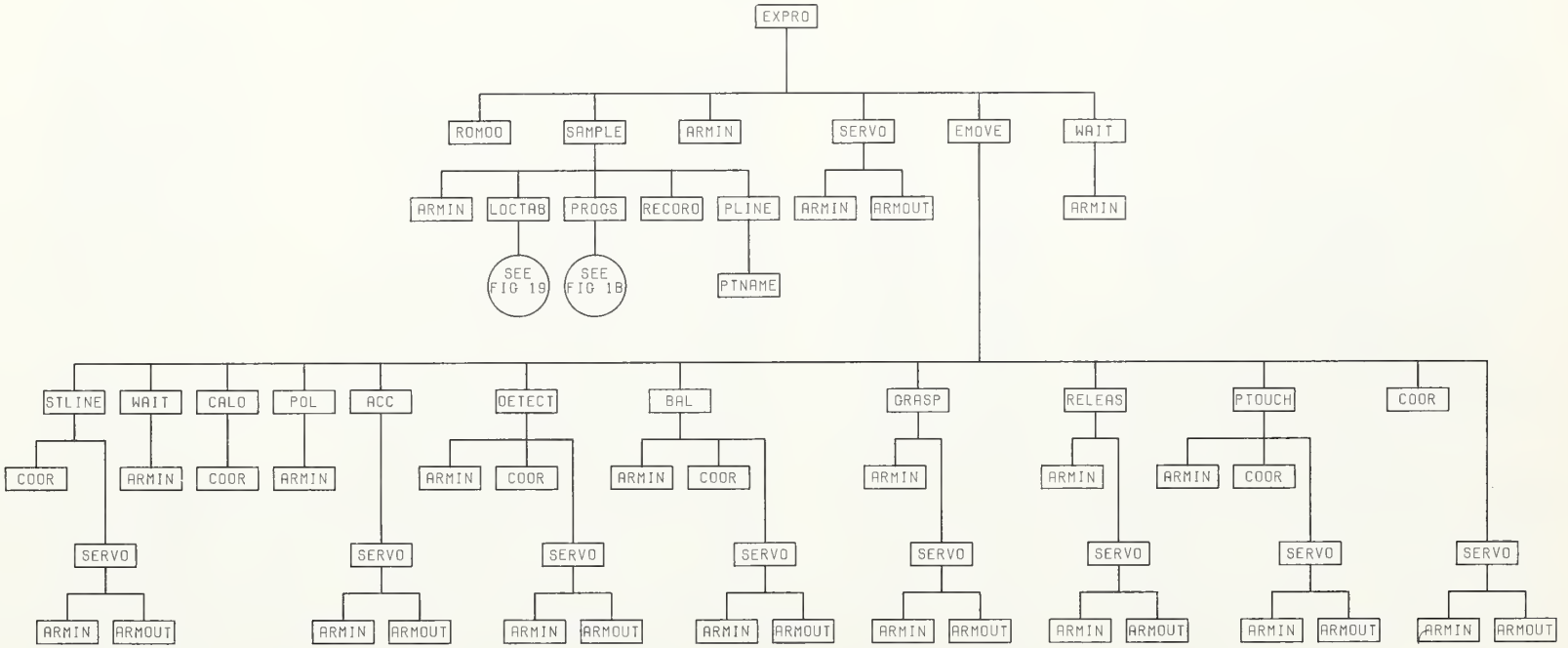


Figure 15 - The schematic representation of the interconnection of the control system subroutines. EXPRO is the executing program that monitors the switchboard for commands from the operator to call up the program module (PROGS),location module (LOCTAB) or to begin executing the robot program in the control hierarchy. The third level of the hierarchy (EMOVE) calls the different primitives of the second level which output to the servos through a call to the subroutine SERVO. All input and output between the control system and the external world occur through the drivers ARMIN and ARMOUT.

Trajectories are modularized into 3 path segments (Figure 14), a critical path segment at each end of the trajectory (the approach paths) and a non-critical segment that is the portion of the trajectory between the two approach paths. The approach paths are executed at a relatively slow velocity with accurate control of the path and orientation of the hand. This is accomplished through a call to "STLINE." The non-critical middle section of the trajectory is executed at relatively high velocity with simple liner interpolation between the joint values that specify the end points of the two paths. A call to "CALD" determines the distance between these two end points. This value is used by "POL" and "ACC" to generate the joint position commands to move the arm through this trajectory segment at the designated velocity.

Sensory feedback for real time control is obtained through calls to "ARMIN" to read in the values from the A/D converter. The subroutines that process this information use simple algorithms to generate new position commands, in many instances requiring coordinate transformation (a call to "COOR") to command motion in joint coordinate space in accord with feedback in sensor coordinate space. These joint position commands are sent to the arm by a call to "SERVO" which calls the driver "ARMOUT" to output these values to the D/A interface which is connected to the hardware servo system.

1.2 Program Module (Module #2)

When an elemental move command is entered by the operator, the programming module codes these English-word-like statements into a number of pointers and flags (see Figure 16). All of the subroutines contained in the programming module are used to generate or edit this table of pointers and flags which will form the input commands to the third level of the control hierarchy.

The relationships between the subroutines of the programming modules is shown in block diagram form in Figure 17.

The detailed workings of these subroutines are covered by the next chapter where the functional flow charts and the documented FORTRAN code is listed.

The functional activities of the programming routines can be summarized in the following manner. Editor commands (INSERT, DELETE, PRINT, AVOID, RECORD, LOOP, NAME) are decoded by a test on the ASCII value of their first letter by the subroutine PROGS. This determines which of the editor subroutines is to be called. Other subroutines (LIMIT, NEXT) search the operator entered commands for the key information symbols. The subroutines "LINE" and "LOCPT" code the English-like elemental move commands into the proper location table pointers and function flags. The subroutines "PLINE" and "PTNAME" decode these pointers and flags into the English-like elemental move commands and print them on the terminal.

i^{th} line in program table

1	2	3	4	5	6
Pointer for present location	Pointer for destination	Velocity indicator in cm/sec	Interlock channel numbers	Optional, not presently used	Function code number

Specific example for the elemental move command GOTO PALLET (03) GRASP PROX VELOCITY(15) SEND(03) WAIT(10). The line in the program table would look like (assuming the present position was neutral):

74	03	15	310*		3

The following is the corresponding function code number for the presently implemented functions.

Function	Code Number
grasp	1
release	2
grasp proximity	3
release proximity	4
detect	5
balance	6
unstack	7
touch	8
line	9

*The output channel number is multiplied by 100 and added to the input channel number.

Figure 16 - The program table representation of an elemental move command.

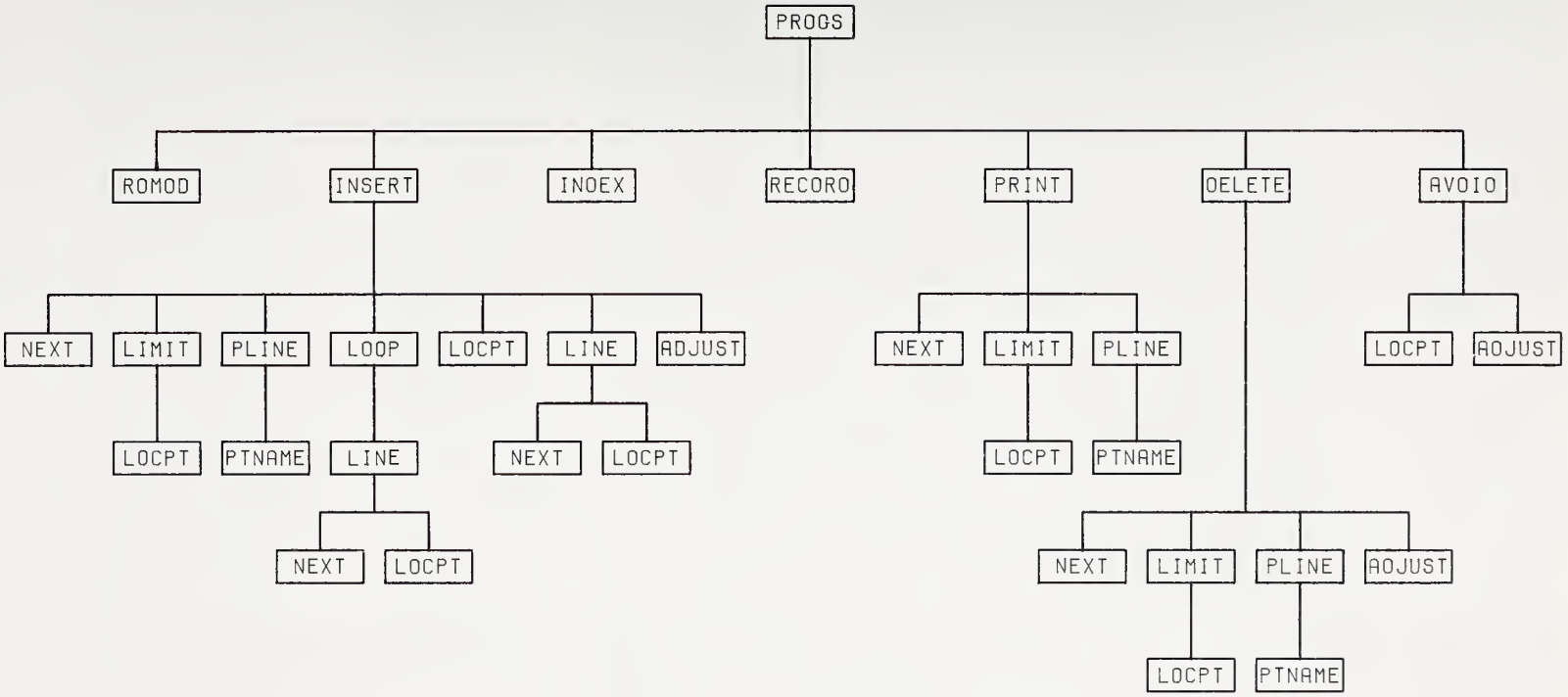


Figure 17 - The schematic representation of the interconnections between the subroutines of the program module. PROGS monitors the operator's input commands and branches to the appropriate subroutine call to execute these commands.

The output from the program module (in the form of a program table) becomes the sequence of commands to the third level of the hierarchical control system. Each line in this table corresponds to an elemental move command and contains six positions for pointers and flags (Figure 16).

1.3 Location Module (Module #3)

The subroutines of the location module perform the function of generating and storing the required information for each location point so it can be used by the control system in the execution of the elemental move commands. A block diagram of these subroutines is shown in Figure 18. Translation of the location name into the correct pointer value is accomplished by "LOCPT." When arrays of locations are to be entered, "ARRAY" requests the necessary dimensional information and "ARRLOC" generates the required values for each location of the array. When joystick control is to be used, "LOCTAB" makes a call to "JOY" which reads in the present joint position values of the arm by a call to "POS." "JOY" also reads in the commanded x,y,z offsets from the joystick by a call to the assembly language input driver, "ARMIN" and generates the new joint position commands by a call to "COOR" to perform the coordinate transformations. These joint position commands are sent to the arm by a call to "SERVO" which outputs these values to the hardware servos by a call to the assembly language output driver "ARMOUT." In the present system, the location table can store information on 86 different locations. Indexed location names are assigned pointer values of one to 60 (three indexed names are allowed - each containing twenty locations). One arbitrary name for each letter of the alphabet is allowed and assigned pointer values 61 through 86. The last two pointer locations (89 and 90) do not store position information. Instead, the ASCII code for the complete names of the indexed locations reside in these positions.

A pointer to the location table designates a group of four sequential lines of seven positions each. The information stored in these four lines is detailed in Figure 19. Two of the lines are particular for the specific robot executing the application program. They contain the joint indicator values that characterize the position of the arm for that location point.

The other two lines contain information that describe the location in an external X, Y, Z coordinate system. This information is robot-independent in that it can be post-processed by another robot's coordinate transformation routine into joint position values specific for that robot. Therefore, the table of location points, stored in terms of this external coordinate system, can reside in a central control computer to be post processed by the specific coordinate transformation routine of the robot that is to use this location table.

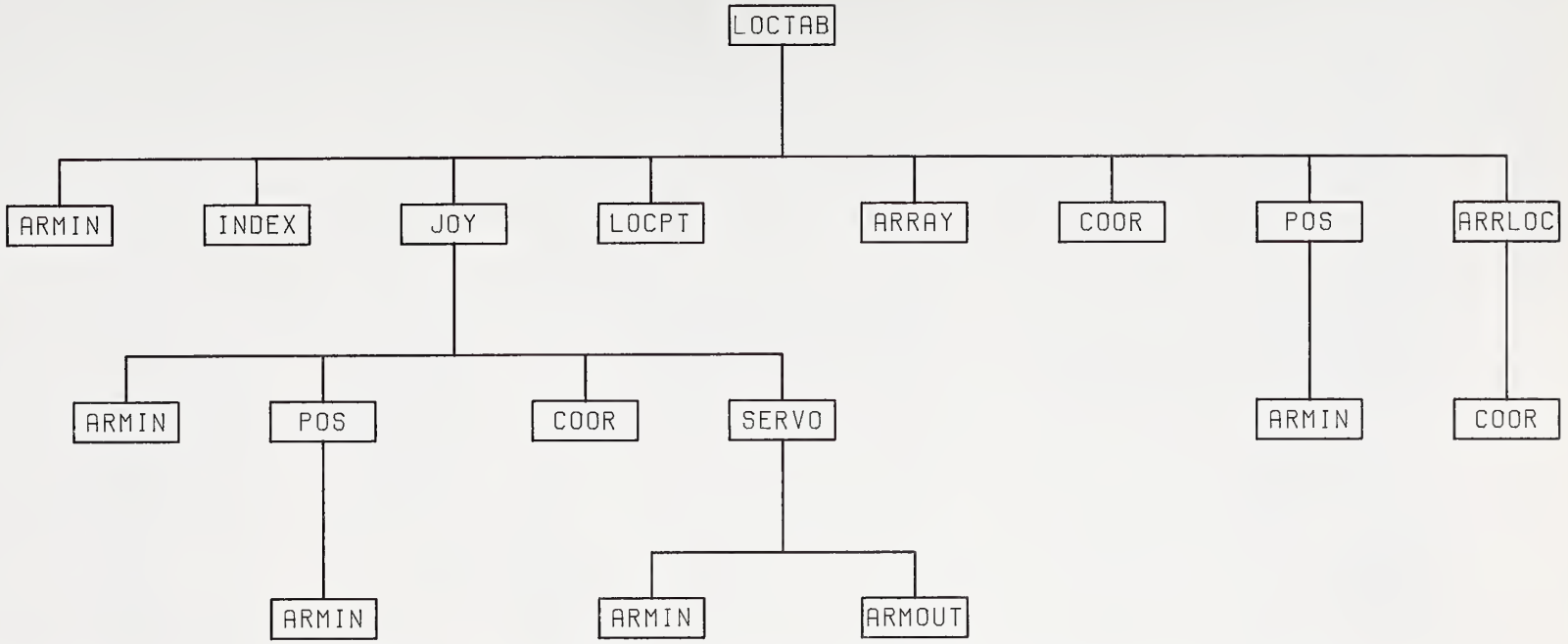


Figure 18 - The schematic representation of the interconnections between the subroutines of the location module. LOCTAB monitors the operator's input commands and branches to appropriate subroutine calls.

Pointer XX locates a set of four - 7 element-lines in the location table (LTAB(XX,1-4,1-7)) to completely describe a single location point.

Line 1	$J1_{LP}$	$J2_{LP}$	$J3_{LP}$	$J4_{LP}$	$J5_{LP}$	$J6_{LP}$	
Line 2	X_{LP}	Y_{LP}	Z_{LP}	i	j	k	STOP FLAG
Line 3	i_h	j_h	k_h	ΔX	ΔY	ΔZ	
Line 4	$J1_{APP}$	$J2_{APP}$	$J3_{APP}$	$J4_{APP}$	$J5_{APP}$	$J6_{APP}$	

Line 1 and line 4 contain the position values of the six joints to describe the location point and the starting point of the approach path. These two lines will contain values unique to a particular robot's position servos.

The other 2 lines (line 2 and line 3) contain the description of the location point in an external coordinate system system. Line 2 contains a description of the location point in the same format as used by the CLDATA file to store location values. These are the coordinates of (X_L, Y_L, Z_L,) the location point and the i, j, k unit vectors to describe the direction of the hand-wrist axis (surface normal). This provides a 5 axis description. The 6th axis, the direction of a vector perpendicular to the hand wrist axis and passing through the two finger tips (the hand rotation) is described by the unit vectors i_h, j_h, k_h in line 3. The delta X, Y, Z distances (ΔX, ΔY, ΔZ) from the location point to the start of the approach path are also contained in line 3. The stop flag, to indicate if the hand comes to a stop at this location (used to decide on acceleration-deceleration profiles) is stored in the seventh position of line 2.

Pointer Value	Name Description
1 - 20	20 pointers allocated for the first indexed location name
21 - 40	20 pointers allocated for the second indexed location name
41 - 60	20 pointers allocated for the third indexed location name
61 - 86	26 pointers allocated for 26 names each starting with a different letter of the alphabet.

Figure 19 - The Location Table representation of a location point.

2. IMPLEMENTING A NEW FUNCTION

2.1 Subroutine Design

Functions, specified by elemental move commands entered in the program module, are executed by branches to the appropriate subroutine (or sequence of subroutines). Therefore, a new function, such as implementing a new operation using sensor feedback, requires the addition of the user supplied subroutine(s) to accomplish it. In addition, modifications have to be made to two subroutines (LINE, PLINE) in the program module and one subroutine (EMOVE) in the control hierarchy module.

The approach in developing these subroutines that are the primitive functions has been to make them as general as possible. This allows them to be more easily used in sequences to perform other operations as well. To this end, it is also advisable to limit their scope to a very simple action. Again, using sequences of these simple actions to perform more complex actions.

As an example, consider a user requirement to insert a part into a vise where the part is to be pushed against a bottom plate within the vise to properly seat it. A subroutine to accomplish this seating action using a force sensor in the wrist would be written by the user. It might employ a simple algorithm such as:

(a) read in the sensor value
(b) if its above a predetermined threshold value go to (e) otherwise continue
(c) move a defined incremental distance in the -z direction
(d) go to (a)
(e) exit subroutine.

This subroutine might be called FORCE. It performs a very limited, simple action. This action in itself is insufficient to accomplish the actual operation (which will be called INSERT). The primitive (RELEAS) to cause the hand to release the part must also be called. This releasing action could have been written into the primitive FORCE but this would have increased the complexity of its actions and thereby narrowed its range of application to other operations.

2.2 Control Hierarchy Modification

To implement this new subroutine requires a modification in the control hierarchy's third level (EMOVE) to test for its function code number in the elemental move command input and to branch to its calling statement. Figure 16 shows that code numbers 1 through 9 are presently being used. Therefore, this INSERT operation could be assigned number 10. The value of the function code number is used to control two multiple "GOTO" statements in EMOVE. In this way, the function number acts much as an address to the location of code to execute it. Figure 20 shows how the relevant code is to be modified for this example.

EMOVE before modifications (changes handwritten in)

```
C
        FUN=1+ELE(6)                                          2 20
        GOTO(20,20,20,3,4,20,20,7,8,20)FUN
C
C   IF THERE IS AN APPROACH PATH, IT IS GENERATED AND EXECUTED BY
C       THIS CALL TO 'STLINE'.
C
20      CALL STLINE(-LTAB(ELE(2),3,4),-LTAB(ELE(2),3,5),-LTAB(ELE(2),3,6),1)
        IF(EABORT.LT.-2000)GOTO 40
        CALL WAIT(34)                                    10
C
C   THIS MULTIPLE 'GOTO' STATEMENT CAUSES THE BRANCHING TO THE APPROPRIATE
C       CODE TO CARRY OUT THE FUNCTION SPECIFIED.     10  Call Force
C                                                         Call Release (0)
        GOTO(110,1,2,3,4,5,6,7,8,110)FUN                  Call Stline(-off(1),-off(2),-off(3),1)
C                                                         GOTO 435
C   THIS IS THE CODE FOR A 'GRASP PROXIMITY' FUNCTION.  THE HAND IS
```

EMOVE after modifications

```
C
        FUN=1+ELE(6)
        GOTO(20,20,20,3,4,20,20,7,8,20,20)FUN
C
C   IF THERE IS AN APPROACH PATH, IT IS GENERATED AND EXECUTED BY
C       THIS CALL TO 'STLINE'.
C
20      CALL STLINE(-LTAB(ELE(2),3,4),-LTAB(ELE(2),3,5),-LTAB(ELE(2),3,6),1)
        IF(EABORT.LT.-2000)GOTO 40
        CALL WAIT(34)
C
C   THIS MULTIPLE 'GOTO' STATEMENT CAUSES THE BRANCHING TO THE APPROPRIATE
C       CODE TO CARRY OUT THE FUNCTION SPECIFIED.
C
        GOTO(110,1,2,3,4,5,6,7,8,110,10)FUN
10      CALL FORCE
        CALL RELEAS(0)
        CALL STLINE(-OFF(1),-OFF(2),-OFF(3),1)
        GOTO 435
C
C   THIS IS THE CODE FOR A 'GRASP PROXIMITY' FUNCTION.  THE HAND IS
```

Figure 20 (a) - Shows the section of code in "EMOVE" before and after the modifications required for the addition of a new function. In this example, the new function code number (10) plus one gives the value of the variable "FUN" used to specify the section of code to be executed through a multiple "GOTO" statement. The code required to execute the "INSERT" function is, first, a call to the user written "FORCE" subroutine to seat the part, a call to "RELEAS" to let go of the part, a call to "STLINE" to move back the distance covered during the "FORCE" subroutine and to branch to statement 435 to test for interlock signals.

LINE before modifications

```
C
C    'TOUCH'
C
64       IF(CS(PN),NE,8276)GOTO 65
         PROG(PLN,6)=8
         GOTO 81
C
C    'LINE'
C
65       IF(CS(PN),NE,8268)GOTO 85
         PROG(PLN,6)=9
         GOTO 81
C
C    TEST FOR A 'VELOCITY' COMMAND, IF THERE IS ONE THEN DECODE THE
C        ASCII FORM OF THE SPECIFIED VELOCITY (CM/SEC) BY A
```

Change to 66

ASCII Code for the letter "I".

66 IF(CS(PN).NE. 8265) GOTO 85
PROG (PLN, 6) = 10
GOTO 81

LINE after modifications

```
C
C    'TOUCH'
C
64       IF(CS(PN),NE,8276)GOTO 65
         PROG(PLN,6)=8
         GOTO 81
C
C    'LINE'
C
65       IF(CS(PN),NE,8268)GOTO 66
         PROG(PLN,6)=9
         GOTO 81
C
C    'INSERT'
C
66       IF(CS(PN),NE,8265)GOTO 85
         PROG(PLN,6)=10
         GOTO 81
C
C    TEST FOR A 'VELOCITY' COMMAND, IF THERE IS ONE THEN DECODE THE
C        ASCII FORM OF THE SPECIFIED VELOCITY (CM/SEC) BY A
```

Figure 20 (b) - Shows section of code in "LINE" before and after modifications to code and store the new function command in program table. The additional code is a test for the first letter ("I") of the new function command (INSERT). If an "I" is detected then the appropriate function code number (here, it is a 10) is stored in the sixth position of the present line of the program table.

PLINE before modifications

```
      COMMON/CMD/CS(50),PN,PLN
      DIMENSION FN(6,10)
      DATA FN/6*'    ',
     2 'GR','AS','P ',3*'    ',
     3 'RE','LE','AS','E ',2*'    ',
     4 'GR','AS','P ','PR','OX','    ',
     5 'RE','LE','AS','E ','PR','OX',
     6 'DE','TE','CT',3*'    ',
     7 'BA','LA','NC','E ',2*'    ',
     8 'UN','ST','AC','K ',2*'    ',
     9 'TO','UC','H ',3*'    ',
     1 'LI','NE',4*'    '/
C
C    IF THE FIRST LINE OF THE PROGRAM MODULE IS TO BE PRINTED OUT,
```

Change to 11

Change to a comma

' ','IN','SE','RT',3*'   '/

PLINE after modifications

```
      COMMON/CMD/CS(50),PN,PLN
      DIMENSION FN(6,11)
      DATA FN/6*'    ',
     2 'GR','AS','P ',3*'    ',
     3 'RE','LE','AS','E ',2*'    ',
     4 'GR','AS','P ','PR','OX','    ',
     5 'RE','LE','AS','E ','PR','OX',
     6 'DE','TE','CT',3*'    ',
     7 'BA','LA','NC','E ',2*'    ',
     8 'UN','ST','AC','K ',2*'    ',
     9 'TO','UC','H ',3*'    ',
     1 'LI','NE',4*'    ',
     1 'IN','SE','RT',3*'    '/
C
C    IF THE FIRST LINE OF THE PROGRAM MODULE IS TO BE PRINTED OUT,
```

Figure 20 (c) - Shows section of code in "PLINE" before and after modifications to cause a print out of the new function name. The array that contains all of the possible function names is enlarged by one line and the letters to be printed out are stored.

2.3 Program Module Modification

Two subroutines (LINE, PLINE) in the program module are to be altered so that the new function command, INSERT, can be recognized and coded into its function number (LINE), and decoded and printed on the terminal (PLINE) by the editor PRINT command. The modifications for this example are illustrated in Figure 20.

3. PORTABILITY

The partitioning of the entire control system into a number of modules with well-defined interfaces to the rest of the system results in a large number of the modules being independent of both the robot they are controlling and the computer system they are running on. Those instances where the control system must interface to either the host computer system or to a particular robot can be contained in a limited number of modules. These modules have been identified and it is only these modules that require alteration for portability of the control system (Table 6).

3.1 Input Channels

As mentioned before, the operator communicates with the control system through a terminal, a switch panel, and a joystick control box. The values of the switches and the joystick control box are read into an input buffer through a call to the assembly language driver ARMIN. Thus subroutine reads in 64 channels through a 14 bit A/D (Analog to Digital) converter and stores these values in the common variable INBUF(64). This call takes approximately 4 milliseconds to execute (the maximum conversion frequency of the A/D unit is 20 kilohertz). Table 7 provides a summary of the functions of the 64 input channels.

3.1.1 Function Switch Panel

The switch panel consists of eight switches which can be set to +5.0 volts or reset to 0.0 volts. These values are read into the computer through the D/A converter with a value -4000 corresponding to a +5.0 volts and zero corresponding to 0.0 volts.

3.1.2 Joystick Control Box

The joystick control box uses a joystick in two dimensions actuating two potentiometers to provide the delta offsets in the X and Y direction. The Z motion is provided by two buttons, one for up and one for down. Push buttons also provide control of the wrist roll, wrist flex, hand roll and opening and closing of the fingers. A detailed listing is provided in Table 7.

3.1.3 Joint Position Indicator Input

The joint position indicators are also read in through the A/D converter and range from +10 to -10 volts resulting in values of -8383 to +8383 into the computer.

3.2 Output Channels

A call to the output driver ARMOUT will cause the values in the first 16 locations in the common variable OUTBUF(64) to be sent to the interface. The system allows 64 output channels but only 16 are presently implemented.

Table 6

Non-Portable Functions

These are the subroutines whose functions are tailored to either the host computer's system or the particular robot, and must be modified to allow the control system to be implemented with a different computer or robot.

Subroutine Name	Function
RDMOD	Used to interface control system to host computer's file accessing method. Causes the previously recorded location table and program table to be read from host computer's mass storage into the control system's programs.
RECORD	Causes the program table and location table to be written into the host computer's mass storage system.
ARMIN	An assembly language input driver to read the input values from the system interface and store in a input buffer to be accessed by the control system subroutines.
ARMOUT	An assembly language output driver that sends the values stored in an output buffer to the system interface.
COOR	This is the coordinate transformation routine that allows conversion of X, Y, Z coordinate values into the corresponding joint coordinate values and vice versa for a particular robot. As provided here, only a three axis (two shoulder rotational and a boom prismatic joint) coordinate transformation is done, with the remaining 3 wrist joints pantographing the shoulder moves. This was all that was required for the demonstration using workpieces on a bench top. However, the complete 6 axis coordinate transformation routine is being implemented and is required for complete general purpose programming of the robot.

Table 7

Listing of the input channels read into the command variable "INBUF(64)" by the call to ARMIN.

INPUT CHANNEL NUMBER	FUNCTION	INPUT VOLTAGE RANGE	CORRESPONDING COMPUTER VALUES
1			
2			
3			
4			
5	joint #6 position	-10 to +10	8191 to -8191
6			
7			
8	joint #2 position	-10 to +10	8191 to -8191
9			
10			
11	joint #3 position	-10 to +10	8191 to -8191
12			
13	joint #4 position	-10 to +10	8191 to -8191
14			
15	joint #5 position	-10 to +10	8191 to -8191
16			
17	joint #1 position	-10 to +10	8191 to -8191
18			
19	joint #7 position	-10 to +10	8191 to -8191
20			
21			
22			
23			
24			
25			
26*	trajectory velocity control	2.7 to 4.0	-2160 to -3160
27*	finger grasp-release	0, 4.6	0, -3700
28**	interrupt-abort	0, 4.6	0, -3700
29**	enter location	0, 4.6	0, -3700
30**	edit program	0, 4.6	0, -3700
31**	store on disc, or display XYZ values	0, 4.6	0, -3700
32**	joystick control	0, 4.6	0, -3700
33**			
34**	pause, or enter location with XYZ values	0, 4.6	0, -3700
35**	start, repeat	0, 4.6	0, -3700

Table 7 cont.

INPUT CHANNEL NUMBER	FUNCTION	INPUT VOLTAGE RANGE	CORRESPONDING COMPUTER VALUES
36**	straight line velocity control	0, 4.6	0, -3700
37*	wrist roll	0, 4.6	0, -3700
38*	wrist flex	0, 4.6	0, -3700
39*	hand roll	0, 4.6	0, -3700
40*	X motion	.9 to 1.9	-730 to -1450
41*	Y motion	.9 to 1.9	-730 to -1450
42*	Z motion	0, 4.6	0, -3700
43			
44			
45			
46			
47			
48	proximity sensor	0 to -10v	0 to 8191
49	proximity sensor	0 to -10v	0 to 8191
50			
.			
.	not used		
.			
.			
64			

* Joystick Box Input Channels
** Switch Panel Input Channels

3.2.1 Joint Position Command Outputs

The position command output values to the six joints and the fingers of the robot are sent on the first seven output channels. These channels in the interface unit (see Appendix A for a detailed description of the interface unit) are connected to a digital to analog converter to provide analog signals to the hardware servo system. Output values from the computer range from 0 to 32767 corresponding to the range of +10 volts to -10 volts respectively (with 16383 corresponding to zero volts).

3.2.2 Brake Control Output

Output channel number 8 is used as a digital output for individual control of the seven sets of brakes (one for each joint and one for the fingers). A fifteen bit word is sent out on channel 8 with the first seven bits used as switch values for setting the brakes. With all of the bits equal to one (i.e. the value of the word's 32767), the brakes are released. To set the brake on the fingers, the seventh bit is set to zero. This requires the number 32703 (i.e. 32767-64) to be sent.

3.2.3 Interlock Channels

There are eight additional output channels (numbers 9 thru 16) that can be used either as digital or analog outputs to control other devices or send interlocking signals on. The "SEND" command presently outputs a value(8192) on the specified channel number that results in an analog signal level of +5 volts.

4. ADVANTAGES OF HIERARCHICAL CONTROL SYSTEM

A summary listing of the advantages afforded by the hierarchical control system architecture described in this report is given in Table 8.

Table 8

Summary of Control System

(1) The system has been modularized, resulting in:

(a) easily defineable functional specifications, interfaces, and communications requirements;

(b) the defining and isolation of those parts of the control system that are specific to the host computer and particular robot;

(c) relative independence of the robot programs and their standard coordinate location tables from the particular robot, thereby allowing off-line programming as well as location table generating from other data bases such as CLDATA files;

(d) a control system compatable with the addition or deletion of extra functions, sensors etc. (accomplished with addition or deletion of subroutines along with minor modifications to the existing system);

(e) quick and simple control system code generation and debugging.

(2) The system has been structured as a deterministic hierarchy of functional levels resulting in:

(a) sophisticated programming techniques using simple higher level language commands to ease and speed program entry, where the hierarchy provides amplification of a single command into the large sequences of detailed steps required to accomplish it;

(b) minimal information processing and transfer within and between levels due to the deterministic feature where only a limited set of states are allowed; the hierarchy that provides a fast and efficient method of chosing between these states based on simple tests of input information flags;

(c) the ability for real time sensor control of a multiaxis robot on a small computing system; this is due to the simplicity of information processing (described above) which requires a relatively small number of computations allowing the trajectories to be modified in real time in accord with the sensory feedback.

(3) The control system has been written in FORTRAN IV resulting in:

(a) relative portability of the system to other host computers (where the computer-dependent modules have been identified);

(b) an ease in understanding of the code due to the simplicity of FORTRAN and its universal use;

(c) the ability of the user to add additional features easily by writing his own FORTRAN subroutines;

(d) the ability to implement the control system in assembly language on microprocessor or minicomputers since it is a rather straight forward task to cross code FORTRAN statements into assembly language.

REFERENCES

1. R. Finkel, et. al., AL, A Programming System for Automation, Stanford Artificial Intelligence Laboratory Memo. AIM-243, STAN-CS-74-456, Stanford University, November, 1974.

2. P. M. Will and D. D. Grossman, An Experimental System for Computer Controlled Mechanical Assembly, IEEE Trans. on Computers, Vol. C-24, No. 9, 879-888, September, 1975.

3. J. Nevins, et. al., Exploratory Research in Industrial Modular Assembly, C. S. Draper Lab Report No. R-921, August, 1975.

4. C. Rosen, et. al., Exploratory Research in Advanced Automation, First Semi-Annual Report, SRI Project 2591, Stanford Research Institute, Menlo Park, California, December, 1973.

5. R. M. Spencer, High Level Control of Reprogrammable Automatic Assembly Machines, Master's Thesis, Electrical Engineering Department, Massachusetts Institute of Technology, 1975.

6. J. S. Albus, and J. M. Evans, Jr., Robot Systems, Scientific American, Vol. 234, No. 2, pp 76-86b, February, 1976.

7. J. S. Albus, Data Storage in the Cerebellar Model Articulation Controller (CMAC), Journal of Dynamic Systems, Measurement, and Control Trans. ASME, Series G, Vol. 97, No. 3, pp 228-233, September, 1975.

APPENDIX A

APPENDIX A

Interface Design

Data transfer between the laboratory hardware and the PDP-11/45 minicomputer is handled via a DR11-C interface card in the computer. This card has 16 data input lines, 16 data output lines, two interrupt lines and two sync lines, one (data transmitted) for input synchronization, and one (data present) for output synchronization.

The DR11-C is connected to the laboratory interface by forty parallel wires powered by 7406 line drivers. These are open collector circuits with resistors connected to the power supply voltage at the receiving end.

The laboratory interface hardware consists of two parts: an input section and an output section.

Input:

A sixty four channel analog multiplexer followed by a 14 bit analog to digital converter forms the heart of the input section. This unit has a range -10 volts to +10 volts. Negative voltages are converted to 2's complement form.

Output from the A/D converter is carried to the computer on the 16 data input lines. Since the A/D converter has only a 14 bit output, the most significant (sign) bit is connected to the three most significant computer input lines. This causes the computer to read the correct values for both positive and negative (2's complement) numbers.

Data transfer between the A/D converter and the computer is signaled by a pulse on the data transmitted line. This pulse, initiated by the computer, indicates that the data lines have been sampled. This data transmitted pulse is used to step the analog multiplexer to the next input and to initiate the analog to digital conversion cycle on the new input input voltage. The input software is timed to sample the input data at a rate slower than the conversion rate of the A/D system. Thus, a "conversion completed" signed is not required by the computer, and only one control line is needed to advance the converter to the next channel.

The multiplexer is operated in a sequential mode. It is reset to the zeroth channel by a reset pulse derived from the output section. Each data transmitted pulse steps the multiplexer to the next sequential channel. The input section is therefore completely under computer control.

Output:

A sixteen channel digital demultiplexer routes output signals from the DR11-C to sixteen holding registers. These provide either digital output directly, or are connected to digital to analog converter so as to provide analog voltages. Data transfer between the DR11-C and the output section is initiated by a pulse on the data present line. The data present pulse occurs simultaneously with the appearance of data on the DR11-C output lines. This pulse is delayed in the laboratory interface hardware for two microseconds before actuating the digital demultiplexer so as to allow the data lines time to settle. The delayed data present pulse gates the output data into the selected holding register and then steps the address counter to the next address.

The most significant (sign) bit of the output is decoded as a reset signal and is used to set the demultiplexer address counter to Channel 1 as well as to reset the analog multiplexer address in the input section.

Using the sign bit for control means that only 15 bits of output are available for data. Numerical output is restricted to the range 0 to 32,767. The digital to analog converters are adjusted such that numerica 0 corresponds to +10 volts; 16,384 corresponds to zero volts, and 32,767 corresponds to -10 volts.

Comments

This report describes an architecture for a control system. Its emphasis has been on the partitioning of the control system into a hierarchy of functional levels. This approach provides a way of thinking about the problem that has been very instructive to us. Of course, the decomposition of a problem or task into sequences of simpler actions is a very old technique. Indeed, it is an implicit fundamental method used by human beings in almost all things they do. So, in that sense, the use of a hierarchical design for a robot control system is hardly a novel idea.

However, instead of the hierarchy being implicit in our approach we have made it the explicit starting frame work. The benefits that we feel have derived from this are many and several have been discussed in the report.

The control system actually implemented in this report is a minimum configuration for a robot performing simple manipulative tasks. It is provided here as an example of how to construct a sensory interactive, goal directed control system when one uses a hierarchical framework to structure the problem. It also demonstrates the ease of greatly increasing control capabilities by the addition of higher levels in the hierarchy while retaining the already developed lower levels. Thus, the system is upwards compatible to almost any degree of complexity of behavior.

Increases in complexity of behavior will usually require increasingly complex sensory data processing such as might occur with force or low level vision. We view the sensory processing required occuring in an ascending hierarchy parallel to the descending control hierarchy with the exchange of differing degrees of processed information between these two hierarchies at all levels. Thus, the control hierarchy will continue to make simple tests on sensory feedback input and branch to appropriate programmed responses regardless of the complexity of the sensory data processing.

This sensory data processing will be occuring in the sensor feedback hierarchy not in the control hierarchy.

USE OF MICROPROCESSORS

One of the major points of this approach has been the partitioning of a sensory interactive control system into a number of functional ' levels. Implicit in this, is the idea that each of these function modules is an independent system, operating all the time, communicating to other control or sensory modules through standard interfaces anytime it needs to receive or send information. The use of a single computer severely restricts this type of system.

These modules can be made to appear to be independent and parallel through complex software manipulation such as real time multi-tasking operating systems on a serial computer. But even a fast computer soon runs out of time to perform all the functions as the requirements of the control system increase.

The obvious solution is for each level of the hierarchies to reside in an inexpensive microprocessor. Information to be exchanged would be placed in common memory. Several levels might be combined into one microprocessor depending on the simplicity of and number of functions to be handled. A separate microprocessor could be allocated for the coordinate transformation routine and one for safety and error checking routines. Future development work will address the use of microprocessors in these hierarchical network structures.

PROGRAMMING STRATEGY

This report presents not only a philosophical approach to the design of a control system but also the documented FORTRAN programs that implement a minimum configuration sensory interactive control system. This has been done to provide both a concrete example of a hierarchical control system and a starting point for anyone wishing to pursue robot control system development.

Both the programs and techniques of data handling can be improved and will be. The programming approach has been to make the system as table driven as possible, to make the applications programs as totally separate from the data as possible. We are updating the system in an attempt to realize these goals.

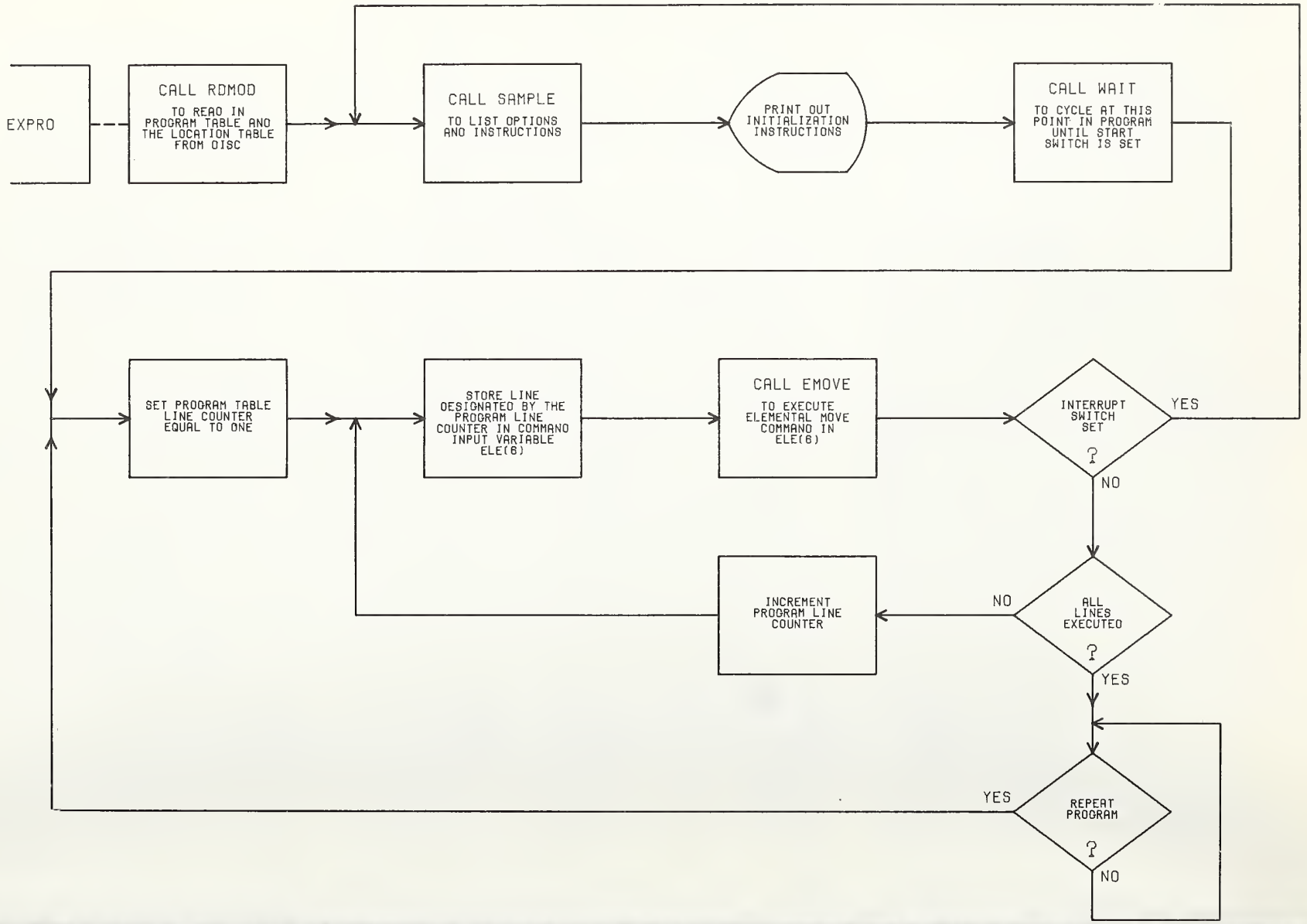
As new levels, functions, and sensors are added and programming techniques are improved, additional reports will be written and distributed.

Control Hierarchy
(Module #1)

3rd Level of Control Hierarchy

2nd Level of Control Hierarchy

1st Level of Control Hierarchy

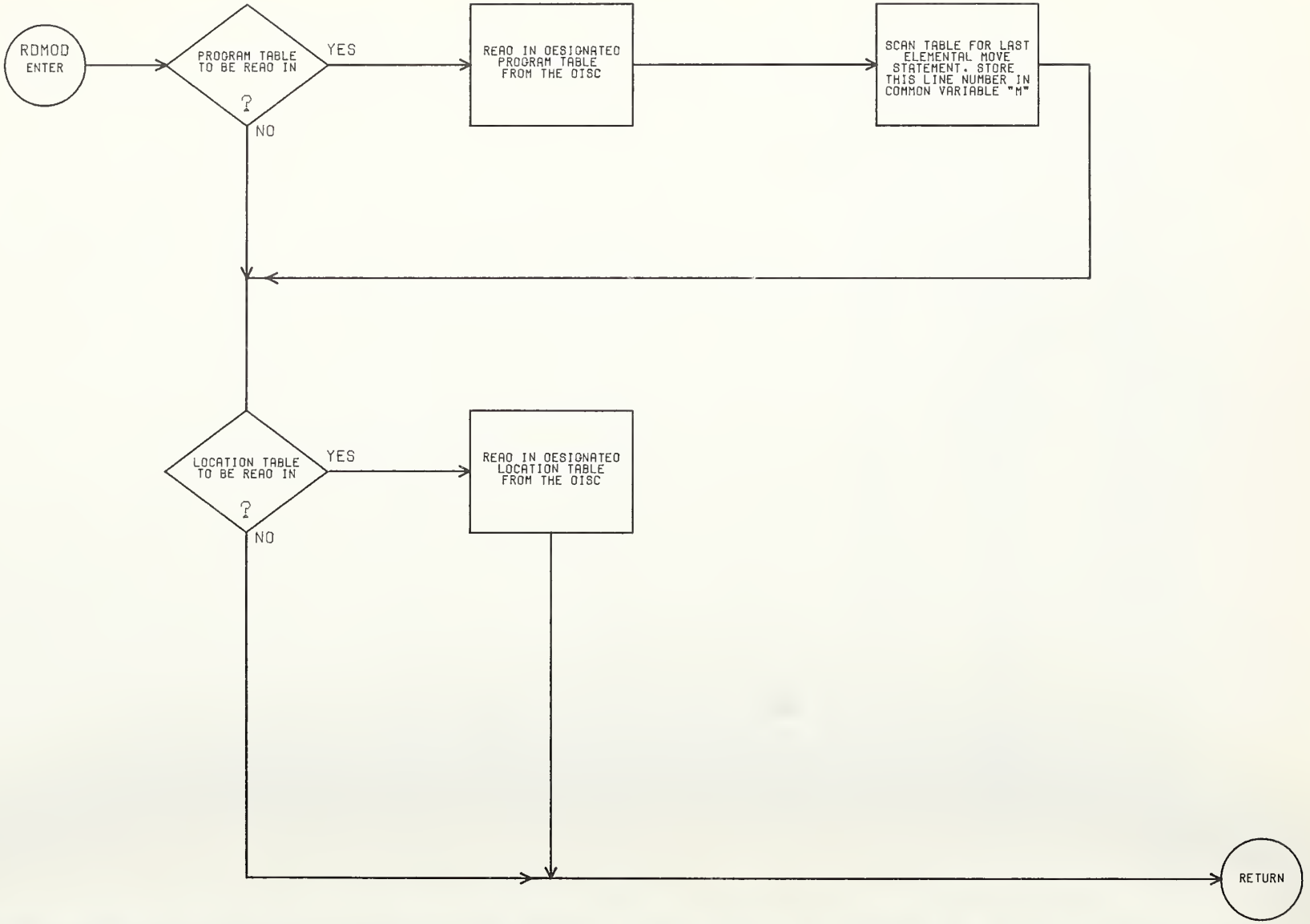

EXPRO
CALL RDMOD
TO READ IN PROGRAM TABLE AND THE LOCATION TABLE FROM DISC
CALL SAMPLE
TO LIST OPTIONS AND INSTRUCTIONS
PRINT OUT INITIALIZATION INSTRUCTIONS
CALL WAIT
TO CYCLE AT THIS POINT IN PROGRAM UNTIL START SWITCH IS SET
SET PROGRAM TABLE LINE COUNTER EQUAL TO ONE
STORE LINE DESIGNATED BY THE PROGRAM LINE COUNTER IN COMMAND INPUT VARIABLE ELE(6)
CALL EMOVE
TO EXECUTE ELEMENTAL MOVE COMMAND IN ELE(6)
INTERRUPT SWITCH SET ?
YES
NO
ALL LINES EXECUTED ?
NO
INCREMENT PROGRAM LINE COUNTER
YES
REPEAT PROGRAM ?
YES
NO

```
C --- PROGRAM : EXPRO
C
C --- ARGUMENTS :
C
C --- CALLED BY :
C
C --- CALLS SUBROUTINES : RDMOD  SAMPLE  ARMIN  SERVO  EMOVE  WAIT
C
C --- INPUT DATA : LTAB(90,3,6) = THE PRESENT LOCATION TABLE MODULE
C                                 THAT IS EITHER READ IN FROM THE DISC
C                                 BY 'RDMOD' OR IS UPDATED BY A CALL TO
C                                 'SAMPLE'.
C                   PROG(105,6)  = THE PRESENT PROGRAM MODULE THAT
C                                 IS EITHER READ IN FROM THE DISC BY
C                                 'RDMOD' OR IS UPDATED BY A CALL TO
C                                 'SAMPLE'.
C                   SW-35        = INPUT SWITCH -35- WHOSE VALUE IS EITHER
C                                 0 OR -4000 (0 TO +5 VOLTS).  THIS SWITCH
C                                 IS USED TO HOLD THE PROGRAM IN AN INITIAL
C                                 STATE UNTIL EXECUTION WHICH IS SIGNALED
C                                 BY FLIPPING THIS SWITCH TO THE UP POSITION.
C                                 THIS SWITCH IS ALSO USED TO INDICATE
C                                 WHETHER THE PROGRAM IS TO BE EXECUTED
C                                 ONCE OR REPEATEDLY.
C                   SW-28        = THE VALUE OF THIS SWITCH IS SET EQUAL TO
C                                 THE VARIABLE 'EABORT' AND IS USED AS AN
C                                 ABORT SWITCH WHICH STOPS THE EXECUTION
C                                 OF THE PROGRAM AND CALLS 'SAMPLE'.  THIS
C                                 ALLOWS THE MODIFICATION OF THE PROGRAM
C                                 MODULE, THE LOCATION TABLE MODULE, THE
C                                 STORING OF THESE TWO MODULES ON DISC,
C                                 AND THE RETURN TO EXECUTION OF THE
C                                 PROGRAM AT ANY SPECIFIED LINE IN THE
C                                 PROGRAM MODULE.
C                   LL           = THE SPECIFIED LINE OF THE PROGRAM
C                                 MODULE TO BE EXECUTED NEXT [FROM'SAMPLE'].
C                   KK           = FLAG RETURNED FROM A CALL TO 'SAMPLE'
C                                 TO INDICATE IF A RE-ENTRY TO A
C                                 PARTICULAR LINE IS DESIRED.
C
C --- OUTPUT DATA : ELE(6)        = THE VALUES OF THE LOCATION TABLE
C                                 POINTERS AND THE FUNCTION FLAGS FOR
C                                 LINE 'LL' OF THE PROGRAM MODULE (AN
C                                 'ELEMENTAL MOVE' COMMAND).
C                    KK           = FLAG USED TO INDICATE WHETHER THE
C                                 CALL TO 'EMOVE' IS FOR THE EXECUTION
C                                 OF A LINE IN THE PROGRAM MODULE (KK=0),
C                                 OR THAT EXECUTION IS TO BEGIN AT
C                                 AT LINE 'LL' IN THE PROGRAM MODULE
C                                 WITH THE ARM STARTING AT SOME ARBITRARY
C                                 LOCATION.  COMES FROM THE CALL TO 'SAMPLE'
C                                 WHERE RE-ENTRY INTO THE EXECUTION OF THE
C                                 PROGRAM MODULE IS SPECIFIED TO BE AT
C                                 LINE 'LL' AND THE PRESENT LOCATION
C                                 OF THE ARM MIGHT BE ANYWHERE IN ITS
C                                 WORKING SPACE. (KK=1).
```

```
C
C --- FUNCTION: SUPPLIES THE THIRD LEVEL OF THE HIERARCHY WITH
C                AN ELEMENTAL MOVE COMMAND, IE. REQUESTS THE
C                EXECUTION OF A SINGLE LINE OF THE PROGRAM MODULE.
C                ALSO, ALLOWS THE ENTERING OF NEW PROGRAM LINES AND
C                LOCATION POINTS, AND INITIALIZING THE SYSTEM TO
C                BEGIN EXECUTION.
C
        IMPLICIT INTEGER (B-Z)
        COMMON/ARMBUF/INBUF(64),OUTBUF(64)
        COMMON/STEP/LL
        COMMON/LMOD/LTAB(90,4,7),PRES,DEST
        COMMON/PMOD/PROG(105,6),ENDP,BRNCH(6),M
        COMMON/OUT/JPOS(8)
        COMMON/EM/ELE(6)
        EQUIVALENCE(EABORT,INBUF(28))
C
C   INITIALIZE THE FLAG 'KK'.
C
        KK=0
C
C   THIS SUBROUTINE ('RDMOD') READS THE PROGRAM MODULE AND THE LOCATION
C       TABLE MODULE FROM THE DISC AND STORES THEM IN THE MATRICES
C       PROG(105,6) AND LTAB(90,3,6).
C
        CALL RDMOD(1,0)
C
C   THIS ROUTINE ALLOWS THE ENTRY OF NEW PROGRAM LINES
C       AND LOCATION POINTS, AND ALLOWS A RE-ENTRY INTO THE EXECUTION OF
C       THE PROGRAM AT A SPECIFIED LINE 'LL'.
C
        CALL SAMPLE(1)
        GOTO 4000
3400    CALL SAMPLE(0)
        IF(LL.LE.0)GOTO 4000
        KK=1
        GOTO 301
C
C   A REQUEST IS MADE FOR THE OPERATOR TO FLIP SW-35 TO THE DOWN
C       POSITION SO THAT THE STARTING LOCATION JOINT POSITION
C       VALUES CAN BE SENT TO THE SERVOS.
C
4000    WRITE(6,4001)
4001    FORMAT(8X'---SW 35 DOWN--- AND',/
        1 8X'***TYPE [CR]***TO INITIALIZE THE SERVOS WITH THE ',/
        1 24X'STARTING LOCATION.')
        READ(6,5000)FHH
5000    FORMAT(I3)
C
C   THE INPUTS ARE READ INTO THE COMPUTER AND SW-28 (THE 'ABORT'
C       SWITCH IS CHECKED TO SEE IF IT HAS BEEN SET.  IF IT HAS,
C       THEN RETURN TO THE ROUTINE 'SAMPLE' TO DETERMINE WHAT ACTION
C       SHOULD BE TAKEN.
C
        CALL ARMIN
        IF(EABORT.LT.-2000)GOTO 3400
```

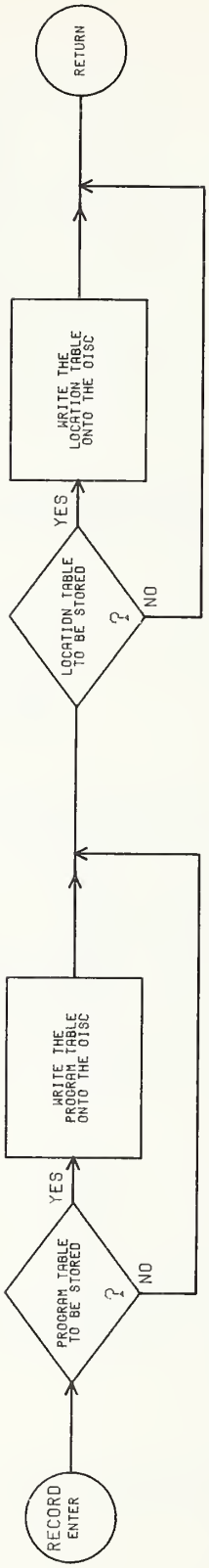
```
C
C   INSTRUCT THE OPERATOR TO NOW TURN ON THE SERVOS, AND THAT
C       SW-35 IS TO BE FLIPPED UP TO BEGIN EXECUTION OF THE PROGRAM.
C
        WRITE(6,4002)
4002    FORMAT(8X'TURN ON SERVOS,',/
       1 8X'---SW 35 UP---TO EXECUTE PROGRAM.',/
       1 8X'LEAVE ---SW 35 UP--- FOR REPEATED EXECUTION.')
C
C   SET THE VARIABLE 'M' EQUAL TO THE NUMBER OF EXECUTABLE LINES IN THE
C       PROGRAM.
C
C
C   SET THE VARIABLE 'FR' EQUAL TO THE LOCATION TABLE POINTER
C       FOR THE STARTING LOCATION OF THE PROGRAM.  SEND OUT THE
C       JOINT POSITION VALUES FOR THIS LOCATION TO THE SERVOS, AND
C       WAIT UNTIL SW-35 IS FLIPPED UP BEFORE CONTINUING WITH
C       THE EXECUTION OF THE PROGRAM.
C
        FR=PROG(1,1)
        DO 200 FRR=1,6
200     JPOS(FRR)=LTAB(FR,1,FRR)
        CALL SERVO(0)
        CALL ARMIN
        IF(EABORT.LT.-2000)GOTO 3400
        CALL WAIT(35)
C
C   PLACE THE VALUES (LOCATION TABLE POINTERS AND FUNCTON FLAGS)
C       OF THE PROGRAM LINE 'LL' INTO THE MATRIX 'ELE(6)'.  THIS IS AN
C       'ELEMENTAL MOVE' THAT WILL BE USED AS A COMMAND TO THE THIRD
C       LEVEL OF THE CONTROL SYSTEM BY A CALL TO 'EMOVE'.
C
305     LL=1
301     DO 202 J=1,6
202     ELE(J)=PROG(LL,J)
C
C   TEST TO ASCERTAIN THAT BOTH A PRESENT LOCATION (ELE(1)) AND A
C       DESTINATION (ELE(2)) POINTER ARE AVAILABLE.  IF NOT, THEN
C       PRINT OUT AN ERROR MESSAGE ON THE TERMINAL AND RETURN TO
C       THE SUBROUTINE 'SAMPLE' TO DETERMINE WHAT IS TO BE DONE NEXT.
C
        IF(LTAB(ELE(1),1,1).EQ.0)GOTO 100
        IF(LTAB(ELE(2),1,1).NE.0)GOTO 300
100     WRITE(6,101)
101     FORMAT(8X'PROGRAM HAS NO LOCATION VALUE---',/
       1 2X'***************PROGRAM ABORTED*****************')
        GOTO 3400
C
C   THIS SUBROUTINE CALL CAUSES THE EXECUTION OF A SINGLE PROGRAM
C       MODULE LINE ('ELEMENTAL MOVE' COMMAND).
C
300     CALL EMOVE(KK)
        KK=0
C
C   THE INTERRUPT SWITCH 28 ('EABORT') IS CHECKED TO
C       SEE IF IT IS SET.
```

```
C
      IF(EABORT.LT.-2000)GOTO 3400
C
C   THE PROGRAM LINE COUNTER ('LL') IS INCREMENTED.  IF THIS VALUE DOES
C      NOT EXCEED THE LAST EXECUTABLE LINE, THEN JUMP BACK TO
C      STATEMENT 301 TO CARRY OUT ITS EXECUTION.
C
      LL=LL+1
      IF(PROG(LL,2).NE.0)GOTO 301
C
C   ONCE THE ENTIRE PROGRAM HAS BEEN EXECUTED, THEN TEST SWITCH 35.  IF
C      IT IS UP THEN REPEAT THE PROGRAM.  IF IT IS DOWN, THEN SIT
C      HERE AND CONTINUE MONITORING IT.
C
      CALL WAIT(35)
C
C   BRANCH BACK TO STATEMENT 305 TO BEGIN EXECUTION.
C
      GOTO 305
      END
```

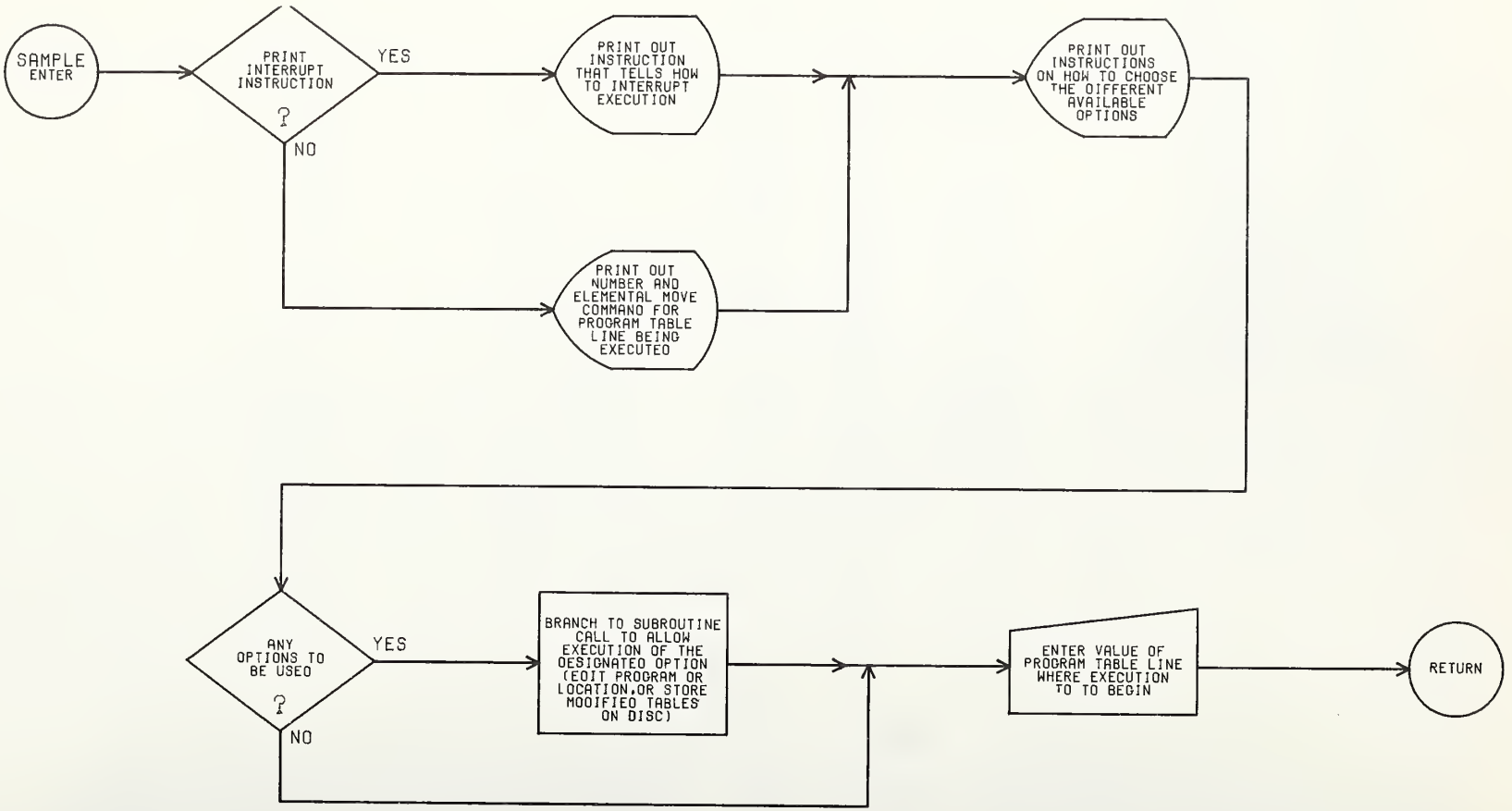

RDMOD
ENTER
PROGRAM TABLE TO BE READ IN
?
YES
NO
READ IN DESIGNATED PROGRAM TABLE FROM THE DISC
SCAN TABLE FOR LAST ELEMENTAL MOVE STATEMENT. STORE THIS LINE NUMBER IN COMMON VARIABLE "M"
LOCATION TABLE TO BE READ IN
?
YES
NO
READ IN DESIGNATED LOCATION TABLE FROM THE DISC
RETURN

```
C --- SUBROUTINE : RDMOD
C
C --- ARGUMENTS : LTM       = IF LTM=1  READ THE LOCATION TABLE MODULE
C                                        OFF THE DISC AND STORE IN THE
C                                        MATRIX 'LTAB(90,4,7)'.
C                             IF LTM=0  DO NOT READ IN THE LOCATION
C                                        TABLE MODULE FROM THE DISC.
C                 PM        = IF  PM=1  READ THE PROGRAM MODULE IN
C                                        FROM THE DISC AND STORE IN THE
C                                        MATRIX 'PROG(105,6)'.
C                             IF  PM=0  DO NOT READ IN THE PROGRAM
C                                        MODULE FROM THE DISC.
C
C --- CALLED BY : EXPRO
C
C --- CALLS SUBROUTINES :
C
C --- INPUT DATA :  THE PROGRAM MODULE ASSIGNED TO DEVICE NUMBER 7,
C                  AND THE LOCATION TABLE MODULE ASSIGNED TO DEVICE
C                  NUMBER 4.
C
C --- OUTPUT DATA : PROG(105,6)  = THE MATRIX CONTAINING THE
C                                  PROGRAM MODULE.
C                  NTAB(90,4,7) = THE MATRIX CONTAINING THE
C                                  LOCATION TABLE MODULE.
C                  ENDP          = THE PROGRAM MODULE LINE NUMBER
C                                   OF THE LAST ELEMENTAL MOVE
C
C --- FUNCTION: READS IN THE PROGRAM MODULE AND THE LOCATION
C              TABLE MODULE FROM THE DISC AND STORES THEM
C              IN THEIR PROPER MATRICES IN THE COMMON BLOCK.
C
      SUBROUTINE RDMOD(LTM,PM)
      IMPLICIT INTEGER (B-R),(U-Z)
      COMMON/LMOD/LTAB(90,4,7),PRES,DEST
      COMMON/PMOD/PROG(105,6),ENDP,BRNCH(6),M
C
C   TEST IF 'PM' SET EQUAL TO ONE.  IF IT IS, THEN READ IN
C      THE PROGRAM MODULE ASSIGNED TO DEVICE NUMBER 7 FROM THE
C      DISC.
C
      IF(PM.NE.1)GOTO 400
      DO 15 JK=1,105
      READ(7,15,END=100)(PROG(JK,J),J=1,6)
15    FORMAT(6(2X,I6))
100   CONTINUE
C
C   TEST IF 'LTM' SET EQUAL TO ONE.  IF IT IS, THEN READ IN
C      THE LOCATION TABLE MODULE ASSIGNED TO DEVICE NUMBER 4
C      FROM THE DISC.
C
400   IF(LTM.NE.1)GOTO 40
      DO 14 JB=1,90
      DO 14 JC=1,4
      READ(4,14,END=40)(LTAB(JB,JC,J),J=1,7)
14    FORMAT(7(2X,I6))

40    RETURN
      END
```

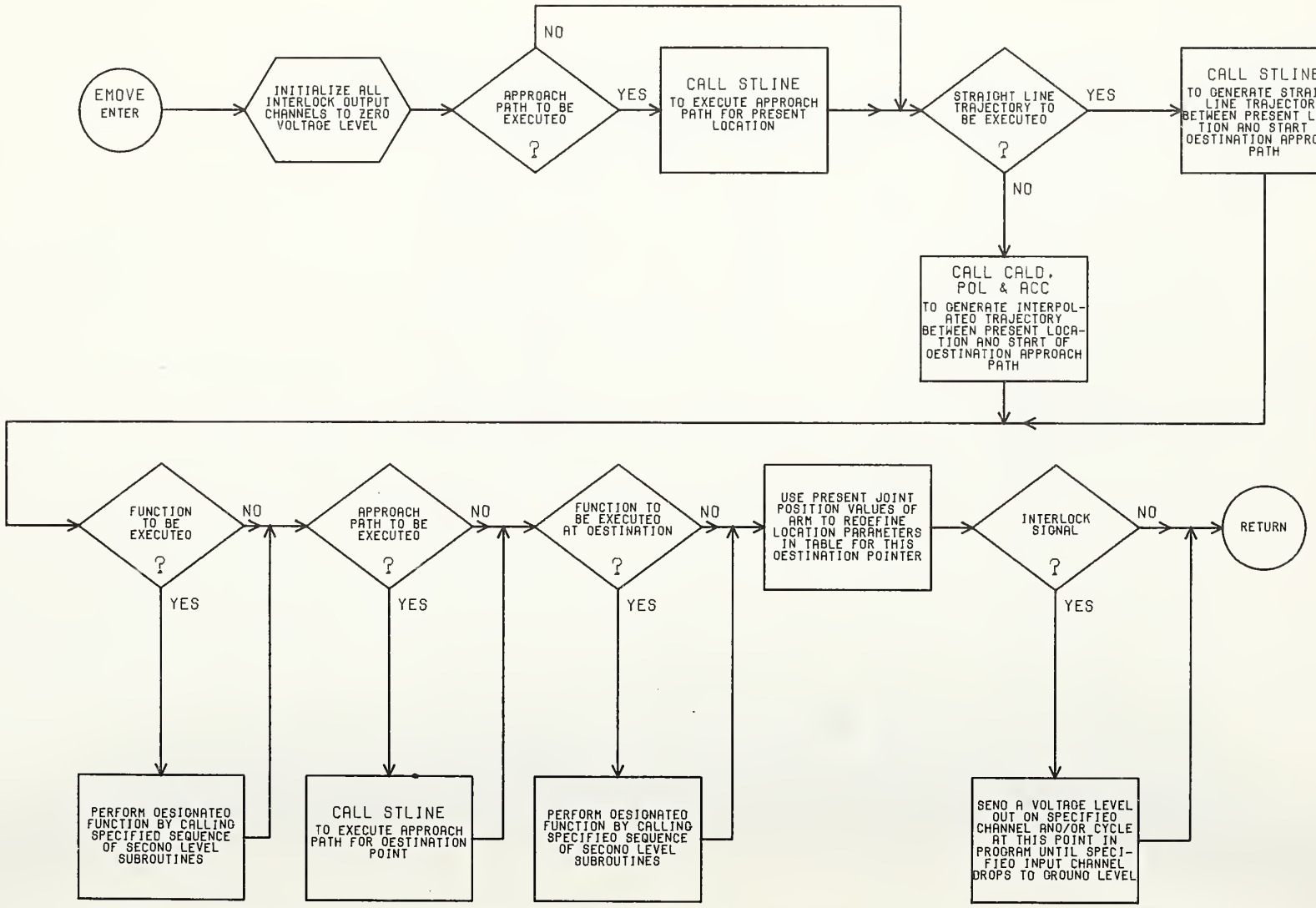

RECORD ENTER
PROGRAM TABLE TO BE STORED ?
YES
NO
WRITE THE PROGRAM TABLE ONTO THE DISC
LOCATION TABLE TO BE STORED ?
YES
NO
WRITE THE LOCATION TABLE ONTO THE DISC
RETURN

```
C --- SUBROUTINE : RECORD
C
C --- ARGUMENTS : LTM       = IF LTM=1  STORE THE LOCATION TABLE MODULE
C                                        (LTAB(90,4,7)) ON THE DISC UNDER
C                                        THE NAME ASSIGNED TO DEVICE 2.
C                             IF LTM=0  DO NOT STORE THE LOCATION TABLE
C                                        ON THE DISC.
C                 PM        = IF  PM=1  STORE THE PROGRAM MODULE
C                                        (PROG(105,6)) ON THE DISC UNDER
C                                        THE NAME ASSIGNED TO DEVICE 3.
C                                  PM=0  DO NOT STORE THE PROGRAM MODULE ON
C                                        THE DISC.
C
C --- CALLS SUBROUTINES :
C
C --- INPUT DATA : NTAB(90,4,7) = THE MATRIX CONTAINING THE LOCATION
C                                  TABLE MODULE.
C                   PROG(105,6)  = THE MATRIX CONTAINING THE PROGRAM
C                                  MODULE.
C
C --- OUTPUT DATA : THE LOCATION TABLE MODULE STORED ON DISC UNDER
C                   THE NAME ASSIGNED TO DEVICE 2.
C                   THE PROGRAM MODULE STORED ON THE DISC UNDER THE
C                   NAME ASSIGNED TO DEVICE 3.
C
C --- FUNCTION: STORES THE LOCATION TABLE MODULE (LTAB(90,4,7)) AND
C               THE PROGRAM MODULE (PROG(105,6)) ON DISC UNDER
C               ASSIGNED NAMES.
C
        SUBROUTINE RECORD(LTM,PM)
        IMPLICIT INTEGER(B-Z)
        COMMON/LMOD/LTAB(90,4,7),PRES,DEST
        COMMON/PMOD/PROG(105,6),ENDP,BRNCH(6),M
C
C   TEST IF 'PM' SET EQUAL TO ONE.  IF IT IS, THEN WRITE THE
C       PROGRAM MODULE ONTO THE DISC.
C
        IF(PM.NE.1)GOTO 400
        DO 15 JK=1,105
        WRITE(3,15)(PROG(JK,J),J=1,6)
15      FORMAT(6(2X,I6))
        END FILE 3
C
C   TEST IF 'LTM' SET EQUAL TO ONE.  IF IT IS, THEN WRITE THE
C       LOCATION TABLE MODULE ONTO THE DISC.
C
400     IF(LTM.NE.1)GOTO 40
        DO 14 JB=1,90
        DO 14 JC=1,4
        WRITE(2,14)(LTAB(JB,JC,J),J=1,7)
14      FORMAT(7(2X,I6))
        END FILE 2
40      RETURN
        END
```


SAMPLE ENTER
PRINT INTERRUPT INSTRUCTION ?
YES
NO
PRINT OUT INSTRUCTION THAT TELLS HOW TO INTERRUPT EXECUTION
PRINT OUT NUMBER AND ELEMENTAL MOVE COMMAND FOR PROGRAM TABLE LINE BEING EXECUTED
PRINT OUT INSTRUCTIONS ON HOW TO CHOOSE THE DIFFERENT AVAILABLE OPTIONS
ANY OPTIONS TO BE USED ?
YES
NO
BRANCH TO SUBROUTINE CALL TO ALLOW EXECUTION OF THE DESIGNATED OPTION (EDIT PROGRAM OR LOCATION,OR STORE MODIFIED TABLES ON DISC)
ENTER VALUE OF PROGRAM TABLE LINE WHERE EXECUTION TO TO BEGIN
RETURN

```
C --- SUBROUTINE : SAMPLE
C
C --- ARGUMENTS : KG          = USED AS A FLAG TO CHOOSE THE MESSAGES TO BE
C                               PRINTED OUT ON THE TERMINAL.
C
C --- CALLED BY : EXPRO
C
C --- CALLS SUBROUTINES : ARMIN  LOCTAB  PROGS  RECORD  PLINE
C
C --- INPUT DATA : LL         = THE PRESENT PROGRAM LINE NUMBER THAT IS
C                               BEING EXECUTED.
C                   INBUF(28) = THE ABORT SWITCH VALUE.
C                   INBUF(29) = THE VALUE OF THE SWITCH THAT IS USED FOR
C                               CALLING UP THE TEACH LOCATION POINT
C                               PROGRAMS.
C                   INBUF(30) = THE VALUE OF THE SWITCH THAT IS USED FOR
C                               CALLING UP THE ROUTINES FOR ENTERING
C                               A PROGRAM.
C                   INBUF(31) = THE VALUE OF THE SWITCH USED TO
C                               INDICATE IF THE PROGRAM MODULE AND THE
C                               LOCATION TABLE MODULE ARE TO BE STORED ON
C                               DISC.
C
C --- OUTPUT DATA : LL        = THE NUMBER OF THE PROGRAM LINE TO BE
C                               EXECUTED NEXT.
C
C --- FUNCTION: THIS PROGRAM IS CALLED BY THE ABORT SWITCH AND ALLOWS
C               THE ENTRY OF NEW OR MODIFIED LOCATION VALUES, THE
C               EDITING OF THE PROGRAM MODULE, THE STORING OF THE NEW
C               LOCATION TABLE MODULE AND PROGRAM MODULE ON DISC, AND
C               THE ABILITY TO RETURN TO THE EXECUTION PROGRAM
C               AT ANY LINE IN THE PROGRAM MODULE.
C
        SUBROUTINE SAMPLE(KG)
        IMPLICIT INTEGER(B-Z)
        COMMON/ARMBUF/INBUF(64),OUTBUF(64)
        COMMON/PMOD/PROG(105,6),ENDP,BRNCH(6),M
        COMMON/IND/IN(30),FL(6),NAME(3),NPT(3,20)
        COMMON/STEP/LL
        EQUIVALENCE(EABORT,INBUF(28))
C
C   TEST ARGUMENT 'KG', IF EQUAL TO ZERO, THEN SKIP NEXT MESSAGE.
C
5000    IF(KG.EQ.0)GOTO 6000
        WRITE(6,6240)
6240    FORMAT(8X'---SW 28 UP--- INTERRUPTS  RUN PROGRAM')
        GOTO 6235
C
C   PRINT OUT PRESENT PROGRAM LINE NUMBER AND PRINT OUT THE ENGLISH
C       TEXT OF THIS LINE AS THE ORIGINAL 'GOTO' STATEMENT.
C
6000    WRITE(6,6200)LL
6200    FORMAT(8X'YOU ARE AT PROGRAM STEP',I4)
        IF(LL.EQ.0)GOTO 425
        MR=0
        DO 204 J=101,105
```

```
        DO 204 L=1,6
        MR=MR+1
204     IN(MR)=PROG(J,L)
        CALL INDEX(1)
        CALL PLINE(LL)
C
C   PRINTS OUT ON TERMINAL THE VARIOUS OPERATIONS AVAILABLE TO THE USER.
C
425     WRITE(6,6201)
6201    FORMAT(8X'---SW 28 UP--- RETURNS PROGRAM TO BEGINNING')
6235    WRITE(6,6225)
6225    FORMAT(8X'---SW 29 UP--- TEACH LOCATION',/
        1 8X'---SW 30 UP--- RECORD PROGRAM SEQUENCE',/
        1 8X'---SW 31 UP--- STORE LOCATION TABLE ON DISC',/
        1 8X'***TYPE (CR)*** TO CONTINUE')
        READ(6,6001)TR
6001    FORMAT(I4)
C
C   READS IN THE SWITCH VALUES TO DETERMINE WHICH IS THE PROPER SUBROUTINE
C       TO BRANCH TO.
C
        CALL ARMIN
        DO 6203 J=1,4
        K=27+J
        CHKV=IABS(INBUF(K))
        IF(CHKV.GT.2000)GOTO(45,11,12,13)J
6203    CONTINUE
        GOTO 45
11      CALL LOCTAB
        GOTO 5000
12      CALL PROGS
        GOTO 5000
13      CALL RECORD(1,0)
        WRITE(6,225)
225     FORMAT(2X'LOCATION TABLE STORED')
        GOTO 5000
45      IF(KG.EQ.1)GOTO 40
        WRITE(6,6220)
6220    FORMAT(8X'WHICH PROGRAM STEP DO YOU WANT?')
        READ(6,6221)LL
6221    FORMAT(I4)
40      RETURN
        END
```


EMOVE
ENTER
INITIALIZE ALL INTERLOCK OUTPUT CHANNELS TO ZERO VOLTAGE LEVEL
APPROACH PATH TO BE EXECUTED ?
NO
YES
CALL STLINE
TO EXECUTE APPROACH PATH FOR PRESENT LOCATION
STRAIGHT LINE TRAJECTORY TO BE EXECUTED ?
YES
NO
CALL STLINE
TO GENERATE STRAIGHT LINE TRAJECTORY BETWEEN PRESENT LOCATION AND START OF DESTINATION APPROACH PATH
CALL CALD, POL & ACC
TO GENERATE INTERPOLATED TRAJECTORY BETWEEN PRESENT LOCATION AND START OF DESTINATION APPROACH PATH
FUNCTION TO BE EXECUTED ?
NO
YES
PERFORM DESIGNATED FUNCTION BY CALLING SPECIFIED SEQUENCE OF SECOND LEVEL SUBROUTINES
APPROACH PATH TO BE EXECUTED ?
NO
YES
CALL STLINE
TO EXECUTE APPROACH PATH FOR DESTINATION POINT
FUNCTION TO BE EXECUTED AT DESTINATION ?
NO
YES
PERFORM DESIGNATED FUNCTION BY CALLING SPECIFIED SEQUENCE OF SECOND LEVEL SUBROUTINES
USE PRESENT JOINT POSITION VALUES OF ARM TO REDEFINE LOCATION PARAMETERS IN TABLE FOR THIS DESTINATION POINTER
INTERLOCK SIGNAL ?
NO
YES
SEND A VOLTAGE LEVEL OUT ON SPECIFIED CHANNEL AND/OR CYCLE AT THIS POINT IN PROGRAM UNTIL SPECIFIED INPUT CHANNEL DROPS TO GROUND LEVEL
RETURN

```
C --- SUBROUTINE : EMOVE
C
C --- ARGUMENTS : KK =    IF KK=1  THEN THE STRAIGHT LINE DISTANCE
C                                    FROM THE PRESENT LOCATION OF THE
C                                    ARM WHICH MIGHT BE POSITIONED
C                                    ANYWHERE IN THE WORK SPACE, TO THE
C                                    DESTINATION SPECIFIED IN THE GIVEN
C                                    LINE OF THE PROGRAM MODULE IS CALCULATED
C                                    AND USED AS THE TRAJECTORY FOR THIS
C                                    COMMANDED 'ELEMENTAL MOVE'.
C                          IF KK=0   THE TRAJECTORY IS ASSUMED TO BE BETWEEN
C                                    THE TWO LOCATION TABLE POINTERS SPECIFIED
C                                    BY THE 'ELEMENTAL MOVE' COMMAND (ELE(6)),
C                                    IE. THE ACTUAL LOCATION OF THE ARM IS
C                                    CONSIDERED TO BE THE SAME AS THAT DEFINED
C                                    BY THE JOINT POSITION VALUES OF
C                                    THE 'PRESENT LOCATION'POINTER
C                                    IN THIS ELEMENTAL MOVE COMMAND.
C
C --- CALLED BY : RUNPRO
C
C --- CALLS SUBROUTINES : STLINE  WAIT  GRASP  POS  POL  DETECT
C                         BAL  RELEAS  PTOUCH  COOR  SERVO  CALD  ACC
C
C --- INPUT DATA : KK            = THE FLAG THAT INDICATES WHETHER OR
C                                   NOT THE ACTUAL LOCATION OF THE ARM
C                                   IS AT THE LOCATION SPECIFIED BY THE
C                                   'PRESENT LOCATION' POINTER (ELE(1))
C                                   FROM THE ELEMENTAL MOVE COMMAND.
C                  SW-28           = THE ABORT SWITCH ('EABORT') USED TO
C                                   BRANCH THE CONTROL SYSTEM BACK TO
C                                   THE SUBROUTINE 'SAMPLE' TO ALLOW
C                                   MODIFICATION OF THE PROGRAM OR
C                                   LOCATION TABLE MODULES AND RE-ENTRY INTO
C                                   THE PROGRAM AT A SPECIFIED LINE.
C                  ELE(6)          = THE MATRIX THAT CONTAINS A SINGLE
C                                   'ELEMENTAL MOVE' COMMAND, IE. THE LOCATION
C                                   TABLE POINTERS FOR THE 'PRESENT LOCATION',
C                                   THE 'DESTINATION', THE VALUE OF THE
C                                   VELOCITY INDICATOR, AND THE VARIOUS
C                                   FUNCTION FLAGS.
C
C --- OUTPUT DATA : THE PROPER SEQUENCE OF CALLS TO THE NECESSARY
C                   SUBROUTINES ('PRIMITIVES') TO CARRY OUT THE
C                   'ELEMENTAL MOVE' COMMAND.
C
C --- FUNCTION: FROM THE INFORMATION CONTAINED IN A SINGLE LINE
C               FROM THE PROGRAM MODULE, THIS SUBROUTINE,
C               WHICH IS THE THIRD LEVEL OF THE CONTROL HIERARCHY,
C               CALLS THE SUBROUTINES (PRIMITIVES) TO GENERATE A
C               TRAJECTORY BETWEEN THE TWO LOCATIONS SPECIFIED,
C               CAUSES THE ARM TO MOVE ALONG THIS TRAJECTORY AT A
C               GIVEN VELOCITY, AND TESTS ON THE FUNCTION
C               FLAGS AND CALLS SUBROUTINES IN ACCORD WITH
C               THE VALUES DETECTED.
C
```

```
      SUBROUTINE EMOVE(KK)
      IMPLICIT INTEGER(B-Z)
      COMMON/ARMBUF/INBUF(64),OUTBUF(64)
      COMMON/GRIP/HAND
      COMMON/LMOD/LTAB(90,4,7),PRES,DEST
      COMMON/OUT/JPOS(8)
      COMMON/EM/ELE(6)
      COMMON/DIS/ADIST
      COMMON/OFFS/OFF(3)
      COMMON/CORT/AC(6),AXEP,AYEP,AZEP
      COMMON/END/PNT
      EQUIVALENCE(NPO,INBUF(33))
      EQUIVALENCE(EABORT,INBUF(28))
C
C   A CALL TO 'SERVO(-1)' IS MADE TO RESET ALL OF THE OUTPUT CHANNEL
C      LEVELS TO ZERO.
C
      CALL SERVO(-1)
C
C   THE COORDINATE OFFSET VALUES THAT ARE USED IN THE SENSOR
C      SUBROUTINE CALLS ARE SET EQUAL TO ZERO.
C
      DO 70 J=1,3
70    OFF(J)=0
C
C   TESTS ON THE FLAG 'KK' AND IF SET, JUMPS THE APPROACH PATH CALCULATION.
C
      IF(KK.EQ.1)GOTO 220
C
C   TESTS THE FLAG TO SEE IF THE ARM STOPPED AT THE PRESENT LOCATION
C      IF IT DID (LTAB(ELE(1),2,7)=1), THEN CONTINUE.  IF IT DID
C      NOT STOP(LTAB(ELE(1),2,7)=0), THEN JUMP OVER THE APPROACH PATH
C      CALCULATION (IT IS ASSUMED THAT THERE IS NO APPROACH PATH FOR
C      THOSE LOCATIONS THAT THE ARM DOES NOT STOP AT).
C
      IF(LTAB(ELE(1),2,7).EQ.0)GOTO 220
C
C   THE FOLLOWING SUBROUTINE ('STLINE') GENERATES A STRAIGHT LINE PATH
C      FROM ITS PRESENT LOCATION TO A POINT A DELTA X, DELTA Y, DELTA Z
C      DISTANCE AWAY (THE DELTAS ARE LISTED AS ARGUMENTS).  HERE IT IS
C      USED TO GENERATE THE SPECIFIED APPROACH PATH FOR THE 'PRESENT
C      LOCATION' POINT.
C
      CALL STLINE(LTAB(ELE(1),3,4),LTAB(ELE(1),3,5),LTAB(ELE(1),3,6),1)
C
C   'EABORT' IS THE EQUIVALENT VALUE OF SWITCH 28 WHICH IS USED TO
C      ABORT THE EXECUTION OF THE PROGRAM AND RETURN TO THE SUBROUTINE
C      'SAMPLE'.  THIS VALUE IS CHECKED CONSTANTLY IN THE SUBROUTINE
C      'SERVO' AND AFTER EVERY SUBROUTINE CALL THAT RESULTS IN A CALL
C      TO 'SERVO' (IE. EVERY TIME THE ARM IS COMMANDED TO MOVE)>
C
      IF(EABORT.LT.-2000)GOTO 40
C
C   THE FUNCTON FLAG IS CHECKED TO SEE IF IT IS A 'GRASP PROXIMITY'
C      (ELE(6)=3).  IF IT IS, THEN CALL 'GRASP' TO CLOSE THE FINGERS
C      FOR A SEARCH.
```

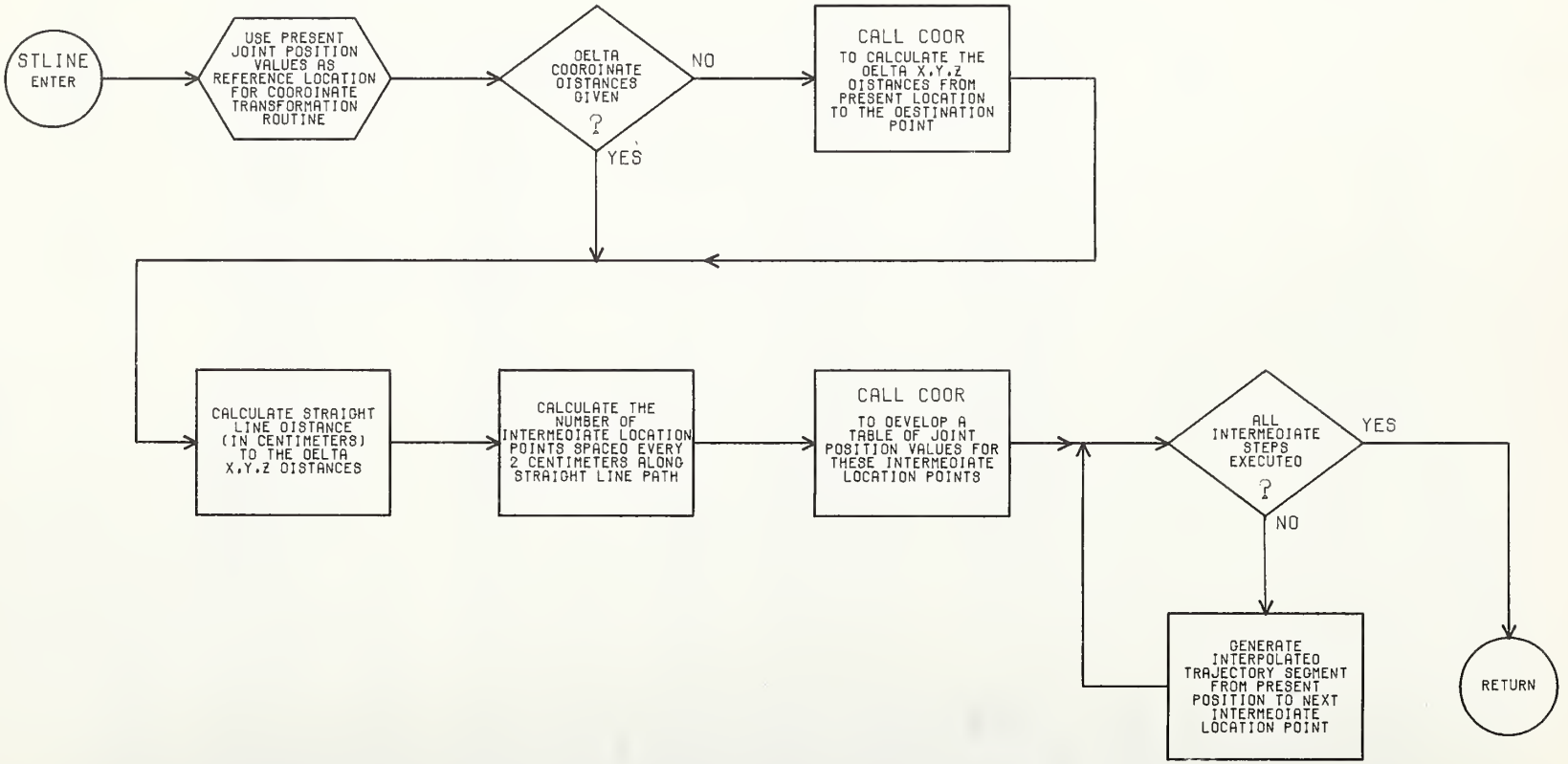
```
C
220     IF(ELE(6).NE.3)GOTO 230
        CALL GRASP(1)
C
C   IF THE FUNCTION 'STRAIGHT LINE' IS CALLED FOR (ELE(6)=9), THEN
C       BRANCH TO THE SUBROUTINE CALLS 'POS' AND 'STLINE'.
C
230     IF(ELE(6).EQ.9)GOTO 301
C
C   THE ARGUMENT 'KK' IS CHECKED TO SEE IF THE TRAJECTORY IS TO BE
C       A STRAIGHT LINE FROM THE PRESENT LOCATION OF THE ARM
C       TO THE DESTINATION CALLED FOR IN PROGRAM LINE 'LL' (KK=1),
C       OR IF IT IS TO BE LINEAR INTERPOLATION PATH BETWEEN
C       THE TWO LOCATIONS SPECIFIED IN THE PROGRAM LINE 'LL' (KK=0).
C
        IF(KK.NE.1)GOTO 300
C
C   TO GENERATE A STRAIGHT LINE TRAJECTORY FROM THE PRESENT LOCATION
C       OF THE ARM, THE SUBROUTINE 'POS' IS CALLED TO READ IN THE PRESENT
C       JOINT POSITION VALUES TO BE USED BY 'STLINE'.
C
301     CALL POS
C
C   THIS SUBROUTINE ('STLINE(DX,DY,DZ,I)') GENERATES AND EXECUTES A
C       STRAIGHT LINE TRAJECTORY BETWEEN TWO LOCATION POINTS.
C       IF THE ARGUMENT 'I' IS SET EQUAL TO ZERO, THEN THE STRAIGHT LINE
C       IS GENERATED BETWEEN THE PRESENT LOCATION OF THE ARM (WHICH MAY OR
C       MAY NOT BE DEFINED IN THE 'LOCATION TABLE MODULE') AND THE
C       START OF THE APPROACH PATH OF THE 'DESTINATION' POINT LISTED
C       IN THE 'ELEMENTAL MOVE' COMMAND (LINE 'LL' OF THE PROGRAM
C       MODULE).  IF 'I' IS SET EQUAL TO ONE, THEN THE STRAIGHT LINE
C       TRAJECTORY IS FROM THE PRESENT LOCATION OF THE ARM TO A
C       LOCATION POINT 'DX', 'DY', 'DZ' AWAY.
C
        CALL STLINE(0,0,0,0)
C
C   IF A STRAIGHT LINE TRAJECTORY HAS BEEN USED THEN THE
C       INTERPOLATION CALCULATION IS SKIPPED.
C
        GOTO 325
C
C   IF A STRAIGHT LINE IS NOT GENERATED, THEN THE INTERPOLATION
C       ROUTINE IS USED.  HERE, 'CALD' IS CALLED TO CALCULATE
C       THE STRAIGHT LINE DISTANCE (IN CENTIMETERS) BETWEEN THE PRESENT
C       LOCATION OF THE ARM AND THE START OF THE APPROACH PATH
C       FOR THE DESTINATION POINT.  'POL' IS THEN CALLED TO CALCULATE
C       THE CORRECT DELTAS FOR EACH JOINT TO BE USED BY 'ACC' TO CAUSE
C       THE JOINT POSITION COMMANDS TO BE INCREMENTED BY THESE
C       DELTAS AT PERIODIC TIMES TO GENERATE THE SPECIFIED VELOCITY
C       AND ACCELERATION AND DECCELERATION WHERE REQUIRED.
C
300     CALL CALD
        CALL POL
        CALL ACC
C
C   THIS SWITCH IS EXPLAINED ABOVE.
```

```
C
325     IF(EABORT.LT.-2000)GOTO 40
C
C   AT THIS POINT IN THE EXECUTION, THE ARM HAS ALREADY ACCOMPLISHED ITS
C       TRAJECTORY MOTION BY THE ABOVE SUBROUTINE CALLS.
C       IT NOW REMAINS TO MOVE THROUGH THE FINAL APPROACH PATH IF ONE IS
C       CALLED FOR, AND TO DETERMINE WHAT FUNCTIONS ARE TO BE
C       ACCOMPLISHED.  FIRST, IT IS DETERMINED IF THE ARM
C       WILL STOP AT THE END OF THIS TRAJECTORY (LTAB(ELE(2),2,1)=1).
C       IF NOT, THEN  RETURN TO THE CALLING PROGRAM.
C
400     IF(LTAB(ELE(2),2,7).EQ.0)GOTO 40
C
C   THE COMMON VARIABLE 'PNT' IS SET EQUAL TO ONE.  THIS FLAG IS
C       USED BY THE SUBROUTINE 'SERVO'.  WHEN IT IS SET TO ONE,
C       'SERVO' WILL MAINTAIN CONTROL UNTIL THE POSITION OF ALL OF THE
C       JOINTS IS WITHIN SOME SPECIFIED MINIMUM DISTANCE OF THE
C       COMMANDED POSITION.  THEREFORE, THE CONTROL PROGRAM DOES NOT
C       INITIATE THE NEXT STEP UNTIL THE ARM HAS ARRIVED AT THIS
C       END POINT OF THE TRAJECTORY.
C
        PNT=1
        CALL SERVO(0)
C
C   THE VARIABLE 'FUN' IS SET EQUAL TO THE VALUE OF THE FUNCTION FLAG
C       ('ELE(6)') PLUS ONE, TO BE USED TO COMMAND A MULTIPLE 'GOTO'
C       STATEMENT TO THE APPROPRIATF CODE TO CARRY OUT THE SPECIFIED
C       FUNCTION.  ('FUN'=1,NO FUNCTION;  =2 'GRASP'   =3 'RELEASE'
C       =4 'GRASP PROXIMITY'   =5 'RELEASE PROXIMITY'   =6 'DETECT'
C       =7 'BALANCE'   =8 'UNSTACK'   =9 'PTOUCH').
C
        FUN=1+ELE(6)
        GOTO(20,20,20,3,4,20,20,7,8,20,20)FUN
C
C   IF THERE IS AN APPROACH PATH, IT IS GENERATED AND EXECUTED BY
C       THIS CALL TO 'STLINE'.
C
20      CALL STLINE(-LTAB(ELE(2),3,4),-LTAB(ELE(2),3,5),-LTAB(ELE(2),3,6),1)
        IF(EABORT.LT.-2000)GOTO 40
C
C   THIS MULTIPLE 'GOTO' STATEMENT CAUSES THE BRANCHING TO THE APPROPRIATE
C       CODE TO CARRY OUT THE FUNCTION SPECIFIED.
C
        GOTO(110,1,2,3,4,5,6,7,8,110,10)FUN
C
C   THIS IS THE CODE FOR A 'GRASP PROXIMITY' FUNCTION.  THE HAND IS
C       MOVED DOWN THE APPROACH PATH TO WITHIN SIX CENTIMETERS OF THE
C       DESTINATION POINT BY A CALL TO 'STLINE'.  IT HAS TO FIRST
C       STOP ABOVE THE OBJECT TO USE ITS PROXIMITY SENSORS TO DETECT IT.
C       IT DETERMINES THAT THE OBJECT IS UNDER, AT LEAST, ONE OF
C       THE SENSORS BY A CALL TO 'DETECT' WHICH MOVES THE HAND
C       UNTIL ONE OF THE SENSORS REACHES A THRESHOLD VALUE.  THE HAND IS
C       THEN MOVED IN THE DIRECTION TO EQUALIZE THE TWO SENSOR INPUTS BY
C       A CALL TO 'BAL'.  NOW THAT THE HAND IS CENTERED OVER THE OBJECT,
C       THE FINGERS ARE OPENED BY A CALL TO 'RELEAS'.  THE  HAND DESCENDS
C       THE REMAINING SIX CENTIMETERS BY A CALL TO 'STLINE', AND THE
```

```
C       FINGERS ARE CLOSED ON THE OBJECT BY A CALL TO 'GRASP'.  IF THE
C       FINGER POSITION INDICATOR ('HAND') IS LESS THAN 7200, THEN THE
C       FINGERS ARE CLOSED TOO FAR TO BE HOLDING AN OBJECT.  IF THIS IS
C       THE CASE THEN THE HAND IS RAISED BACK UP SIX CENTIMETERS
C       BY A CALL TO 'STLINE' AND THE SEARCH BEGUN AGAIN.
C
3       NZ=600-LTAB(ELE(2),3,6)
        CALL STLINE(-LTAB(ELE(2),3,4),-LTAB(ELE(2),3,5),NZ,1)
480     IF(EABORT.LT.-2000)GOTO 40
        CALL DETECT
        IF(EABORT.LT.-2000)GOTO 40
        CALL BAL
        IF(EABORT.LT.-2000)GOTO 40
        CALL RELEAS(0)
        IF(EABORT.LT.-2000)GOTO 40
        CALL STLINE(0,0,-600,1)
        IF(EABORT.LT.-2000)GOTO 40
        CALL GRASP(0)
        IF(EABORT.LT.-2000)GOTO 40
        IF(HAND.GT.7200)GOTO 110
        CALL STLINE(0,0,600,1)
        GOTO 480
C
C   THE FOLLOWING CODE PERFORMS THE 'RELEASE PROXIMITY' FUNCTION.
C       FIRST, THE HAND IS MOVED TO A POSITION ONE CENTIMETER ABOVE
C       AND FIVE CENTIMETERS TO THE SIDE (SINCE THE PROXIMITY SENSORS
C       ARE LOCATED ON THE OUTSIDE OF RATHER THICK FINGERS, THIS SIDEWAYS
C       MOTION IS NECESSARY TO BRING ONE OF THE SENSORS TO A POSITION
C       WHERE IT WOULD BE OVER THE EXPECTED LOCATION OF THE UNDERLYING
C       OBJECT) OF THE DESTINATION POINT BY A CALL TO 'STLINE'.  THE
C       HAND IS MOVED SIDEWAYS UNTIL ONE OF THE SENSORS HAS REACHED THE
C       THRESHOLD LEVEL BY A CALL TO 'DETECT'.  THE CALL TO 'BAL' CAUSES
C       THE HAND TO MOVE IN THE DIRECTION OF THE SENSOR THAT WAS AT
C       THRESHOLD UNTIL THE SENSOR VALUE DECREASES BELOW A CERTAIN
C       VALUE, INDICATING THAT THE EDGE OF THE UNDERLYING
C       OBJECT HAS BEEN REACHED.  THE HAND IS THEN MOVED THE ADDITIONAL
C       DISTANCE OF THE THICKNESS OF THE FINGER TO CAUSE THE SIDE OF THE
C       OBJECT IN THE HAND TO LINE UP FLUSH WITH THE SIDE OF THE
C       UNDERLYING OBJECT.  THE CALL TO 'STLINE' CAUSES THE HAND TO
C       MOVE DOWN THE REMAINING ONE CENTIMETER AND THE FINGERS ARE
C       OPENED AND THE OBJECT RELEASED BY THE CALL TO 'RELEAS'.
C
4       NZ=100-LTAB(ELE(2),3,6)
        NY=500-LTAB(ELE(2),3,5)
        CALL STLINE(-LTAB(ELE(2),3,4),NY,NZ,1)
        IF(EABORT.LT.-2000)GOTO 40
        CALL DETECT
        IF(EABORT.LT.-2000)GOTO 40
        CALL BAL
        IF(EABORT.LT.-2000)GOTO 40
        CALL STLINE(0,0,-100,1)
        IF(EABORT.LT.-2000)GOTO 40
        CALL RELEAS(0)
        GOTO 110
C
C   THE 'GRASP' FUNCTON IS ACCOMPLISHED BY A CALL TO 'GRASP'.
```

```
C
1       CALL GRASP(0)
        GOTO 110
C
C   THE 'RELEASE' FUNCTION IS ACCOMPLISHED BY A CALL TO 'RELEAS'.
C
2       CALL RELEAS(0)
        GOTO 110
C
C   THE 'DETECT' FUNCTION IS ACCOMPLISHED BY A CALL TO 'DETECT'.
C
5       CALL DETECT
        GOTO 110
C
C   THE 'BALANCE' FUNCTION IS ACCOMPLISHED BY A CALL TO 'BAL'.
C
6       CALL BAL
        GOTO 110
C
C   THE FOLLOWING CODE IS USED TO GENERATE THE 'UNSTACK' FUNCTION.
C       THE FINGERS ARE CLOSED BY A CALL TO 'GRASP' TO MAKE CERTAIN
C       THAT THE SENSOR TO DETECT THE TOP OF THE STACK IS OVER THE
C       EXPECTED LOCATION OF THE STACK.  THE CALL TO 'PTOUCH' CAUSES
C       THE HAND TO DESCEND WHILE CONSTANTLY MONITORING THE SENSOR
C       INPUT TO SEE IF THE THRESHOLD VALUE HAS BEEN REACHED.  WHEN
C       THE THRESHOLD IS REACHED THE HAND STOPS.  THE CALL TO 'STLINE'
C       IS USED TO MOVE THE HAND BACK UP ONE CENTIMETER TO MAKE
C       CERTAIN THAT THE HAND DOES NOT COLLIDE WITH THE TOP OF THE
C       STACK DURING THE SEARCH OPERATION.  THE HAND IS CENTERED OVER
C       THE TOP OF THE STACK BY A CALL TO 'BAL'.  THE CALL TO 'RELEAS'
C       OPENS THE FINGERS AND THE CALL TO 'STLINE' CAUSES THE HAND
C       TO DESCEND AROUND THE OBJECT.  THE FINGERS CLOSE ON THE OBJECT
C       DUE TO THE CALL TO 'GRASP'.  THE HAND IS RETURNED TO ITS INITIAL
C       STARTING POINT BY A CALL TO 'STLINE' .
C
7       CALL GRASP(0)
        IF(EABORT.LT.-2000)GOTO 40
        CALL PTOUCH
        IF(EABORT.LT.-2000)GOTO 40
        CALL  BAL
        IF(EABORT.LT.-2000)GOTO 40
        CALL RELEAS(0)
        IF(EABORT.LT.-2000)GOTO 40
        CALL STLINE(0,0,-600,1)
        IF(EABORT.LT.-2000)GOTO 40
        CALL GRASP(0)
        IF(EABORT.LT.-2000)GOTO 40
        OFF(3)=OFF(3)-600
        CALL STLINE(-OFF(1),-OFF(2),-OFF(3),1)
        GOTO 435
C
C   THE 'EDGE' COMMAND
C
10      RP=1
740     CALL EDGE(0)
        CALL WAIT(32)
```

```
        IF(RP.NE.2)GOTO 738
        OFF(2)=OFF(2)-INBUF(36)
        CALL STLINE(0,0,-INBUF(36),1)
738     CALL EDGE(RP)
        CALL STLINE(0,0,500,1)
        OFF(3)=OFF(3)+500
        CALL STLINE(-OFF(1),-OFF(2),-OFF(3),1)
        IF(INBUF(31).GT.-2000)GOTO 435
        IF(RP.EQ.2)GOTO 435
        RP=2
        GOTO 740
C
C   THE 'TOUCH' FUNCTION IS ACCOMPLISHED BY A CALL TO 'PTOUCH'.
C
8       CALL PTOUCH
110     IF(EABORT.LT.-2000)GOTO 40
C
C   THE FOLLOWING CODE RE-DEFINES THE 'DESTINATION' OF THE ELEMENTAL
C       MOVE' COMMAND AS THE PRESENT LOCATION OF THE ARM AND ENTERS
C       THE PRESENT JOINT POSITION VALUES INTO THE LOCATION
C       TABLE UNDER THIS (ELE(2)) DESTINATION NAME.
C
        DO 420 J=1,6
        AC(J)=JPOS(J)
420     LTAB(ELE(2),1,J)=JPOS(J)
C
C   THE START OF THE APPROACH PATH MUST BE CALCULATED FROM THIS RE-DEFINED
C       LOCATION AND ITS JOINT POSITION VALUES ENTERED UNDER THIS LOCATION
C       NAME IN THE LOCATION TABLE.
C
        AX=LTAB(ELE(2),3,4)/100.
        AY=LTAB(ELE(2),3,5)/100.
        AZ=LTAB(ELE(2),3,6)/100.
        CALL COOR(AX,AY,AZ,0,0)
        DO 430 J=1,6
        LTAB(ELE(2),4,J)=JPOS(J)
430     JPOS(J)=LTAB(ELE(2),1,J)
C
C   IF A VOLTAGE LEVEL IS TO BE SENT OUT ON A SPECIFIED CHANNEL (THIS
C       ALLOWS THE ARM TO BE INTERLOCKED WITH AN EXTERNAL DEVICE) THEN
C       THE CHANNEL NUMBER IS DECODED AND A SIGNAL SENT OUT.
C
435     SEN=ELE(4)/100
        IF(SEN.EQ.0)GOTO 440
        CALL SERVO(SEN)
C
C   IF THE ARM IS TO WAIT AT ITS PRESENT LOCATION UNTIL AN INPUT CHANNEL
C       RECEIVES A VOLTAGE LEVEL (THIS ALLOWS THE ARM TO BE SYNCHRONIZED
C       WITH AN EXTERNAL DEVICE) THEN THE INPUT CHANNEL NUMBER IS DECODED
C       AND THE CONTROL SYSTEM WAITS UNTIL THIS CHANNEL RECEIVES A
C       VOLTAGE LEVEL BEFORE CONTINUING.
C
440     WA=ELE(4)-SEN*100
        IF(WA.EQ.0)GOTO 40
        CALL WAIT(WA)
40      RETURN
```


STLINE
ENTER
USE PRESENT JOINT POSITION VALUES AS REFERENCE LOCATION FOR COORDINATE TRANSFORMATION ROUTINE
DELTA COORDINATE DISTANCES GIVEN ?
NO
YES
CALL COOR
TO CALCULATE THE DELTA X,Y,Z DISTANCES FROM PRESENT LOCATION TO THE DESTINATION POINT
CALCULATE STRAIGHT LINE DISTANCE (IN CENTIMETERS) TO THE DELTA X,Y,Z DISTANCES
CALCULATE THE NUMBER OF INTERMEDIATE LOCATION POINTS SPACED EVERY 2 CENTIMETERS ALONG STRAIGHT LINE PATH
CALL COOR
TO DEVELOP A TABLE OF JOINT POSITION VALUES FOR THESE INTERMEDIATE LOCATION POINTS
ALL INTERMEDIATE STEPS EXECUTED ?
YES
NO
GENERATE INTERPOLATED TRAJECTORY SEGMENT FROM PRESENT POSITION TO NEXT INTERMEDIATE LOCATION POINT
RETURN

```
C --- SUBROUTINE : STLINE
C
C --- ARGUMENTS : DX,DY,DZ        = A STRAIGHT LINE TRAJECTORY IS TO BE
C                                   GENERATED FROM THE PRESENT LOCATION OF
C                                   THE ARM TO A POSITION A DELTA 'X'(DX),
C                                   DELTA 'Y' (DY), DELTA 'Z' (DZ) DISTANCE
C                                   AWAY.
C                 I         IF(I=0)  THEN THE STRAIGHT LINE IS TO BE
C                                   GENERATED FROM THE PRESENT LOCATION
C                                   TO THE DESTINATION POINT (ELE(2))
C                                   SPECIFIED IN THE 'ELEMENTAL MOVE'
C                                   COMMAND.
C                           IF(I=1)  THEN THE STRAIGHT LINE TRAJECTORY
C                                   IS TO BE GENERATED FROM THE PRESENT
C                                   POSITION OF THE ARM TO A POINT 'DX',
C                                   'DY', 'DZ' DISTANCE AWAY AS SPECIFIED
C                                   BY THE ARGUMENTS OF THE SUBROUTINE
C                                   CALL.
C
C --- CALLED BY : EMOVE
C
C --- CALLS SUBROUTINES : COOR  SQRT  SERVO
C
C --- INPUT DATA : DX,DY,DZ,I     = THE ARGUMENTS AS DESCRIBED ABOVE.
C                  JPOS(1-6)      = THE VALUE OF THE JOINT POSITION
C                                   INDICATORS.
C                  ELE(2)         = THE DESTINATION POINTER VALUE FOR
C                                   THE PRESENT 'ELEMENTAL MOVE'
C                                   COMMAND.
C                  INBUF(36 )     = THE VALUE OF A POTENTIOMETER
C                                   WHICH IS USED TO CONTROL THE SPEED
C                                   OF THE STRAIGHT LINE TRAJECTORY.
C
C --- OUTPUT DATA : JPOS(1-6)      = THE JOINT POSITION COMMAND VALUES
C                                    SENT TO 'SERVO' TO CAUSE THE HAND
C                                    TO MOVE ALONG THE CALCULATED STRAIGHT
C                                    LINE TRAJECTORY.
C
C --- FUNCTION: TO GENERATE A STRAIGHT LINE TRAJECTORY FROM THE PRESENT
C               POSITION OF THE HAND TO A SPECIFIED LOCATION.
C
      SUBROUTINE STLINE(DX,DY,DZ,I)
      IMPLICIT INTEGER(B-R)
      COMMON/ARMBUF/INBUF(64),OUTBUF(64)
      COMMON/LMOD/LTAB(90,4,7),PRES,DEST
      COMMON/OUT/JPOS(8)
      COMMON/CORT/AC(6),AXEP,AYEP,AZEP
      COMMON/EM/ELE(6)
      EQUIVALENCE(EABORT,INBUF(28))
      DIMENSION ADD(6),COTAB(50,6)
C
C   PLACE THE CURRENT JOINT POSITION VALUES IN THE COMMON VARIABLE
C               'AC(1-6)'.  THE POSITION OF THE ARM DESCRIBED BY THESE
C               VALUES WILL BE USED AS THE STARTING POINT OF THE
C               TRAJECTORY.  THE SUBROUTINE 'COOR' WILL CALCULATE
C               THE CORRECT JOINT POSITION VALUES OF NEW POSITIONS THAT
```

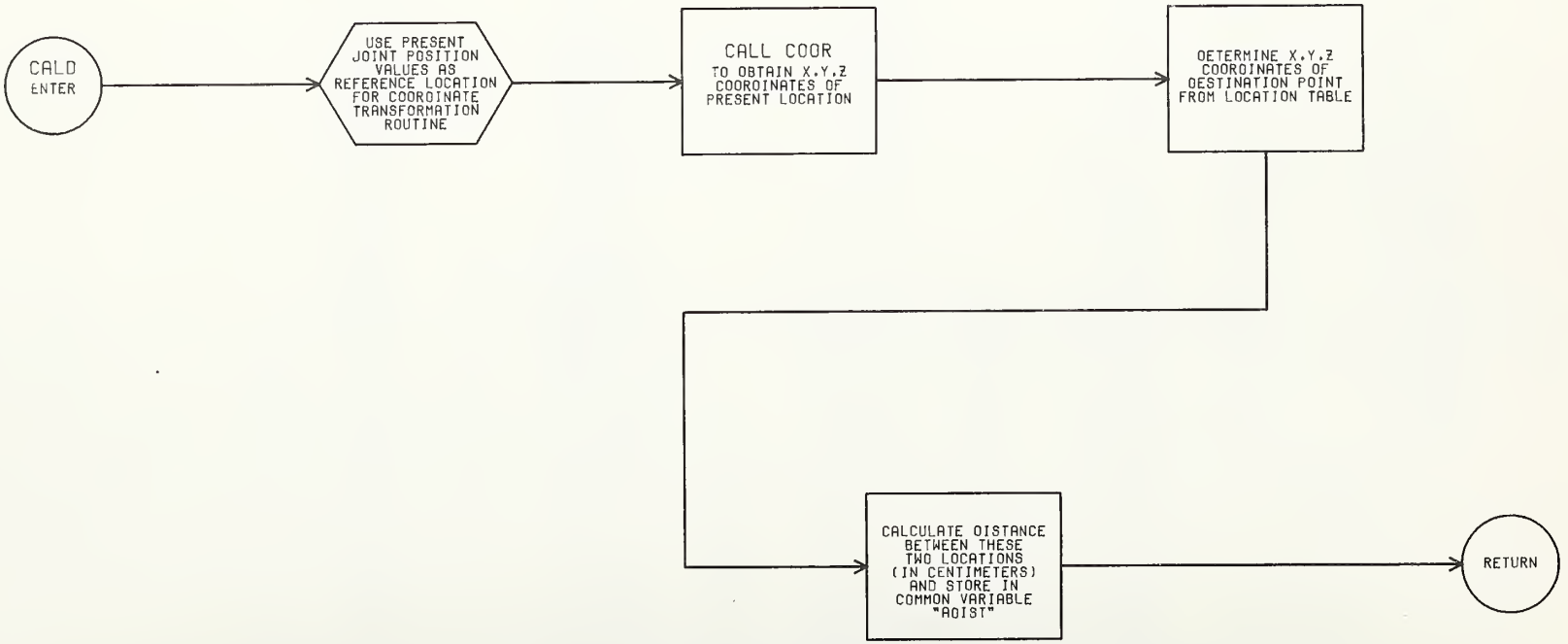
```
C                ARE SPECIFIED BY DELTA 'X', 'Y', 'Z' DISTANCES
C                AWAY FROM THIS REFERENCE POSITION ('AC(1-6)').
C
         DO 15 J=1,6
15       AC(J)=JPOS(J)
C
C    THE ARGUMENT 'I' IS TESTED IN ORDER TO BRANCH TO THE PROPER
C        CODE.
C
         IF(I.EQ.0)GOTO 10
C
C    FOR (I=1), THE END POINT OF THIS STRAIGHT LINE TRAJECTORY IS DEFINED
C        AS A POINT 'DX', 'DY', 'DZ' DISTANCE FROM THE PRESENT POSITION.
C        THESE DELTA DISTANCES ARE CONVERTED TO CENTIMETERS BY DIVIDING BY
C        100 AND THEN STORED IN THE VARIABLES 'AX', 'AY', 'AZ'.
C
         AX=DX/100.
         AY=DY/100.
         AZ=DZ/100.
C
C    ONCE THE VALUES 'AX', 'AY', 'AZ' HAVE BEEN DETERMINED, THEN BRANCH
C        TO STATEMENT 20.
C
         GOTO 20
C
C    FOR (I=0), THE END POINT OF THIS STRAIGHT LINE TRAJECTORY IS DEFINED
C        BY THE DESTINATION POINTER ('ELE(2)') FOR THE PRESENT 'ELEMENTAL
C        MOVE' COMMAND.  THE DELTA 'X', 'Y', 'Z' DISTANCE OF THIS
C        DESTINATION FROM THE PRESENT LOCATION IS CALCULATED BY, FIRST,
C        CALCULATING THE 'X,Y,Z' COORDINATES OF THE PRESENT LOCATION BY
C        A CALL TO 'COOR'.  THE 'X,Y,Z' COORDINATES OF THE BEGINNING OF
C        THE APPROACH PATH ARE OBTAINED FROM THE LOCATION TABLE MODULE
C        (IE.  THE COORDINATES (LTAB(ELE(2),2,1-3)) OF THE END POINT OF THE
C        DESTINATION ARE ADDED TO THE DELTA OFFSETS (LTAB(ELE(2),3,4-6))
C        OF THE START OF THE APPROACH PATH FROM THIS END POINT,
C        THUS, OBTAINING THE COORDINATES 'X,Y,Z' OF THE START OF THE
C        APPROACH PATH).  THE DIFFERENCE BETWEEN THESE VALUES AND THE
C        COORDINATES FOR THE PRESENT LOCATION ARE STORED IN 'AX',
C        'AY', 'AZ'.
C
10       CALL COOR(0.,0.,0.,0,1)
         AX=(LTAB(ELE(2),2,1)+LTAB(ELE(2),3,4))/100.-AXEP
         AY=(LTAB(ELE(2),2,2)+LTAB(ELE(2),3,5))/100.-AYEP
         AZ=(LTAB(ELE(2),2,3)+LTAB(ELE(2),3,6))/100.-AZEP
C
C    THE DISTANCE OF THE STRAIGHT LINE PATH IS CALCULATED.
C
20       DV=SQRT(AX**2+AY**2+AZ**2)
C
C    IF THE DISTANCE OF THE TRAJECTORY IS ZERO, THEN NO FURTHER
C        COMPUTATIONS ARE MADE AND THE CONTROL IS RETURNED TO THE
C        CALLING PROGRAM.
C
         IF(DV.EQ.0)GOTO 40
C
C    THE STRAIGHT LINE TRAJECTORY IS TO BE 'DV' CENTIMETERS LONG.  THIS
```

```
C       TRAJECTORY WILL BE GENERATED BY FIRST, CALCULATING A NUMBER OF
C       INTERMEDIATE LOCATIONS THAT LIE ALONG THIS STRAIGHT LINE AND THEN
C       CALCULATING THE ADDITIONAL POINTS BY INTERPOLATING BETWEEN THESE
C       INTERMEDIATE LOCATIONS.  THE INTERMEDIATE LOCATIONS WILL LIE
C       APPROXIMATELY TWO CENTIMETERS APART ALONG THE STRAIGHT LINE.
C       THE NUMBER OF THESE INTERMEDIATE LOCATIONS IS CALCULATED BY
C       DIVIDING THE LENGTH OF THE TRAJECTORY ('DV') BY TWO.  THIS VALUE
C       IS INCREMENTED BY ONE TO MAKE CERTAIN THAT THE NUMBER OF POINTS IS
C       A NON-ZERO VALUE IF THE TRAJECTORY IS LESS THAN TWO CENTIMETERS
C       LONG.
C
        NDV=DV/2+1
C
C   THE DELTA 'X,Y,Z' COORDINATE VALUES BETWEEN EACH OF THE INTERMEDIATE
C       LOCATIONS IS CALCULATED AND STORED IN 'ANX', 'ANY', 'ANZ'.
C
        ANX=AX/NDV
        ANY=AY/NDV
        ANZ=AZ/NDV
C
C   THE JOINT POSITION VALUES OF THE INTERMEDIATE LOCATIONS ARE CALCULATED
C       BY CALLS TO 'COOR' AND ARE STORED IN SEQUENCE IN THE TABLE
C       'COTAB(50,6)'.
C
        KD=NDV+1
        DO 110 J=1,KD
        KLL=J-1
        ATX=KLL*ANX
        ATY=KLL*ANY
        ATZ=KLL*ANZ
        CALL COOR(ATX,ATY,ATZ,0,0)
        DO 110 KB=1,6
110     COTAB(J,KB)=JPOS(KB)
C
C   THE VALUE OF THE POTENTIOMETER (INBUF(36)) IS READ IN AND USED TO
C       DEFINE THE NUMBER OF ADDITIONAL INTERPOLATION POINTS ('PC') TO BE
C       CALCULATED BETWEEN EACH PAIR OF INTERMEDIATE LOCATIONS.
C
        PC=10-INBUF(36)/30
        ASP=PC
C
C   A LOOP IS SET UP TO PROVIDE THE INTERPOLATED JOINT POSITION COMMANDS
C       FOR ALL 'NDV' OF THE INTERMEDIATE LOCATIONS.
C
        DO 150 P=1,NDV
C
C   FOR EACH INTERMEDIATE LOCATION, THE DELTA JOINT VALUES FOR THE
C       SPECIFIED NUMBER ('ASP') OF INTERPOLATION POINTS IS CALCULATED
C       AND STORED IN 'ADD(1-6)'.
C
        DO 140 MP=1,6
140     ADD(MP)=(COTAB(P+1,MP)-COTAB(P,MP))/ASP
C
C   THIS LOOP CAUSES THE ARM TO MOVE FROM ONE INTERMEDIATE LOCATION TO
C       THE NEXT BY INCREMENTING THE JOINT POSITION COMMANDS BY THE
C       PROPER DELTA INTERPOLATION VALUE ('ADD(1-6)') EACH TIME.
```

```
C
      DO 150 LM=1,PC
      DO 160 JB=1,6
160   JPOS(JB)=COTAB(P,JB)+LM*ADD(JB)
      CALL SERVO(0)
      IF(EABORT.LT.-2000)GOTO 40
150   CONTINUE
40    RETURN
      END
```

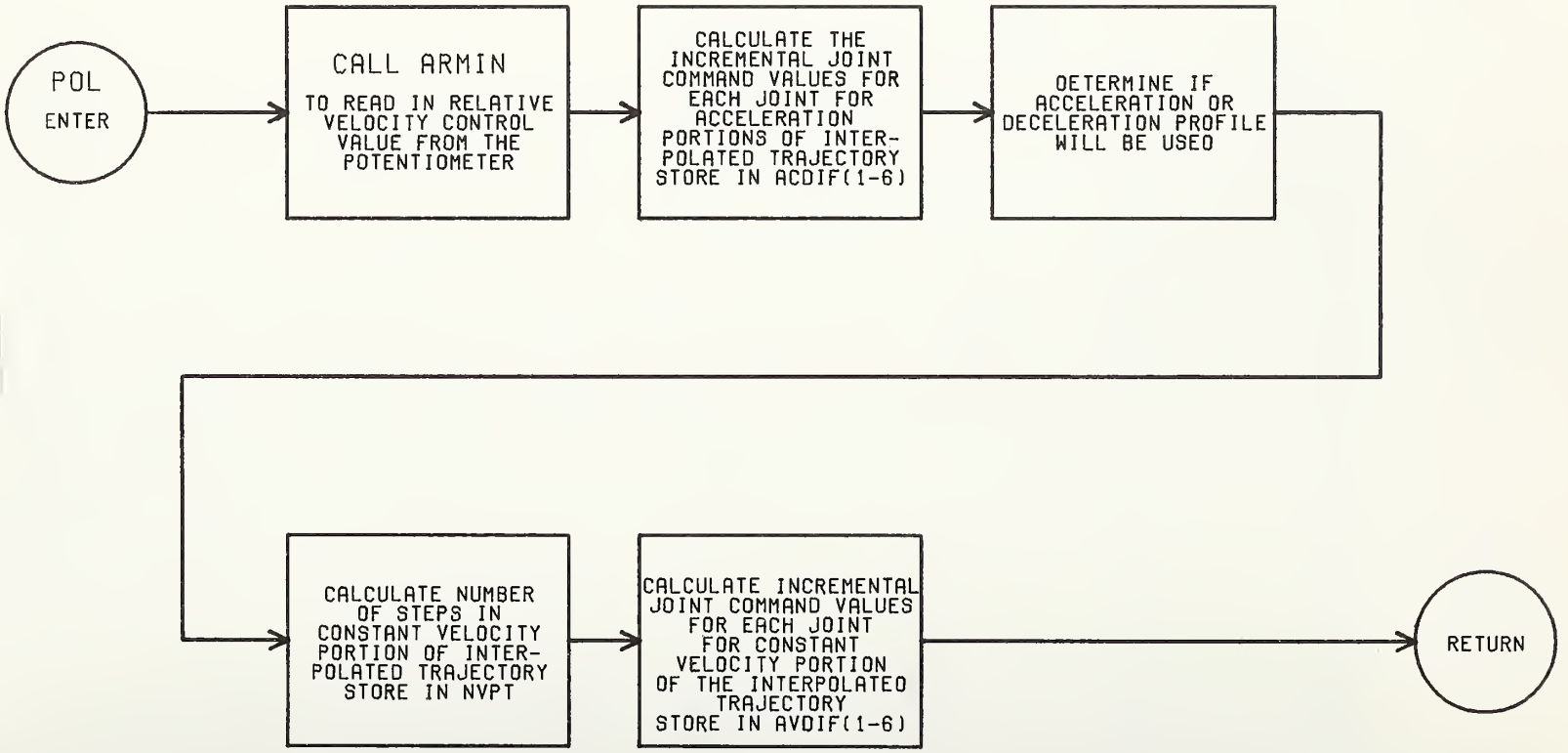
WAIT
ENTER
CALL ARM1N
TO READ IN
VALUE OF SIGNAL
ON SPECIFIED
CHANNEL
IS THIS
VALUE ZERO
?
YES
NO
RETURN

```
C --- SUBROUTINE : WAIT
C
C --- ARGUMENTS : CH         =  THE INPUT CHANNEL NUMBER THAT IS TO
C                                 SAMPLED FOR A VOLTAGE LEVEL.
C
C --- CALLED BY : EXPRO  EMOVE
C
C --- CALLS SUBROUTINES : ARMIN
C
C --- INPUT DATA : CH        = THE ARGUMENT AS EXPLAINED ABOVE.
C
C --- OUTPUT DATA :
C
C --- FUNCTION: SUSPENDS OPERATION OF THE CONTROL SYSTEM UNTIL A
C               SPECIFIED INPUT CHANNEL DROPS TO A ZERO VOLTAGE
C               LEVEL OR THE ABORT SWITCH IS THROWN.
C
        SUBROUTINE WAIT(CH)
        IMPLICIT INTEGER(B-Z)
        COMMON/ARMBUF/INBUF(64),OUTBUF(64)
        EQUIVALENCE(EABORT,INBUF(28))
C
C   THE INPUT CHANNELS ARE READ IN, AND THE VALUE OF THE SPECIFIED
C       CHANNEL ('CH')  IS TESTED TO SEE IF IT IS A POSITIVE OR NEGATIVE
C       VALUE, INDICATING THAT THE PROGRAM IS TO WAIT UNTIL THE
C       SPECIFIED INPUT CHANNEL RECEIVES A HIGH
C       OR A ZERO VOLTAGE LEVEL, RESPECTIVELY.
C
20      CALL ARMIN
        IF(EABORT.LT.-2000)GOTO 40
        IF(CH.LT.0)GOTO 25
C
C   THE VALUE OF THE SPECIFIED INPUT
C       CHANNEL IS TESTED.  AS LONG AS THERE IS ZERO VOLTAGE ON THIS
C       CHANNEL, THE SUBROUTINE WILL CONTINUE SAMPLING IT.  WHEN THE
C       VOLTAGE LEVEL GOES ABOVE ZERO, CONTROL IS RETURNED TO THE CALLING
C       PROGRAM.
C
        IF(INBUF(CH).GT.-2000)GOTO 20
        GOTO 40
C
C   THE VALUE OF THE SPECIFIED INPUT
C       CHANNEL IS TESTED.  AS LONG AS THERE IS A VOLTAGE LEVEL ON THIS
C       CHANNEL, THE SUBROUTINE WILL CONTINUE SAMPLING IT.  WHEN THE
C       VOLTAGE LEVEL DROPS TO ZERO, CONTROL IS RETURNED TO THE CALLING
C       PROGRAM.
C
25      IF(INBUF(-CH).LT.-2000)GOTO 20
40      RETURN
        END
```


CALD
ENTER
USE PRESENT JOINT POSITION VALUES AS REFERENCE LOCATION FOR COORDINATE TRANSFORMATION ROUTINE
CALL COOR
TO OBTAIN X,Y,Z COORDINATES OF PRESENT LOCATION
DETERMINE X,Y,Z COORDINATES OF DESTINATION POINT FROM LOCATION TABLE
CALCULATE DISTANCE BETWEEN THESE TWO LOCATIONS (IN CENTIMETERS) AND STORE IN COMMON VARIABLE "ADIST"
RETURN

```
C --- SUBROUTINE : CALD
C
C --- ARGUMENTS :
C
C --- CALLED BY : EMOVE
C
C --- CALLS SUBROUTINES : COOR  SQRT
C
C --- INPUT DATA : JPOS(1-6)    = THE PRESENT JOINT POSITION VALUES.
C                  ELE(2)       = THE VALUE OF THE DESTINATON POINTER
C                                 TO THE LOCATION TABLE.
C                  AXEP         = THE 'X' COORDINATE VALUE FOR THE HAND
C                                 FOR THE SPECIFIED JOINT VALUES (AC(1-6))
C                                 [FROM 'COOR'].
C                  AYEP         = THE 'Y' COORDINATE VALUE FOR THE HAND
C                                 FOR THE SPECIFIED JOINT VALUES (AC(1-6))
C                                 [FROM 'COOR'].
C                  AZEP         = THE 'Z' COORDINATE VALUE FOR THE HAND
C                                 FOR THE SPECIFIED JOINT VALUES (AC(1-6))
C                                 [FROM 'COOR'].
C
C --- OUTPUT DATA : AC(1-6)        = THE JOINT POSITION VALUES TO BE USED
C                                    BY THE COORDINATE TRANSFORMATION
C                                    ROUTINE ('COOR').
C                   ADIST         = THE DISTANCE IN CENTIMETERS BETWEEN
C                                    THE PRESENT LOCATION OF THE ARM AND
C                                    THE SPECIFIED DESTINATION
C                                    POINT (ELE(2)).
C
C --- FUNCTION: CALCULATES THE DISTANCE (IN CENTIMETERS) FROM THE PRESENT
C               LOCATION OF THE END POINT OF THE HAND TO ITS POSITION
C               AT THE DESTINATION POSITION.
C
      SUBROUTINE CALD
      IMPLICIT INTEGER(B-R)
      COMMON/LMOD/LTAB(90,4,7),PRES,DEST
      COMMON/OUT/JPOS(8)
      COMMON/EM/ELE(6)
      COMMON/DIST/ADIST
      COMMON/CORT/AC(6),AXEP,AYEP,AZEP
C
C   STORE THE JOINT POSITION VALUES IN THE VARIABLE (AC(1-6))
C      SO THAT THE 'X', 'Y', 'Z' VALUES FOR THIS POINT CAN BE
C      CALCULATED BY THE COORDINATE TRANSFORMATION ROUTINE.
C
      DO 20 J=1,6
20    AC(J)=JPOS(J)
C
C   THE CALL TO 'COOR(X,Y,Z,D,J)' IS MADE WITH 'J'=1 TO INDICATE
C      THAT ONLY THE 'X','Y','Z' VALUES ARE TO BE RETURNED (IE. THE
C      JOINT VALUES ARE NOT TO BE CALCULATED).
C
      CALL COOR(0.,0.,0.,0,1)
C
C   THE 'X,Y,Z' VALUES OF THE PRESENT LOCATION OF THE ARM ARE STORED
C      IN THE VARIABLES 'APX', 'APY', AND 'APZ'.
```

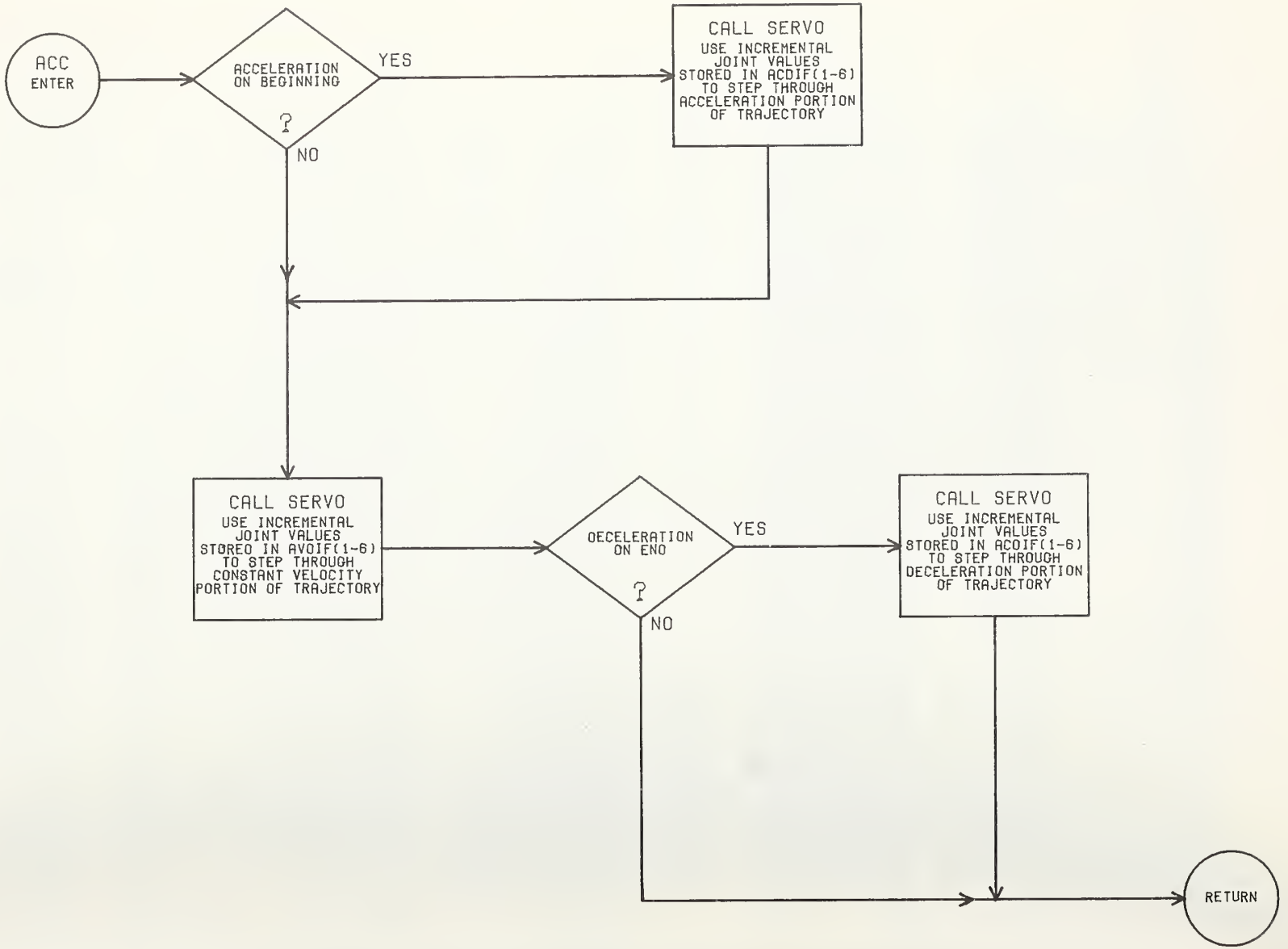
```
C
      APX=AXEP
      APY=AYEP
      APZ=AZEP
C
C   CALCULATE THE 'X,Y,Z' VALUES OF THE DESTINATION POINT (ELE(2)) BY
C      PLACING ITS JOINT POSITION VALUES IN THE COMMON VARIABLE
C      AC(1-6) AND CALLING 'COOR'.
C
      DO 21 J=1,6
21    AC(J)=LTAB(ELE(2),4,J)
      CALL COOR(0.,0.,0.,0,1)
C
C   THE DISTANCE BETWEEN THESE TWO POINTS IS CALCULATED AND STORED IN
C      THE COMMON VARIABLE 'ADIST'.
C
      ADX=APX-AXEP
      ADY=APY-AYEP
      ADZ=APZ-AZEP
      ADIST=SQRT(ADX**2+ADY**2+ADZ**2)
40    RETURN
      END
```


POL
ENTER
CALL ARMIN
TO READ IN RELATIVE VELOCITY CONTROL VALUE FROM THE POTENTIOMETER
CALCULATE THE INCREMENTAL JOINT COMMAND VALUES FOR EACH JOINT FOR ACCELERATION PORTIONS OF INTER-POLATED TRAJECTORY STORE IN ACDIF(1-6)
DETERMINE IF ACCELERATION OR DECELERATION PROFILE WILL BE USED
CALCULATE NUMBER OF STEPS IN CONSTANT VELOCITY PORTION OF INTER-POLATED TRAJECTORY STORE IN NVPT
CALCULATE INCREMENTAL JOINT COMMAND VALUES FOR EACH JOINT FOR CONSTANT VELOCITY PORTION OF THE INTERPOLATED TRAJECTORY STORE IN AVDIF(1-6)
RETURN

```
C --- SUBROUTINE : POL
C
C --- ARGUMENTS :
C
C --- CALLED BY : EMOVE
C
C --- CALLS SUBROUTINES : ARMIN
C
C --- INPUT DATA : INBUF(26)      = THE VALUE OF A POTENTIOMETER USED TO
C                                   SPECIFY THE VELOCITY OF THIS TRAJECTORY
C                                   BY DEFINING THE NUMBER OF INTERPOLATION
C                                   POINTS TO BE USED PER UNIT DISTANCE.
C                  ADIST          = THE DISTANCE BETWEEN THE TWO END POINTS
C                                   OF THE TRAJECTORY IN CENTIMETERS
C                                   [FROM 'CALD'].
C                  ELE(1)         = THE LOCATION TABLE POINTER FOR THE
C                                   PRESENT POSITION OF THE ARM.
C                  ELE(2)         = THE LOCATION TABLE POINTER FOR THE
C                                   DESTINATION OF THIS TRAJECTORY.
C                  JPOS(1-6)      = THE JOINT POSITION VALUES OF THE
C                                   LOCATION OF THE ARM.
C                  LTAB(ELE(2),4,1-6)= THE JOINT POSITION VALUES OF THE
C                                   START OF THE APPROACH PATH FOR THE
C                                   SPECIFIED DESTINATION POINT.
C                  LTAB(ELE(1),2,7)= THE FLAG THAT INDICATES IF THE ARM
C                                   HAS STOPPED AT THE PRESENT LOCATION.
C                  LTAB(ELE(2),2,7)= THE FLAG THAT INDICATES IF THE ARM
C                                   IS TO STOP AT THE DESTINATION POINT.
C
C --- OUTPUT DATA : NACV          = THIS IS THE NUMBER OF COMMANDED
C                                   JOINT POSITION OUTPUTS FOR THE
C                                   ACCELERATION (OR DECCELERATION) PORTION
C                                   OF THE TRAJECTORY.
C                  NVPT           = THIS IS THE NUMBER OF COMMANDED JOINT
C                                   POSITION OUTPUTS FOR THE CONSTANT
C                                   VELOCITY PORTION OF THE TRAJECTORY.
C                  KJ             = A FLAG THAT CODES THE INFORMATION OF
C                                   WHETHER AN ACCELERATION OR DECCELERATION
C                                   PROFILE IS TO BE USED.
C                  ACDIF(1-6)     = THE DELTA JOINT POSITION VALUES THAT
C                                   WOULD CAUSE THE ARM TO MOVE 1/400 OF
C                                   A CENTIMETER ALONG THE INTERPOLATED
C                                   TRAJECTORY.  USED AS THE ACCELERATION
C                                   FACTOR FOR THE ACCELERATION PORTION OF
C                                   THE TRAJECTORY.
C                  AVDIF(1-6)     = THE DELTA JOINT POSITION VALUES THAT
C                                   WOULD CAUSE THE ARM TO MOVE 'NACV'
C                                   TIMES 1/400 CENTIMETERS ALONG THE
C                                   CONSTANT VELOCITY PORTION OF THE
C                                   TRAJECTORY.  THUS, PROVIDING THE PROPER
C                                   SIZE STEP REQUIRED FOR THE ARM TO MOVE
C                                   AT THE SPECIFIED VELOCITY.
C
C --- FUNCTION: GENERATES THE DELTA JOINT POSITION VALUES AND SPECIFIES
C              THE NUMBER OF THEM REQUIRED TO CAUSE THE ARM TO MOVE
C              THROUGH THE TRAJECTORY AT THE CORRECT VELOCITY.  THE
```

```
C               ACTUAL EXECUTION IS ACCOMPLISHED BY THE SUBROUTINE 'ACC'
C               WHICH USES THIS INFORMATION FROM 'POL'.
C
        SUBROUTINE POL
        IMPLICIT INTEGER(B-Z)
        COMMON/ARMBUF/INBUF(64),OUTBUF(64)
        COMMON/LMOD/LTAB(90,4,7),PRES,DEST
        COMMON/OUT/JPOS(8)
        COMMON/INT/NACV,NVPT,KJ
        COMMON/DELT/ACDIF(6),AVDIF(6)
        COMMON/DIST/ADIST
        COMMON/EM/ELE(6)
        EQUIVALENCE(EABORT,INBUF(28))
C
C   THE POTENTIOMETER VALUE THAT WILL CONTROL THE VELOCITY IS READ
C       IN ON 'INBUF(26)'.  IT IS NORMALIZED TO A VALUE BETWEEN
C       ONE AND TEN AND MULTIPLIED BY THE SPECIFIED VELOCITY 'ELE(3)'
C       FOR THIS PARTICULAR TRAJECTORY.  THE VALUE 'NACV' IS SET
C       EQUAL TO THIS NUMBER DIVIDED BY 5.  THIS NUMBER SPECIFIES HOW
C       MANY INCREMENTAL STEPS ('ACDIF(1-6)') ARE TO BE SENT OUT
C       AT A TIME TO THE SERVOS IN ORDER TO CAUSE THE ARM TO MOVE AT THE
C       CORRECT VELOCITY.
C
        CALL ARMIN
        NACV=((-INBUF(26)-2100)/100+1)*ELE(3)
        NACV=NACV/10
C
C   THE VALUE 'ANTRJP' IS SET EQUAL TO 100 TIMES THE TOTAL DISTANCE OF
C       THE TRAJECTORY IN CENTIMETERS.  THE DIFFERENTIAL JOINT VALUES FROM
C       THE PRESENT LOCATION TO THE DESTINATION ('LTAB(ELE(2),4,1-6)')
C       ARE DIVIDE BY 'ANTRJP'.  THIS CREATES THE DELTA JOINT VALUES THAT
C       THAT WOULD CAUSE A MOVEMENT OF THE ARM OF 1/100 OF A CENTIMETER.
C
        ANTRJP=ADIST*100.
        DO 20 H=1,6
20      ACDIF(H)=(LTAB(ELE(2),4,H)-JPOS(H))/ANTRJP
C
C   THE POSITION LTAB(J,2,7), CARRIES A FLAG THAT INDICATES IF THE ARM
C       IS TO STOP AT THAT LOCATION.  IF IT IS A ONE THEN THE ARM STOPS,
C       THEREFORE, AN ACCELERATION PROFILE IS NEEDED FOR THAT POINT.
C       IF IT IS ZERO, IT DOES NOT STOP AND NO ACCELERATION PROFILE IS
C       TO BE USED.  THE VALUE 'KJ' IS SET EQUAL TO THE SUM OF THE FLAGS
C       OF THE PRESENT LOCATION AND THE DESTINATION PLUS ONE.  THE
C       VALUE 'KJ' IS THEN USED TO POINT TO THE APPROPRIATE CODE
C       FOR THE CORRECT ACCELERATION PROFILES.
C
        KJ=2*LTAB(ELE(2),2,7)+LTAB(ELE(1),2,7)+1
        GOTO (110,111,111,112)KJ
C
C   THIS IS THE CONDITION FOR NO ACCELERATION OR DECCELERATION PROFILES.
C
110     NACCP=0
        GOTO 115
C
C   THIS IS THE CONDITION FOR EITHER AN ACCELERATION OR A DECCELERATION
C       PROFILE. 'NACCP' IS THE TOTAL NUMBER OF INCREMENTAL STEPS
```

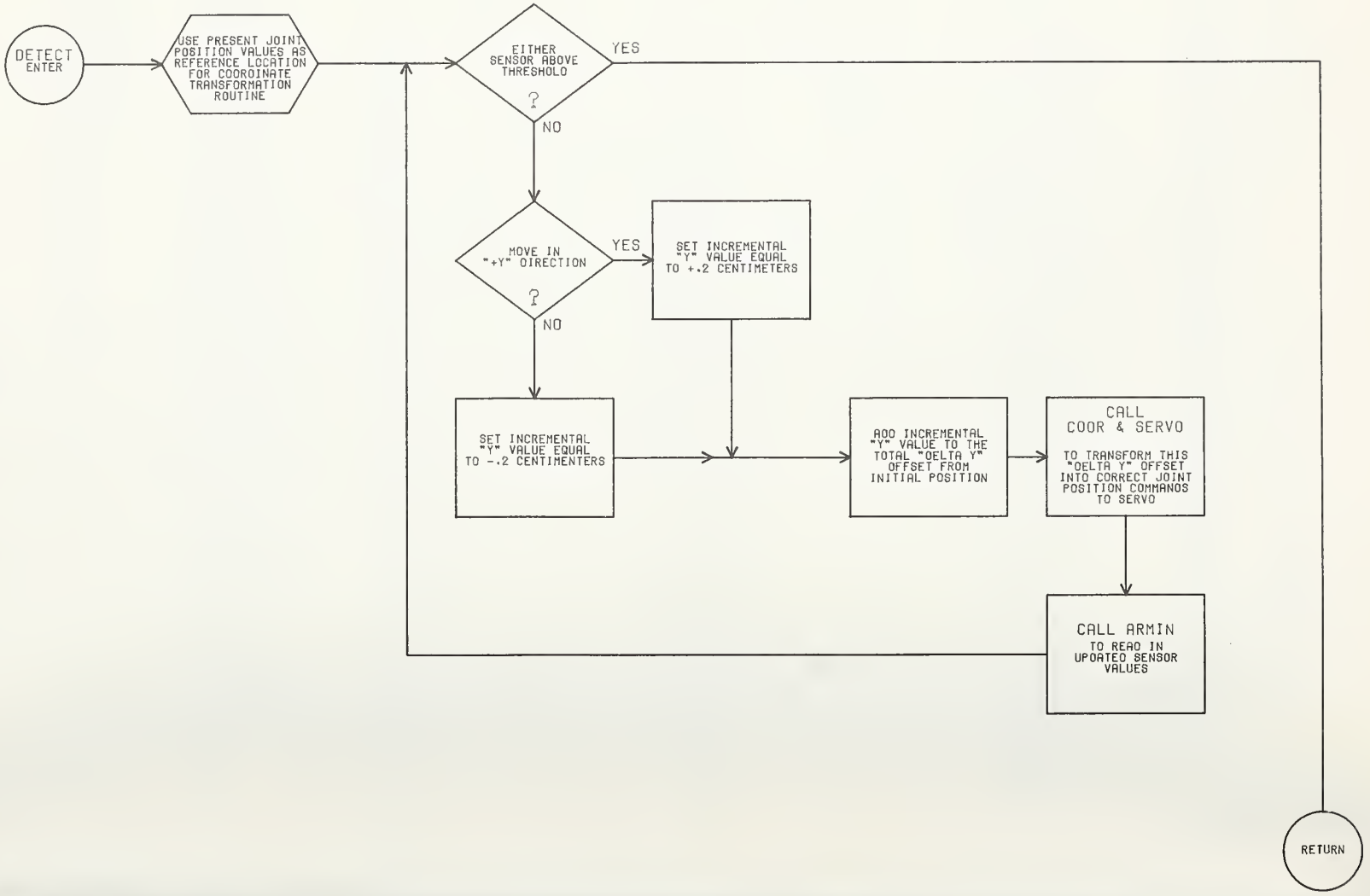
```
C       ('ACDIF(1-6)') THAT THE ARM WILL MOVE THROUGH DURING THE
C       ACCELERATION OR DECCELERATION.  THEREFORE, 'NACCP' IS A MEASURE
C       OF HOW MUCH OF THE TRAJECTORY WILL BE USED FOR THE ACCELERATION OR
C       DECCELERATION REGION.
C
111     NACCP=(NACV**2+NACV)/2
        GOTO 115
C
C   THIS IS THE CONDITION FOR BOTH AN ACCELERATION AND A DECCELERATION
C       REGION FOR THIS TRAJECTORY, THEREFORE, THERE WILL BE TWICE AS MANY
C       INCREMENTAL STEPS ('ACDIF(1-6)') INVOLVED IN THESE REGIONS.
C
112     NACCP=NACV**2+NACV
C
C   THE NUMBER OF INCREMENTAL STEPS USED IN THE ACCELERATION-DECCELERATION
C       PROFILES IS SUBTRACTED FROM THE TOTAL NUMBER OF STEPS IN THE
C       TRAJECTORY TO YIELD THE VALUE 'ANVELP'.  THIS NUMBER IS DIVIDED BY
C       'NACV' (THE NUMBER OF STEPS TO BE SENT TO THE SERVOS AT ONE
C       TIME IN ORDER TO CAUSE THE ARM TO MOVE AT THE CORRECT VELOCITY)
C       TO GIVE THE NUMBER 'NVPT' (THE NUMBER OF TIMES THAT THE DELTAS
C       'AVDIF(1-6)' ARE TO BE SENT TO THE SERVOS FOR THE CONSTANT
C       VELOCITY PORTION OF THE TRAJECTORY.
C
115     ANVELP=ANTRJP-NACCP
        NVPT=ANVELP/NACV
        IF(NVPT.NE.0)GOTO 47
        NVPT=1
47      AMV=ANVELP/NVPT
        DO 200 J=1,6
200     AVDIF(J)=ACDIF(J)*AMV
C       WRITE(6,320)NACV,NACCP,ANTRJP,ANVELP,AMV
320     FORMAT(2X,2(1X,I6),3(1X,F10.4))
C       WRITE(6,321)ADIST,ELE(3),INBUF(26)
321     FORMAT(2X,F10.4,2X,I5,2X,I6)
40      RETURN
        END
```


ACC
ENTER
ACCELERATION
ON BEGINNING
?
YES
NO
CALL SERVO
USE INCREMENTAL
JOINT VALUES
STORED IN ACDIF(1-6)
TO STEP THROUGH
ACCELERATION PORTION
OF TRAJECTORY
CALL SERVO
USE INCREMENTAL
JOINT VALUES
STORED IN AVDIF(1-6)
TO STEP THROUGH
CONSTANT VELOCITY
PORTION OF TRAJECTORY
DECELERATION
ON END
?
YES
NO
CALL SERVO
USE INCREMENTAL
JOINT VALUES
STORED IN ACDIF(1-6)
TO STEP THROUGH
DECELERATION PORTION
OF TRAJECTORY
RETURN

```
C --- SUBROUTINE : ACC
C
C --- ARGUMENTS :
C
C --- CALLED BY : EMOVE
C
C --- CALLS SUBROUTINES : SERVO
C
C --- INPUT DATA : NACV          = THIS IS THE NUMBER OF COMMANDED
C                                  JOINT POSITION OUTPUTS FOR THE
C                                  ACCELERATION (OR DECCELERATION) PORTION
C                                  OF THE TRAJECTORY.
C                   NVPT          = THIS IS THE NUMBER OF COMMANDED JOINT
C                                  POSITION OUTPUTS FOR THE CONSTANT
C                                  VELOCITY PORTION OF THE TRAJECTORY.
C                   KJ            = A FLAG THAT CODES THE INFORMATION OF
C                                  WHETHER AN ACCELERATION OR DECCELERATION
C                                  PROFILE IS TO BE USED.
C                   ACDIF(1-6)    = THE DELTA JOINT POSITION VALUES THAT
C                                  WOULD CAUSE THE ARM TO MOVE 1/200 OF
C                                  A CENTIMETER ALONG THE INTERPOLATED
C                                  TRAJECTORY.  USED AS THE ACCELERATION
C                                  FACTOR FOR THE ACCELERATION PORTION OF
C                                  THE TRAJECTORY.
C                   AVDIF(1-6)    = THE DELTA JOINT POSITION VALUES THAT
C                                  WOULD CAUSE THE ARM TO MOVE 'NACV'
C                                  TIMES 1/200 CENTIMETERS ALONG THE
C                                  CONSTANT VELOCITY PORTION OF THE
C                                  TRAJECTORY.  THUS, PROVIDING THE PROPER
C                                  SIZE STEP REQUIRED FOR THE ARM TO MOVE
C                                  AT THE SPECIFIED VELOCITY.
C
C --- OUTPUT DATA : JPOS(1-6)       = THE COMMANDED JOINT POSITION VALUES
C                                    TO THE SERVOS THAT WILL MOVE THE
C                                    ARM THROUGH THE PROPER TRAJECTORY IN
C                                    SPACE.
C
C --- FUNCTION: CALCULATES AND OUTPUTS TO 'SERVO' THE CORRECT JOINT
C               POSITION COMMANDS TO CAUSE THE ARM TO MOVE THROUGH ITS
C               SPECIFIED TRAJECTORY USING ACCELERATION AND DECELERATION
C               ROUTINES WHERE NECESSARY.
C
      SUBROUTINE ACC
      IMPLICIT INTEGER(B-Z)
      COMMON/EM/ELE(6)
      COMMON/LMOD/LTAB(90,4,7),PRES,DEST
      COMMON/ARMBUF/INBUF(64),OUTBUF(64)
      COMMON/OUT/JPOS(8)
      COMMON/INT/NACV,NVPT,KJ
      COMMON/DELT/ACDIF(6),AVDIF(6)
      EQUIVALENCE(EABORT,INBUF(28))
      DIMENSION AJPOS(6)
C
C   THE JOINT POSITION VALUES ARE CONVERTED TO FLOATING POINT VALUES,
C      SO THAT THEY CAN BE INCREMENTED BY VALUES LESS THAN ONE IF
C      NECESSARY.
```

```
C
        DO 10 K=1,6
10      AJPOS(K)=JPOS(K)
C
C   THE FLAG 'KJ' (WHOSE VALUE IS DETERMINED BY WHETHER OR NOT
C       ACCELERATION AND/OR DECCELERATION PROFILES ARE TO BE USED)
C       IS USED TO POINT TO THE APPROPRIATE CODE TO BE EXECUTED.
C
        GOTO (11,12,11,12)KJ
C
C   THIS IS THE CODE THAT CAUSES AN ACCELERATION TO BE PROVIDED AT THE
C       BEGINNING OF THE TRAJECTORY.  THE ACCELERATION LASTS FOR 'NACV'
C       OUTPUTS TO THE SERVOS.  EACH TIME THE JOINT POSITION COMMANDED
C       OUTPUT IS INCREMENTED BY A DELTA THAT IS 'ACDIF(J)' LARGER THAN
C       THE LAST ONE, THEREBY PROVIDING AN ACCELERATION UP TO
C       THE SPECIFIED VELOCITY.
C
12      DO 400 LD=1,NACV
        DO 401 J=1,6
        AJPOS(J)=AJPOS(J)+LD*ACDIF(J)
401     JPOS(J)=AJPOS(J)
        CALL SERVO(0)
        IF(EABORT.LT.-2000)GOTO 40
400     CONTINUE
C
C   THIS IS THE SECTION OF THE CODE THAT PROVIDES THE CONSTANT
C       VELOCITY PORTION OF THE TRAJECTORY.  THE JOINT POSITION COMMANDED
C       VALUES ARE INCREMENTED BY THE DELTA VALUES 'AVDIF(1-6)' FOR
C       'NVPT' TIMES TO CAUSE THE ARM TO MOVE AT A CONSTANT VELOCITY
C       THROUGH THIS PORTION OF THE TRAJECTORY.
C
11      DO 500 LD=1,NVPT
        DO 501 J=1,6
        AJPOS(J)=AJPOS(J)+AVDIF(J)
501     JPOS(J)=AJPOS(J)
        CALL SERVO(0)
        IF(EABORT.LT.-2000)GOTO 40
500     CONTINUE
C
C   THE FLAG 'KJ' IS USED TO RETURN TO THE CALLING PROGRAM ('EMOVE') IF
C       NO DECCELERATION IS TO BE USED OR EXECUTE THE NEXT SECTION OF CODE
C       IF A DECCELERATION IS CALLED FOR.
C
        GOTO (40,40,13,13)KJ
C
C   THIS SECTION OF CODE CREATES A DECCELERATION PROFILE.  THE
C       DECCELERATION LASTS FOR 'NACV' OUTPUTS TO THE SERVOS.
C       FOR EACH OUTPUT, THE JOINT POSITION COMMANDED VALUES ARE
C       INCREMENTED BY A DELTA THAT IS 'ACDIF(J)' SMALLER THAN THE LAST
C       ONE UNTIL THE DESTINATION POSITION HAS BEEN REACHED.
C
13      DO 600 LD=1,NACV
        LLD=NACV+1-LD
        DO 601 J=1,6
        AJPOS(J)=AJPOS(J)+LLD*ACDIF(J)
601     JPOS(J)=AJPOS(J)
```

```
        CALL SERVO(0)
        IF(EABORT.LT.-2000)GOTO 40
600     CONTINUE
C47     WRITE(6,48)(LTAB(ELE(2),4,J),J=1,6),(JPOS(H),H=1,6)
48      FORMAT(2X,6(2X,I6))
40      RETURN
        END
```

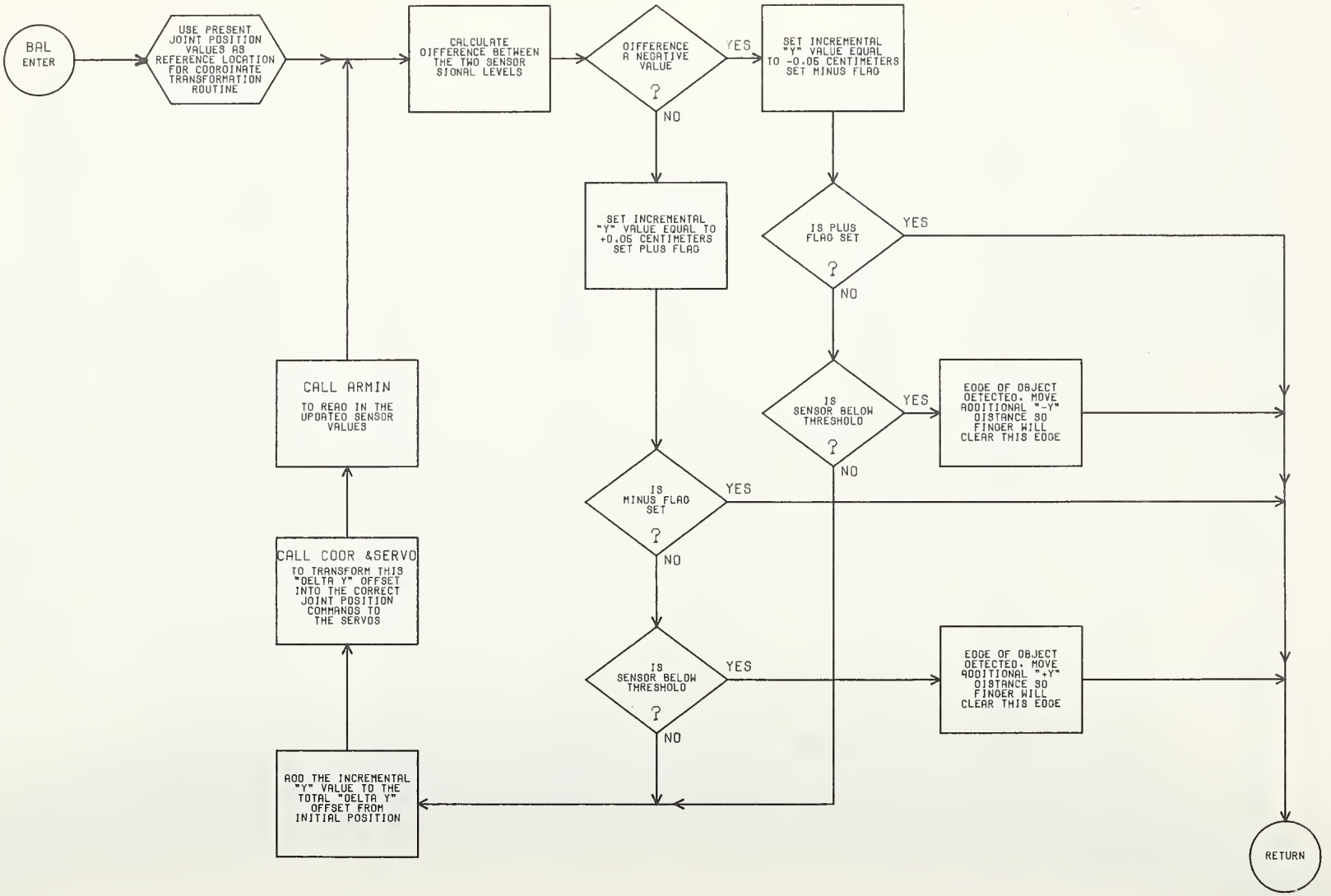

DETECT
ENTER
USE PRESENT JOINT POSITION VALUES AS REFERENCE LOCATION FOR COORDINATE TRANSFORMATION ROUTINE
EITHER SENSOR ABOVE THRESHOLD
?
YES
NO
MOVE IN "+Y" DIRECTION
?
YES
NO
SET INCREMENTAL "Y" VALUE EQUAL TO +.2 CENTIMETERS
SET INCREMENTAL "Y" VALUE EQUAL TO -.2 CENTIMENTERS
ADD INCREMENTAL "Y" VALUE TO THE TOTAL "DELTA Y" OFFSET FROM INITIAL POSITION
CALL COOR & SERVO
TO TRANSFORM THIS "DELTA Y" OFFSET INTO CORRECT JOINT POSITION COMMANDS TO SERVO
CALL ARMIN
TO READ IN UPDATED SENSOR VALUES
RETURN

```
C --- SUBROUTINE : DETECT
C
C --- ARGUMENTS :
C
C --- CALLED BY : EMOVE
C
C --- CALLS SUBROUTINES : COOR  ARMIN  SERVO
C
C --- INPUT DATA : JPOS(7)    = THE PRESENT JOINT POSITION VALUES
C                   P1            = THE PRESENT VALUE OF THE SIGNAL
C                                   FROM THE PROXIMITY SENSOR ON
C                                   ONE OF THE FINGERS [FROM 'ARMIN'
C                                   ON INPUT CHANNEL NUMBER 49].
C                  P2             = THE PRESENT VALUE OF THE SIGNAL
C                                   FROM THE PROXIMITY SENSOR ON
C                                   THE OTHER FINGER [FROM 'ARMIN'
C                                   ON INPUT CHANNEL NUMBER 48].
C
C --- OUTPUT DATA : JPOS(7)          = THE COMMANDED JOINT POSITION VALUES
C                                     THAT WILL CAUSE THE ARM TO MOVE
C                                     ALONG A SEARCH PATH IN THE 'Y'
C                                     DIRECTION [FROM 'COOR'].
C
C --- FUNCTION: TESTS THE PROXIMITY SENSORS FEEDBACK FOR A LARGE ENOUGH
C               SIGNAL TO INDICATE THAT THERE IS AN OBJECT IN FRONT OF
C               AT LEAST ONE OF THE SENSORS.  IF THIS THRESHOLD SIGNAL
C               IS NOT DETECTED BY EITHER SENSOR, THEN THE HAND
C               IS MOVED IN A SEARCH PATTERN UNTIL THE THRESHOLD
C               LEVEL IS DETECTED.
C
C
        SUBROUTINE DETECT
        IMPLICIT INTEGER(B-Z)
        COMMON/ARMBUF/INBUF(64),OUTBUF(64)
        COMMON/OUT/JPOS(8)
        COMMON/CORT/AC(6),AXEP,AYEP,AZEP
        COMMON/OFFS/OFF(3)
        EQUIVALENCE(EABORT,INBUF(28))
        EQUIVALENCE(P1,INBUF(49)),(P2,INBUF(48))
        CALL ARMIN
C
C   PLACE THE PRESENT JOINT POSITION VALUES IN THE MATRIX AC(6) TO BE
C       USED BY THE COORDINATE TRANSFORMATION ('COOR') CALL.
C
        DO 51 JJ=1,6
51      AC(JJ)=JPOS(JJ)
C
C  SET THE COUNTER FOR MOVEMENT IN THE 'Y' DIRECTION EQUAL TO ZERO.
C
        AY=0.
C
C   ZERO THE FLAG ('MIN') USED TO INDICATE THAT THE HAND SHOULD BE
C       MOVING IN THE MINUS 'Y' DIRECTION.
C
60      MIN=0
C
```

```
C    TEST EACH PROXIMITY SENSOR FOR THE THRESHOLD VALUE AND RETURN
C        TO THE CALLING PROGRAM ('EMOVE') IF THIS VALUE IS DETECTED.
C
20       IF(P1.GT.2000)GOTO 45
         IF(P2.GT.2000)GOTO 45
C
C    IF MOVING IN THE '-Y' DIRECTION, THEN BRANCH TO THE SECTION OF
C        CODE TO DECREMENT THE DELTA 'Y' COUNTER ('AY').
C
         IF(MIN.EQ.1)GOTO 30
C
C    ONCE THE HAND HAS MOVED 15 CENTIMETERS IN THE PLUS DIRECTION,
C        THEN BRANCH TO THE CODE TO MOVE THE HAND IN THE MINUS
C        DIRECTION.
C
         IF(AY.GT.15.)GOTO 31
C
C    THIS SECTION OF CODE ASSIGNS THE DELTA MOTION ('AYSIGN') AS
C        A POSITIVE .2 CENTIMETER.
C
25       MIN=0
         AYSIGN=.2
         GOTO 50
C
C    IF THE HAND HAS MOVED 15 CENTIMETERS IN THE MINUS DIRECTION FROM
C        THE INITIAL LOCATION, THEN BRANCH TO THAT CODE TO REVERSE
C        THE DIRECTION OF MOTION TO THE '+Y' DIRECTION.
C
30       IF(AY.LT.-15.)GOTO 25
C
C    THIS SECTION OF CODE ASSIGNS THE DELTA MOTION ('AYSIGN') AS
C        A NEGATIVE .2 CENTIMETER.  THE FLAG ('MIN') INDICATING
C        MOVEMENT IN THE NEGATIVE 'Y' DIRECTION IS SET EQUAL TO ONE.
C
31       MIN=1
         AYSIGN=-.2
C

C    HERE, THE DELTA MOTION IN THE 'Y' DIRECTION ('AYSIGN') IS ADDED
C        TO THE TOTAL DELTA 'Y' ('AY') FROM THE ORIGINAL STARTING POINT
C        TO DETERMINE WHAT THE NEW DELTA IS.
C
50       AY=AY+AYSIGN
C
C    THIS DELTA 'Y' OFFSET FROM THE STARTING POINT IS TRANSFORMED
C        INTO THE PROPER JOINT POSITION VALUES BY A CALL TO
C        'COOR'.  THESE JOINT POSITION VALUES ARE THEN
C        SENT TO THE SERVOS TO BE EXECUTED BY A CALL TO 'SERVO'.  IN
C        THIS WAY THE HAND MOVES IN THE 'Y' DIRECTION ALONG A PATH
C        THAT EXTENDS 15 CENTIMETERS ON EITHER SIDE OF THE STARTING
C        POINT MOVING IN INCREMENTS OF .2 CENTIMETERS AND CHECKING FOR
C        THE THRESHOLD LEVEL FROM THE SENSORS.
C
         CALL COOR(0.,AY,0.,0,0)
         CALL SERVO(0)
         IF(EABORT.GT.-2000)GOTO 20
```

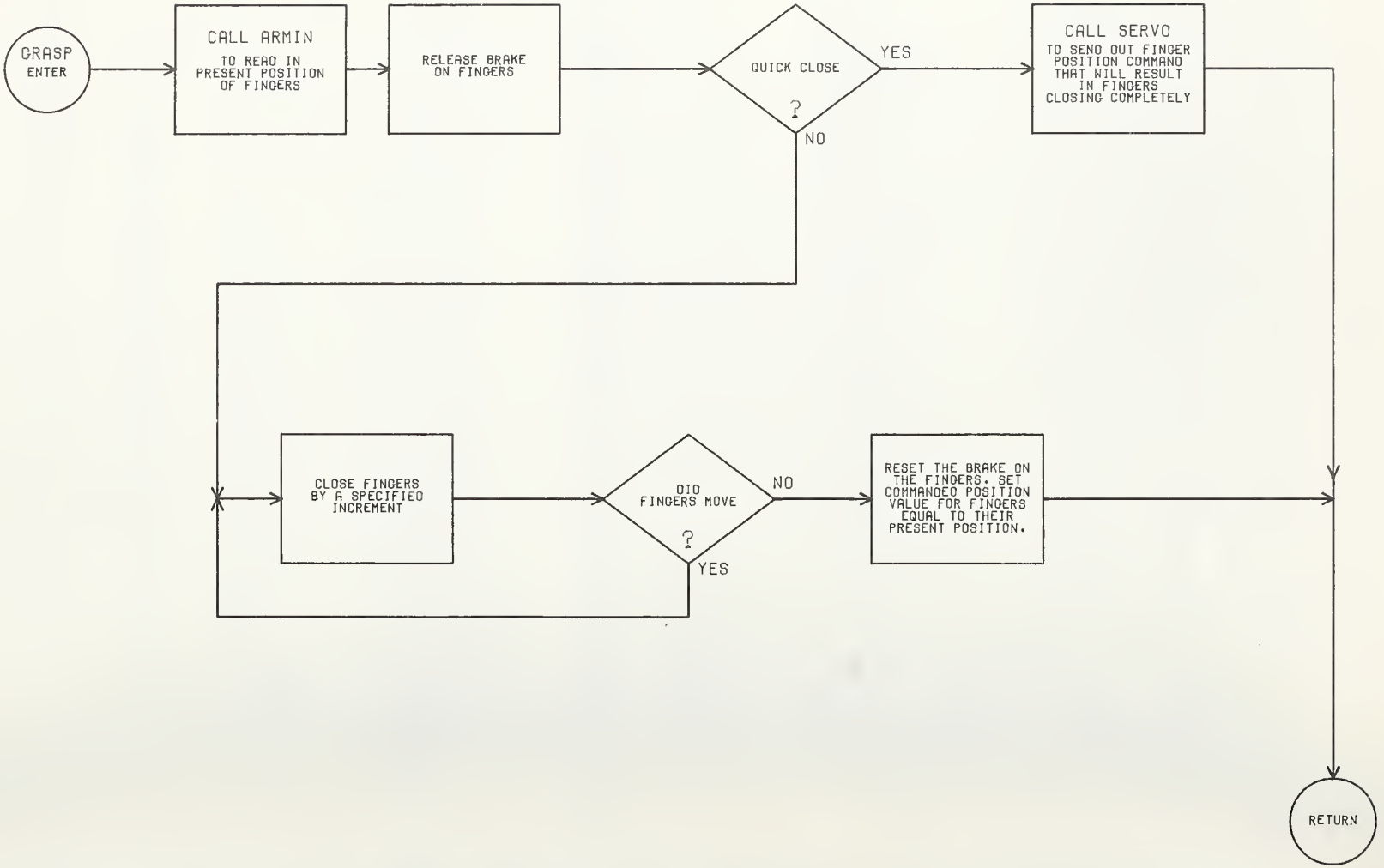
```
45      OFF(2)=OFF(2)+100*AY
40      RETURN
        END
```


BAL
ENTER
USE PRESENT JOINT POSITION VALUES AS REFERENCE LOCATION FOR COORDINATE TRANSFORMATION ROUTINE
CALCULATE DIFFERENCE BETWEEN THE TWO SENSOR SIONAL LEVELS
DIFFERENCE A NEGATIVE VALUE ?
YES
NO
SET INCREMENTAL "Y" VALUE EQUAL TO -0.05 CENTIMETERS SET MINUS FLAG
SET INCREMENTAL "Y" VALUE EQUAL TO +0.05 CENTIMETERS SET PLUS FLAG
IS PLUS FLAG SET ?
YES
NO
IS SENSOR BELOW THRESHOLD ?
YES
NO
EDGE OF OBJECT DETECTED. MOVE ADDITIONAL "-Y" DISTANCE SO FINGER WILL CLEAR THIS EDGE
IS MINUS FLAG SET ?
YES
NO
IS SENSOR BELOW THRESHOLD ?
YES
NO
EDGE OF OBJECT DETECTED. MOVE ADDITIONAL "+Y" DISTANCE SO FINGER WILL CLEAR THIS EDGE
ADD THE INCREMENTAL "Y" VALUE TO THE TOTAL "DELTA Y" OFFSET FROM INITIAL POSITION
CALL COOR &SERVO
TO TRANSFORM THIS "DELTA Y" OFFSET INTO THE CORRECT JOINT POSITION COMMANDS TO THE SERVOS
CALL ARMIN
TO READ IN THE UPDATED SENSOR VALUES
RETURN

```
C --- SUBROUTINE : BAL
C
C --- ARGUMENTS :
C
C --- CALLED BY : EMOVE
C
C --- CALLS SUBROUTINES : COOR  ARMIN  SERVO
C
C --- INPUT DATA : JPOS(7)      = THE PRESENT JOINT POSITION VALUES
C                  P1           = THE PRESENT VALUE OF THE SIGNAL
C                                 FROM THE PROXIMITY SENSOR ON
C                                 ONE OF THE FINGERS [FROM 'ARMIN'
C                                 ON INPUT CHANNEL NUMBER 49].
C                  P2           = THE PRESENT VALUE OF THE SIGNAL
C                                 FROM THE PROXIMITY SENSOR ON
C                                 THE OTHER FINGER [FROM 'ARMIN'
C                                 ON INPUT CHANNEL NUMBER 48].
C
C --- OUTPUT DATA : JPOS(7)         = THE COMMANDED JOINT POSITION VALUES
C                                    THAT WILL CAUSE THE ARM TO MOVE
C                                    ALONG A SEARCH PATH IN THE 'Y'
C                                    DIRECTION [FROM 'COOR'].
C
C --- FUNCTION: ASSUMES THAT AT LEAST ONE OF THE PROXIMITY SENSORS
C               IS ABOVE THE THRESHOLD VALUE, IE. AN OBJECT HAS BEEN
C               DETECTED).  THIS ROUTINE THEN INCREMENTS THE HAND'S
C               MOTION IN THE DIRECTION SO AS TO CAUSE THE TWO PROXIMITY
C               SIGNALS TO BE EQUAL.  IF BOTH SENSORS DO NOT REACH
C               THE THRESHOLD VALUE, THIS ROUTINE CAUSES THE
C               HAND TO MOVE UNTIL THE OBJECT DETECTED IS
C               WITHIN THE GRASP OF THE HAND (IE. IT ALIGNS THE INSIDE
C               EDGE OF ONE FINGER WITH THE OUTSIDE EDGE OF THE DETECTED
C               OBJECT).
C
        SUBROUTINE BAL
        IMPLICIT INTEGER(B-Z)
        COMMON/ARMBUF/INBUF(64),OUTBUF(64)
        COMMON/OUT/JPOS(8)
        COMMON/CORT/AC(6),AXEP,AYEP,AZEP
        COMMON/OFFS/OFF(3)
        EQUIVALENCE(EABORT,INBUF(28))
        EQUIVALENCE(P1,INBUF(49)),(P2,INBUF(48))
        CALL ARMIN
C
C   PLACE THE PRESENT JOINT POSITION VALUES IN THE MATRIX AC(6) TO BE
C       USED BY THE COORDINATE TRANSFORMATION ('COOR') CALL.
C
        DO 51 JJ=1,6
51      AC(JJ)=JPOS(JJ)
C
C  SET THE COUNTER FOR MOVEMENT IN THE 'Y' DIRECTION EQUAL TO ZERO.
C
        AY=0.
C
C   THE FLAGS ('M' AND 'P') TO INDICATE IF THE HAND IS BEING MOVED
C       IN A MINUS OR PLUS 'Y' DIRECTION DURING THE BALANCE FUNCTION,
```

```
C       ARE SET EQUAL TO ZERO.
C
        M=0
        P=0
C
C   THE DIFFERENCE BETWEEN THE VALUES OF THE TWO PROXIMITY SENSORS IS
C       CALCULATED AND STORED IN 'DS'.
C
30      DS=P1-P2
C
C   THE ALGEBRAIC SIGN OF 'DS' IS AN INDICATION OF WHICH OF THE TWO
C       SIGNALS IS THE LARGER AND IS USED TO DETERMINE WHICH
C       CODE SHOULD NOW BE EXECUTED.
C
        IF(DS.GT.0)GOTO 20
C
C   IF 'P2' IS THE LARGER SIGNAL, THEN THE HAND SHOULD MOVE IN THE
C       '-Y' DIRECTION, THEREFORE, THE DELTA 'Y' MOTION VALUE ('AYSIGN')
C       IS ASSIGNED THE VALUE OF -.05 CENTIMETERS.  THE MINUS 'Y'
C       DIRECTION FLAG ('M') IS SET EQUAL TO ONE.
C
        AYSIGN=-.05
        M=1
C
C   THE PLUS 'Y' DIRECTION FLAG ('P') IS TESTED.  IF IT IS EQUAL TO
C       ONE, THIS MEANS THAT THE HAND WAS MOVING IN THE PLUS 'Y'
C       DIRECTION AND NOW THE SENSORS INDICATE IT SHOULD MOVE IN THE
C       MINUS 'Y' DIRECTION.  FOR THIS SITUATION TO OCCUR, THE
C       SENSORS MUST HAVE PAST THROUGH A BALANCE POINT (IE. 'P1' WAS
C       THE LARGER, NOW 'P2' IS THE LARGER.  BETWEEN THESE TWO
C       CONDITIONS, 'P1' MUST HAVE EQUALED 'P2'), THEREFORE, RETURN TO
C       THE CALLING PROGRAM ('EMOVE') BECAUSE THE BALANCE POINT HAS BEEN
C       OBTAINED.
C
        IF(P.EQ.1)GOTO 45
C
C   IF THE CONDITION OCCURS THAT 'P2' STARTS DECREASING
C       IN VALUE BEFORE THE 'BALANCE' CONDITION IS REACHED, THEN
C       BRANCH TO THE SECTION OF CODE THE 'EDGE' DETECTION.  SINCE 'P2'
C       WAS ABOVE THE THRESHOLD LEVEL AND IS NOW DECREASING, THE SENSOR
C       MUST BE PASSING ACROSS THE EDGE OF THE UNDERLYING OBJECT.
C
        IF(P2.LT.1500)GOTO 75
        GOTO 50
C
C   IF 'P1' IS THE LARGER SIGNAL, THEN MOVE IN THE PLUS 'Y' DIRECTION
C       BY ASSIGNING A VALUE OF +.05 CENTIMETERS TO 'AYSIGN'.
C
20      AYSIGN=.05
C
C   THE SAME TYPE OF TESTS MADE FOR THE MINUS 'Y' MOTION ABOVE
C       ARE ALSO MADE HERE FOR THE PLUS 'Y' MOTION.
C
        P=1
        IF(M.EQ.1)GOTO 45
        IF(P1.LT.1500)GOTO 80
```

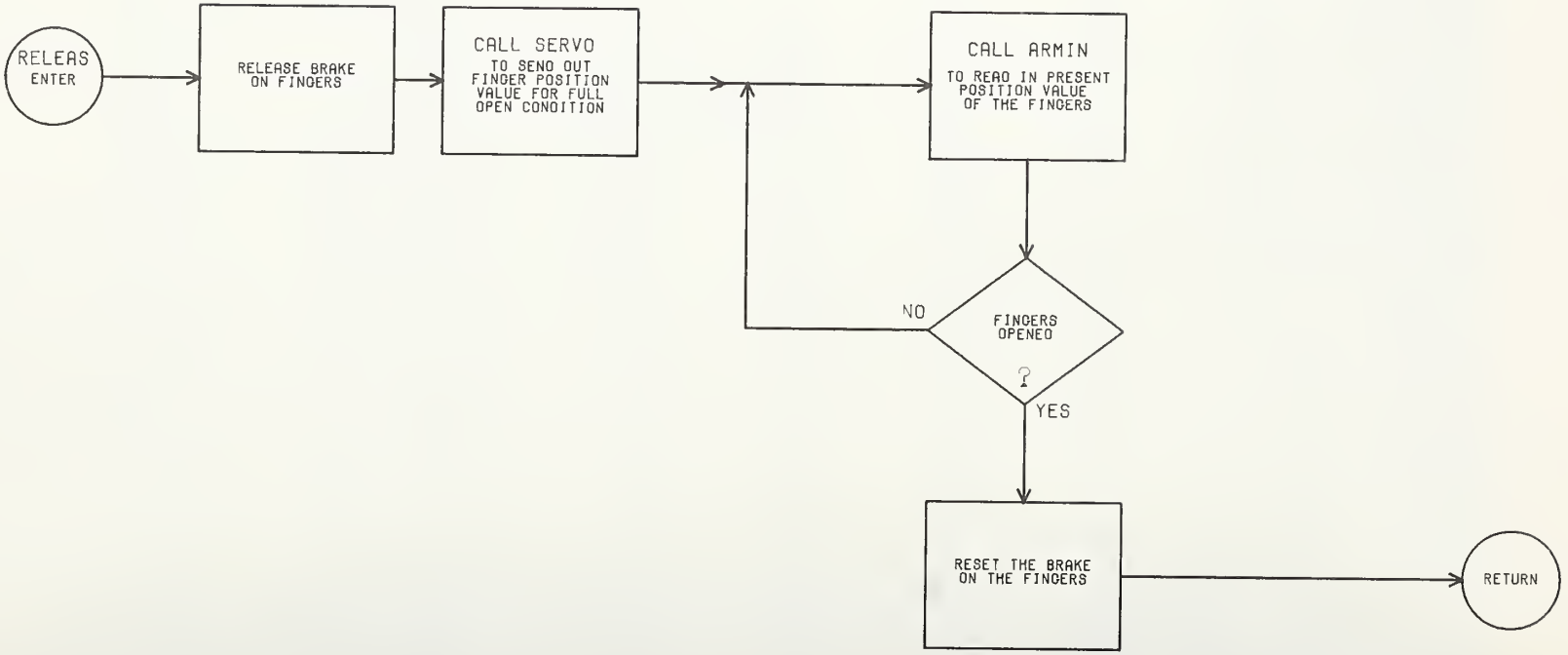
```
C
C    THE ACTUAL COMMANDED MOTION OF THE ARM IS CALCULATED BY THE
C        FOLLOWING CODE.  FIRST 'AY', THE TOTAL DELTA 'Y' MOTION FROM
C        THE STARTING POINT IS CALCULATED USING THE DELTA 'Y' VALUE
C        ASSIGNED ABOVE.  THEN A CALL TO 'COOR', THE COORDINATE
C        TRANSFORMATION ROUTINE, CALCULATES THE CORRECT JOINT POSITION
C        VALUES FOR THE LOCATION 'AY' DISTANCE AWAY FROM THE STARTING
C        POINT.  THEN A CALL TO 'SERVO' SENDS THESE VALUES OUT TO THE SERVO
C        SYSTEM OF THE ARM.  'ARMIN' IS CALLED TO BRING IN THE CURRENT
C        VALUES OF THE PROXIMITY SENSORS FOR ANOTHER TEST, STARTING
C        BACK AT STATEMENT 30.
C
50       AY=AY+AYSIGN
         CALL COOR(0.,AY,0.,0,0)
         CALL SERVO(0)
         IF(EABORT.LT.-2000)GOTO 40
         GOTO 30
C
C    IF AN EDGE IS DETECTED BY 'P2', THEN THE FOLLOWING CODE ADVANCES
C        THE HAND AN ADDITIONAL DISTANCE IN THE '-Y' DIRECTION (WHICH
C        DISTANCE IS EQUAL TO THE THICKNESS OF THE FINGER THAT THE
C        SENSOR 'P2' IS ATTACHED TO ) TO PLACE THE INSIDE SURFACE OF
C        THE FINGER FLUSH WITH THE UNDERLYING DETECTED EDGE.
C
75       DO 77 GT=1,48
         AY=AY-.05
         CALL COOR(0.,AY,0.,0,0)
77       CALL SERVO(0)
         GOTO 45
C
C    IF 'P1' DETECTS AN EDGE, THIS CODE MOVES THE HAND IN THE '+Y'
C        DIRECTION BY AN AMOUNT EQUAL TO THE THICKNESS OF THE FINGER AS
C        ABOVE.
C
80       DO 81 GGT=1,40
         AY=AY+.05
         CALL COOR(0.,AY,0.,0,0)
81       CALL SERVO(0)
45       OFF(2)=OFF(2)+100*AY
40       RETURN
         END
```


GRASP
ENTER
CALL ARMIN
TO READ IN PRESENT POSITION OF FINGERS
RELEASE BRAKE ON FINGERS
QUICK CLOSE
?
YES
NO
CALL SERVO
TO SEND OUT FINGER POSITION COMMAND THAT WILL RESULT IN FINGERS CLOSING COMPLETELY
CLOSE FINGERS BY A SPECIFIED INCREMENT
DID FINGERS MOVE
?
NO
YES
RESET THE BRAKE ON THE FINGERS. SET COMMANDED POSITION VALUE FOR FINGERS EQUAL TO THEIR PRESENT POSITION.
RETURN

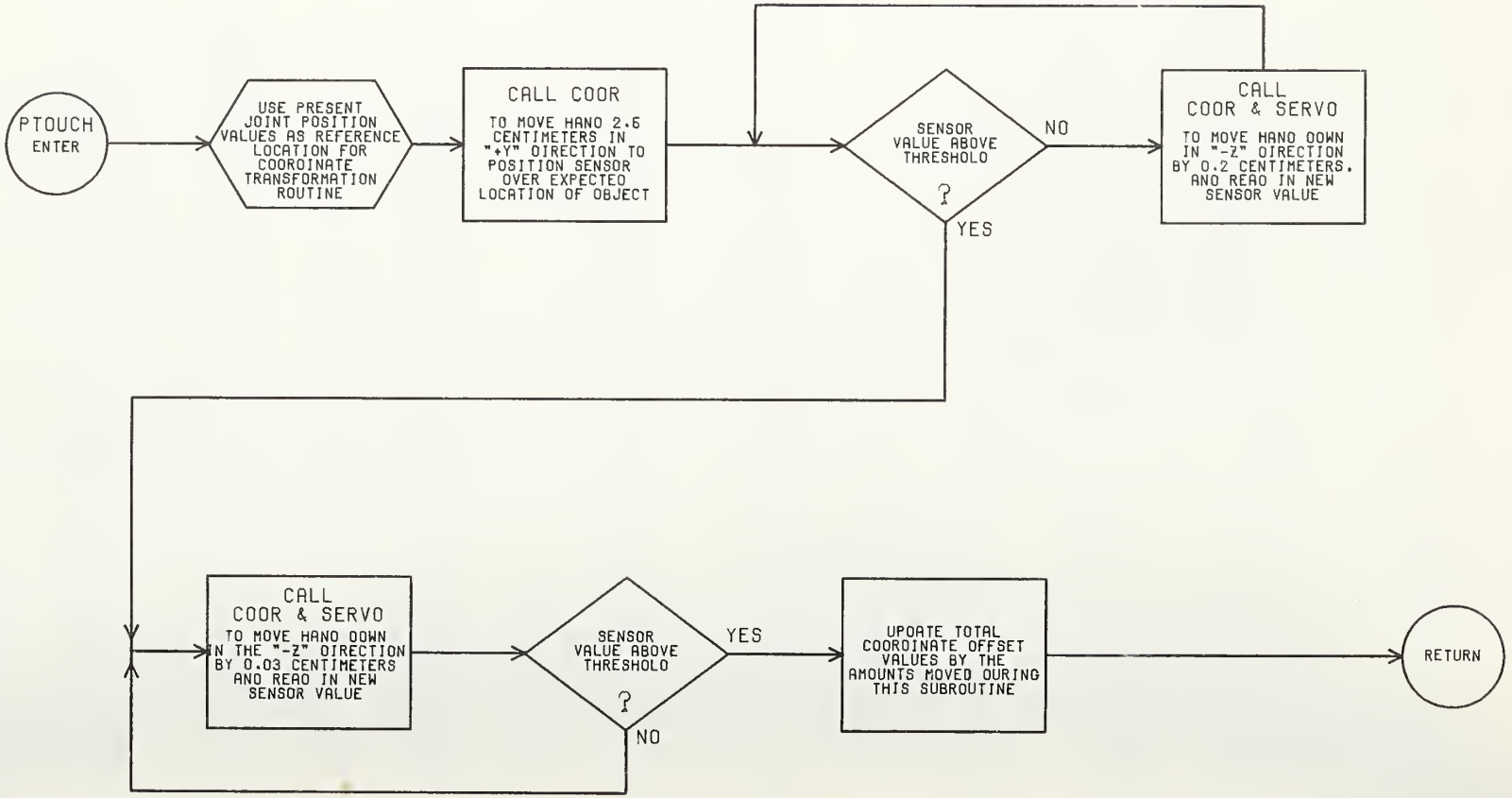
```
C --- SUBROUTINE : GRASP
C
C --- ARGUMENTS : D        IF D=0  THIS IS THE QUICK CLOSE COMMAND.
C                                  THE POSITION VALUE OF THE FINGERS IN
C                                  THE CLOSED POSITION IS SENT TO THE
C                                  SERVOS.  CONTROL IS IMMEDIATELY
C                                  RETURNED TO THE CALLING PROGRAM
C                                  WITHOUT WAITING FOR THE FINGERS TO
C                                  COMPLETE THE CLOSING OPERATION.
C                           IF D=1  HERE, THE FINGERS ARE CLOSED AN
C                                  INCREMENT AT A TIME UNTIL A DIFFERENCE
C                                  IS DETECTED BETWEEN THE COMMANDED
C                                  POSITION AND THE ACTUAL POSITION AS
C                                  READ IN FROM THE POSITION INDICATOR.
C
C --- CALLED BY : EMOVE
C
C --- CALLS SUBROUTINES : ARMIN  SERVO
C
C --- INPUT DATA : INBUF(19)   = THE INPUT CHANNEL THAT CONTAINS
C                                THE PRESENT VALUE OF THE POSITION
C                                INDICATOR FOR THE FINGERS.
C
C --- OUTPUT DATA : JPOS(7)    = THE COMMANDED OUTPUT POSITION VALUE
C                                FOR THE FINGERS.
C                  JPOS(8)    = THE DIGITAL (16 BITS) OUTPUT CHANNEL
C                                THAT CONTROLS THE SETTING
C                                OF THE BRAKES ON ALL OF THE JOINTS.
C
C --- FUNCTION: CAUSES THE FINGERS TO CLOSE UNTIL THEY EXERT A PREDEFINED
C              FORCE ON AN OBJECT,  THEN IT SETS THE BRAKE ON THE
C              FINGERS (D=0).  OR SENDS OUT THE POSITION COMMAND THAT
C              WILL RESULT IN THE FINGERS CLOSING ALL OF THE WAY (D=1).
C
      SUBROUTINE GRASP(D)
      IMPLICIT INTEGER(B-Z)
      COMMON/ARMBUF/INBUF(64),OUTBUF(64)
      COMMON/OUT/JPOS(8)
      EQUIVALENCE(EABORT,INBUF(28))
      COMMON/GRIP/HAND
C
C   READ IN THE PRESENT POSITION OF THE FINGERS AND SET THE OUTPUT
C       POSITION COMMAND FOR THE FINGERS (JPOS(7)) EQUAL TO THIS
C       VALUE.
C
      CALL ARMIN
      JPOS(7)=2*INBUF(19)+16383
C
C   SEND OUT THE PROPER CODE NUMBER TO SET UP THE BIT PATTERN
C       ON OUTPUT CHANNEL 8 (JPOS(8)) TO RELEASE THE BRAKE ON THE
C       FINGERS.
C
      JPOS(8)=32767
      CALL SERVO(0)
C
C   TEST 'D'.  IF D=1, THEN SEND THE POSITION VALUE OF 6600 TO THE
```

```
C       FINGER SERVO (THIS IS USED TO CLOSE THE FINGERS
C       SO THAT THE SENSORS CAN BE USED IN SOME OF THE SEARCH ROUTINES.
C       THE OUTPUTTING OF THIS VALUE TO THE SERVOS CAUSES THE FINGERS TO
C       CLOSE COMPLETELY, HOWEVER, THIS ROUTINE
C       DOES NOT WAIT FOR THIS ACTION TO BE FINISHED BEFORE RETURNING TO
C       THE CALLING PROGRAM.  AS A RESULT, THE FINGERS WILL STILL BE IN
C       MOTION WHILE THE OTHER SUBROUTINES OF THE CONTROL SYSTEM ARE
C       IN EXECUTION.).  IF D=0 THEN BRANCH TO STATEMENT 20.
C
        IF(D.NE.1)GOTO 20
        JPOS(7)=6600
        CALL SERVO(0)
        GOTO 40
C
C   THE COUNTER 'KB' WHICH IS USED IN THE DETECTION OF THE GRIPPING FORCE
C       OF THE FINGERS IS SET EQUAL TO ZERO.
C
20      KB=0
C
C   THE JOINT POSITION COMMAND FOR THE FINGERS (JPOS(7)) IS DECREMENTED
C       WHICH WILL CAUSE AN INCREMENTAL CLOSING WHEN SENT TO THE SERVO.
C
        JPOS(7)=JPOS(7)-100
100     CALL SERVO(0)
C
C   THE INPUT CHANNELS ARE SAMPLED TO DETERMINE IF AN ABORT HAS BEEN
C       CALLED, AND TO DETERMINE THE PRESENT VALUE OF THE POSITION
C       OF THE FINGERS ('FINGER').
C
        IF(EABORT.LT.-2000)GOTO 40
        FINGER=2*INBUF(19)+16383
C
C   THE DIFFERENCE BETWEEN THE PRESENT POSITION ('FINGER') OF THE
C       FINGERS AND THE COMMANDED (JPOS(7)) POSITION OF THE FINGERS
C       IS CALCULATED AND STORED IN TST.
C
        TST=FINGER-JPOS(7)
C
C   AS LONG AS THIS DIFFERENCE ('TST') IS LESS THAN 800, THEN
C       WE CONTINUE TO DECREMENT THE COMMANDED POSITION VALUE, THUS
C       CONTINUING TO CLOSE THE FINGERS.  IF, HOWEVER, THE DIFFERENCE
C       BECOMES GREATER THAN 800, WE THEN USE 'KB' AS A COUNTER AND
C       SAMPLE THE PRESENT POSITION FOR .2 SEC (KB=50) TO MAKE CERTAIN
C       THAT THE FINGERS ARE NO LONGER CLOSING.  THIS INDICATES THAT
C       THE FINGERS HAVE GRASPED AN OBJECT WITH A CERTAIN FORCE.
C
        IF(TST.LT.800)GOTO 20
        KB=KB+1
        IF(KB.LT.50)GOTO 100
C
C   ONCE AN OBJECT IS GRASPED, THE BRAKE ON THE FINGERS IS SET BY
C       SETTING THE OUTPUT CHANNEL 8 (JPOS(8)) EQUAL TO 32703.
C
310     JPOS(8)=32703
        CALL SERVO(0)
C
```

```
C    THE PRESENT POSITION VALUE OF THE FINGERS IS READ IN AND THE
C        COMMANDED POSITION VALUE (JPOS(7)) OF THE FINGERS SET EQUAL
C        TO IT.  THIS ZEROS THE SERVO ERROR, THEREFORE, THE
C        ACTUATOR IS NO LONGER DRIVING THE FINGERS TO CLOSE, SINCE THE
C        BRAKE WILL NOW HOLD THE FINGERS CLOSED ON THE OBJECT.
C
         CALL ARMIN
         JPOS(7)=2*INBUF(19)+16383
C
C    THE ACTUAL POSITION VALUE OF THE FINGERS IS STORED IN THE COMMON
C        VARIABLE 'HAND' SO THAT IT CAN BE ACCESSED BY OTHER SUBROUTINES
C        IF NECESSARY.
C
         HAND=JPOS(7)
         CALL SERVO(0)
40       RETURN
         END
```

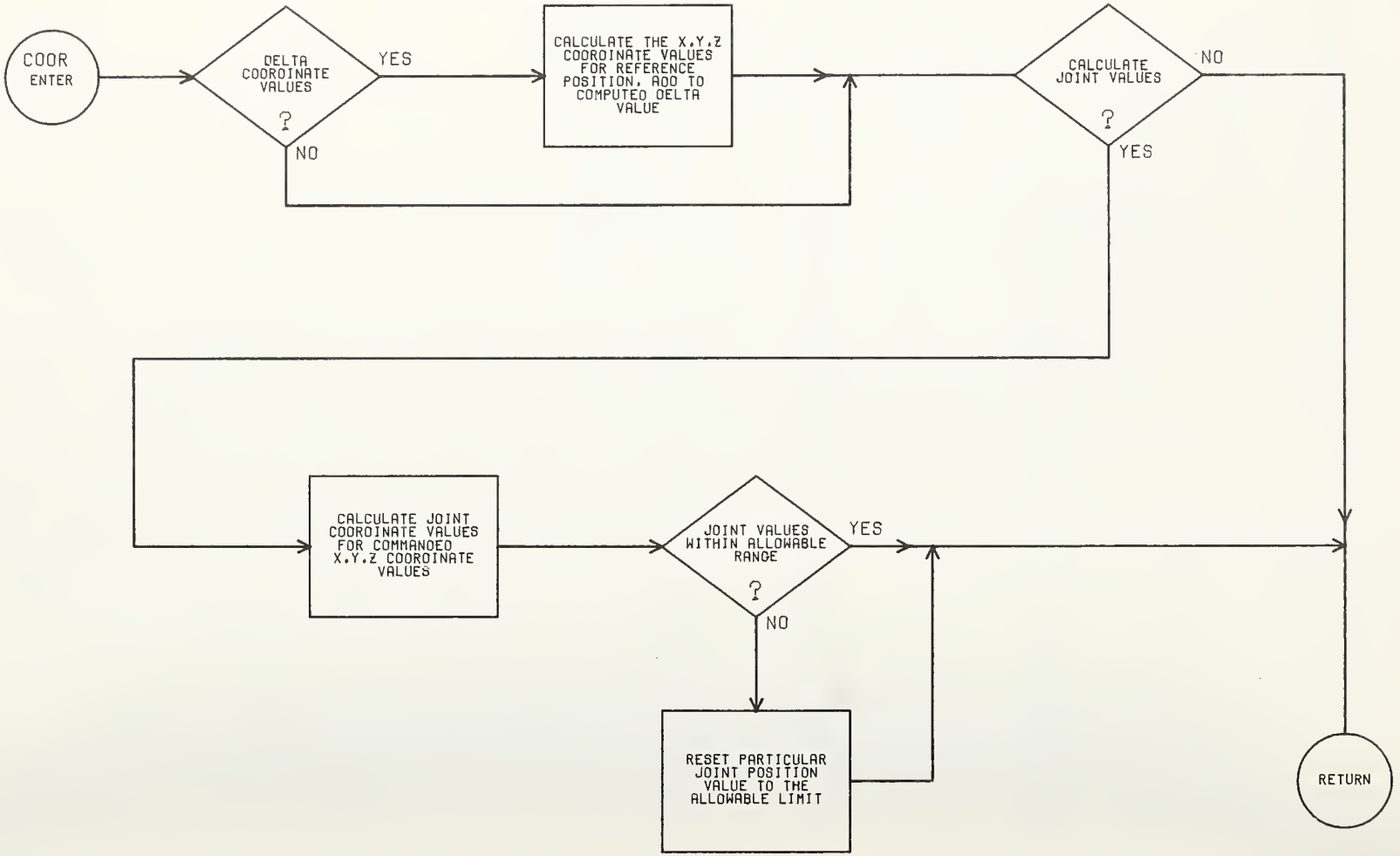

RELEAS
ENTER
RELEASE BRAKE
ON FINGERS
CALL SERVO
TO SENO OUT
FINGER POSITION
VALUE FOR FULL
OPEN CONDITION
CALL ARMIN
TO READ IN PRESENT
POSITION VALUE
OF THE FINGERS
NO
FINGERS
OPENED
?
YES
RESET THE BRAKE
ON THE FINGERS
RETURN

```
C --- SUBROUTINE : RELEAS
C
C --- ARGUMENTS :
C
C --- CALLED BY : EMOVE
C
C --- CALLS SUBROUTINES : ARMIN  SERVO
C
C --- INPUT DATA : INBUF(19)   = THE INPUT CHANNEL THAT CONTAINS
C                                THE PRESENT VALUE OF THE POSITION
C                                INDICATOR FOR THE FINGERS.
C
C --- OUTPUT DATA : JPOS(7)    = THE COMMANDED OUTPUT POSITION VALUE
C                                FOR THE FINGERS.
C                   JPOS(8)    = THE DIGITAL (16 BITS) OUTPUT CHANNEL
C                                THAT CONTROLS THE SETTING
C                                OF THE BRAKES ON ALL OF THE JOINTS.
C
C --- FUNCTION: CAUSES THE FINGERS OF THE HAND TO FULLY OPEN.
C
      SUBROUTINE RELEAS
      IMPLICIT INTEGER(B-Z)
      COMMON/ARMBUF/INBUF(64),OUTBUF(64)
      COMMON/OUT/JPOS(8)
      EQUIVALENCE(EABORT,INBUF(28))
C
C   SEND OUT THE PROPER CODE NUMBER TO SET UP THE BIT PATTERN
C       ON OUTPUT CHANNEL 8 (JPOS(8)) TO RELEASE THE BRAKE ON THE
C       FINGERS.
C
      JPOS(8)=32767
C
C   SEND THE POSITION VALUE OF 31000 TO THE SERVO.  WHEN THE FINGERS
C       ARE SERVOED TO THIS POSITION THEY WILL BE FULLY OPENED.
C
      JPOS(7)=31000
      CALL SERVO(0)
C
C
C   THE INPUT CHANNELS ARE SAMPLED TO DETERMINE IF AN ABORT HAS BEEN
C       CALLED, AND TO DETERMINE THE PRESENT VALUE OF THE POSITION
C       OF THE FINGERS ('FINGER').
C
100   CALL ARMIN
      IF(EABORT.LT.-2000)GOTO 40
      FINGER=2*INBUF(19)+16383
C
C   IF THE FINGERS HAVE NOT YET REACHED THE FULLY OPENED POSITION
C       THEN CONTINUING SAMPLING THEIR POSITION (GOTO STATEMENT 100).
C
      IF(FINGER.LT.30400)GOTO 100
C
C   ONCE THE HAND IS FULLY OPENED, SET THE BRAKE (JPOS(8)), AND
C       SET THE COMMANDED POSITION VALUE OF THE FINGERS EQUAL TO
C       THEIR PRESENT POSITION.
C
      JPOS(8)=32703
      JPOS(7)=FINGER
      CALL SERVO(0)
40    RETURN
      END
```


PTOUCH
ENTER
USE PRESENT JOINT POSITION VALUES AS REFERENCE LOCATION FOR COORDINATE TRANSFORMATION ROUTINE
CALL COOR
TO MOVE HAND 2.6 CENTIMETERS IN "+Y" DIRECTION TO POSITION SENSOR OVER EXPECTED LOCATION OF OBJECT
SENSOR VALUE ABOVE THRESHOLD ?
NO
YES
CALL COOR & SERVO
TO MOVE HAND DOWN IN "-Z" DIRECTION BY 0.2 CENTIMETERS. AND READ IN NEW SENSOR VALUE
CALL COOR & SERVO
TO MOVE HAND DOWN IN THE "-Z" DIRECTION BY 0.03 CENTIMETERS AND READ IN NEW SENSOR VALUE
SENSOR VALUE ABOVE THRESHOLD ?
YES
NO
UPDATE TOTAL COORDINATE OFFSET VALUES BY THE AMOUNTS MOVED DURING THIS SUBROUTINE
RETURN

```
C --- SUBROUTINE : PTOUCH
C
C --- ARGUMENTS :
C
C --- CALLED BY : EMOVE
C
C --- CALLS SUBROUTINES : ARMIN  SERVO  COOR
C
C --- INPUT DATA : INBUF(48)     = 'P2' THE VALUE OF ONE OF THE PROXIMITY
C                                  SENSORS.
C                   JPOS(1-7)    = THE JOINT POSITION VALUES AT THE
C                                  BEGINNING OF THE ROUTINE.
C                   JPOS(1-7)    = THE JOINT POSITION COMMANDS NECESSARY
C                                  TO MOVE THE ARM TO THE POSITION
C                                  DELTA 'Z' FROM THE STARTING POINT [FROM
C                                  'COOR'].
C
C --- OUTPUT DATA : ADY          = THE PRESENT DELTA 'Y' OFFSET VALUE FROM
C                                  THE INITIAL POSITION OF THE ARM.
C                    ADZ          = THE PRESENT DELTA 'Z' OFFSET VALUE FROM
C                                  THE INITIAL POSITION OF THE ARM.
C                    JPOS(1-7)    = THE JOINT POSITION COMMANDS TO BE
C                                  SENT TO THE SERVOS.
C                     OFF(1-3)    = THE TOTAL OFFSET IN THE X,THE Y,
C                                  AND THE Z DIRECTIONS FROM THE ORIGINAL
C                                  STARTING LOCATION.
C
C --- FUNCTION: CAUSES THE HAND TO DESCEND IN A STRAIGHT LINE IN THE 'Z'
C               DIRECTION WHILE TESTING THE VALUE OF THE SENSOR
C               FOR A THRESHOLD LEVEL EVERY .2 CENTIMETER.  WHEN THE
C               THRESHOLD LEVEL IS DETECTED, THE HAND IS
C               STOPPED AT THAT LOCATION AND CONTROL IS RETURNED TO THE
C               CALLING PROGRAM.
C
      SUBROUTINE PTOUCH
      IMPLICIT INTEGER(B-Z)
      COMMON/ARMBUF/INBUF(64),OUTBUF(64)
      COMMON/OFFS/OFF(3)
      COMMON/OUT/JPOS(8)
      COMMON/CORT/AC(6),AXEP,AYEP,AZEP
      EQUIVALENCE(EABORT,INBUF(28))
      EQUIVALENCE(P1,INBUF(49)),(P2,INBUF(48))
C
C   SET THE DELTA 'Y' (ADY) AND THE DELTA 'Z' VALUE EQUAL TO ZERO.
C
      ADY=0.
      ADZ=0.
C
C   PLACE THE PRESENT JOINT POSITION VALUES IN THE COMMON
C       VARIABLE 'AC(1-6)' FOR USE BY 'COOR'.
C
      DO 20 J=1,6
20    AC(J)=JPOS(J)
C
C   MOVE THE HAND 3.0 CENTIMETERS IN THE '+Y' DIRECTION TO
C       POSITION THE SENSOR ON THE OUTSIDE EDGE OF THE FINGER
```

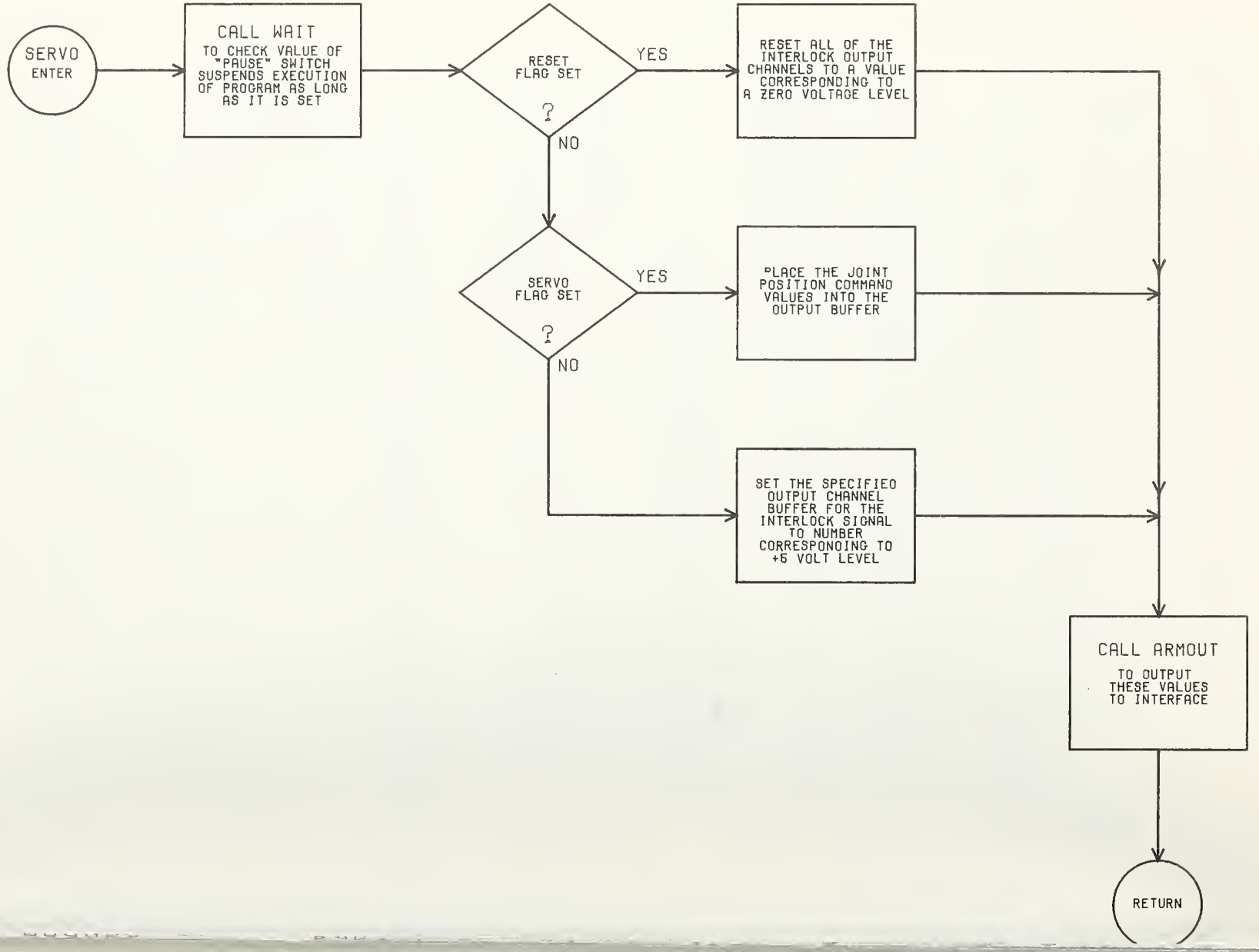
```
C       OVER THE EXPECTED CENTER OF THE STACK.
C
        DO 30 J=1,12
        ADY=ADY+.25
        CALL COOR(0.,ADY,0.,0,0)
30      CALL SERVO(0)
C
C   TEST THE VALUE OF THE PROXIMITY SENSOR TO DETERMINE IF THE TOP OF
C       THE STACK HAS BEEN DETECTED.
C
50      IF(P2.GT.1500)GOTO 35
C
C   MOVE THE HAND ANOTHER .2 CENTIMETERS IN THE '-Z' DIRECTION, THEN
C       TEST THE SENSOR VALUE AGAIN.
C
        ADZ=ADZ-.2
36      CALL COOR(0.,ADY,ADZ,0,0)
        CALL SERVO(0)
        IF(EABORT.LT.-2000)GOTO 40
        GOTO 50
C
C   ONCE THE TOP OF THE STACK HAS BEEN DETECTED, MOVE THE HAND DOWN
C       SLOWLY UNTIL THE THRESHOLD LEVEL IS DETECTED.
C
35      ADZ=ADZ-.03
        IF(P2.GT.4000)GOTO 45
        GOTO 36
C
C   UPDATE THE TOTAL OFFSET VALUES BY THE AMOUNTS MOVED DURING THIS
C       SUBROUTINE CALL.
C
45      OFF(2)=100*ADY+OFF(2)
        OFF(3)=100*ADZ+OFF(3)
40      RETURN
        END
```


COOR
ENTER
DELTA
COORDINATE
VALUES
?
YES
NO
CALCULATE THE X,Y,Z
COORDINATE VALUES
FOR REFERENCE
POSITION, ADD TO
COMPUTED DELTA
VALUE
CALCULATE
JOINT VALUES
?
NO
YES
CALCULATE JOINT
COORDINATE VALUES
FOR COMMANDED
X,Y,Z COORDINATE
VALUES
JOINT VALUES
WITHIN ALLOWABLE
RANGE
?
YES
NO
RESET PARTICULAR
JOINT POSITION
VALUE TO THE
ALLOWABLE LIMIT
RETURN

```
C --- SUBROUTINE : COOR
C
C --- ARGUMENTS : AX        = THE 'X' COORDINATE VALUE
C                 AY        = THE 'Y' COORDINATE VALUE.
C                 AZ        = THE 'Z' COORDINATE VALUE.
C                 DA        = THE FLAG TO INDICATE WHETHER THE
C                             'AX,AY,AZ' VALUES ARE ABSOLUTE COORDINATE
C                             POSITIONS (DA=1), OR DELTA COORDINATE OFFSETS
C                             FROM THE PRESENT POSITION OF THE ARM (DA=0).
C                 NA        = THE FLAG TO INDICATE WHETHER THE JOINT
C                             COORDINATE VALUES ARE TO BE CALCULATED (NA=0),
C                             OR NOT (NA=1).
C
C --- CALLED BY : JOY  LOCTAB  ARRLOC  EMOVE  STLINE  CALD  DETECT
C                 BAL  PTOUCH
C
C --- CALLS SUBROUTINES :
C
C --- INPUT DATA : AX,AY,AZ,DA,NA       AS EXPLAINED ABOVE
C                  AC(1-6)      = THE VALUES OF THE SIX JOINT COORDINATES
C                                 USED AS A REFERENCE POINT FOR THIS
C                                 TRANSFORMATION.
C                  A04,A05,A06 = THE VALUES OF THE OFFSETS FOR THE FOURTH,
C                                 FIFTH, AND SIXTH JOINTS (FROM 'JOY').
C
C --- OUTPUT DATA : AXEP,AYEP,AZEP = THE NEW 'X,Y,Z' COORDINATE POSITION
C                                  CALCULATED FROM THE REFERENCE
C                                  POINT (AC(1-6)) AND THE INPUT 'X,Y,Z'
C                                  VALUES (AX,AY,AZ).
C                   JPOS(1-6)      = THE JOINT POSITION VALUES THAT
C                                  ARE CALCULATED FROM THE REFERENCE
C                                  POINT (AC(1-6)) AND THE INPUT 'X,Y,Z'
C                                  VALUES (AX,AY,AZ).
C
C --- FUNCTION: TRANSFORMS THE 'X,Y,Z' COORDINATE VALUES FOR A POSITION
C               IN SPACE INTO THE CORRESPONDING JOINT COORDINATE VALUES,
C               AND VISE VERSA.(THIS IS ONLY A THREE AXIS TRANSFORMATION,
C               NOT THE ENTIRE SIX AXIS TRANSFORMATION.  THE FOURTH,
C               FIFTH AND SIXTH AXIS PLANOGRAPH THE MOTIONS
C               OF THE FIRST THREE.)
C
      SUBROUTINE COOR(AX,AY,AZ,DA,NA)
      IMPLICIT INTEGER(D-R)
      COMMON/XYZO/A04,A05,A06
      COMMON/CORT/AC(6),AXEP,AYEP,AZEP
      COMMON/OUT/JPOS(8)
      COMMON/ARMBUF/INBUF(64),OUTBUF(64)
      DIMENSION AOM(6)
C
C   TEST IF AX,AY,AZ ARE TO BE DELTA OR ABSOLUTE COORDINATES.
C
      IF(DA)5,5,6
C
C   IF ABSOLUTE,SET 'AXEP','AYEP','AZEP' EQUAL TO THEM AND BRANCH
C       TO STATEMENT 10 TO CALCULATE THE CORRESPONDING JOINT POSITION
C       VALUES.
```

```
C
6       AXEP=AX
        AYEP=AY
        AZEP=AZ
        GOTO 10
C
C   IF AX,AY,AZ ARE DELTA COORDINATE OFFSETS, CALCULATE COORDINATES
C       OF PRESENT POSITION (PRESENT POSITION JOINT INDICATOR
C       VALUES ARE IN 'AC(1-6)').
C
5       AR=(34552.-AC(3))/358.7
        AT1=(15018.-AC(6))/5182.
        AT2=(17821.-AC(2))/5190.
        ARR=AR*COS(AT2)
        AB=SQRT(275.56+ARR**2)
        AA=ATAN(ARR/16.6)
        AG=AT1+AA-1.5708
C**********
        AAZ=AR*SIN(AT2)
        AAX=AB*COS(AG)
        AAY=AB*SIN(AG)
C**********
C
C   ADD PRESENT POSITION X,Y,Z COORDINATE VALUES TO THE COMMANDED
C       DELTA COORDINATE OFFSET VALUES.
C
        AXEP=AAX+AX
        AYEP=AAY+AY
        AZEP=AAZ+AZ
C
C   IF JOINT POSITIONS NOT TO BE CALCULATED (NA=1), THEN RETURN TO THE
C       CALLING PROGRAM.
C
        IF(NA.EQ.1)GOTO 40
C**********
C
C   CALCULATE THE JOINT POSITION VALUES THAT CORRESPOND TO THE POSITION
C       IN SPACE DESIGNATED BY THE COORDINATES 'AXEP','AYEP','AZEP'.
C
10      ARRPS=AXEP**2+AYEP**2-275.56
        IF(ARRPS)1323,1323,1324
1323    WRITE(6,1325)ARRPS
1325    FORMAT(2X'ERROR**  ARRPS=',F10.3)
        READ(6,1326)KKR
1326    FORMAT(I3)
1324    ARRP=SQRT(ARRPS)
        IF(AXEP)202,201,200
202     IF(AYEP)203,204,204
203     AT1P=ATAN(AYEP/AXEP)-1.5708-ATAN(ARRP/16.6)
        GOTO 205
204     AT1P=ATAN(AYEP/AXEP)-1.5708-ATAN(ARRP/16.6)
        IF(AT1P.GT.-1.60)GOTO 205
        AT1P=ATAN(AYEP/AXEP)+4.7124-ATAN(ARRP/16.6)
        IF(AT1P.LT.2.54)GOTO 205
        WRITE(6,220)
220     FORMAT(2X'THIS IS FORBIDDEN!!')
```

```
        GOTO 40
201     AXEP=.0001
200     AT1P=ATAN(AYEP/AXEP)+1.5708-ATAN(ARRP/16.6)
205     ARP=SQRT(ARRPS+AZEP**2)
        AT2P=ATAN(AZEP/ARRP)
C**********
1000    AOM(6)=15018.-(AT1P*5182.)
        AOM(2)=17821.-(AT2P*5190.)
        AOM(3)=34552.-(ARP*358.7)
        AOM(4)=17437.+AO4
        AOM(5)=1.202*AOM(2)-16461.+AO5
        AOM(1)=-.8837*AOM(6)+35380.+AO6
C
C   TEST TO MAKE CERTAIN THAT ALL THE NEW JOINT VALUES
C       (JPOS(1-6)) ARE WITHIN THEIR ALLOWABLE LIMITS.
C
720     DO 265 JB=1,6
        IF(AOM(JB).GT.30000.)GOTO 266
        IF(AOM(JB).LT.2000.)GOTO 267
        JPOS(JB)=AOM(JB)
        GOTO 265
266     JPOS(JB)=30000
        GOTO 265
267     JPOS(JB)=2000
265     CONTINUE
40      RETURN
        END
```


SERVO
ENTER
CALL WAIT
TO CHECK VALUE OF "PAUSE" SWITCH SUSPENDS EXECUTION OF PROGRAM AS LONG AS IT IS SET
RESET FLAG SET ?
YES
NO
RESET ALL OF THE INTERLOCK OUTPUT CHANNELS TO A VALUE CORRESPONDING TO A ZERO VOLTAGE LEVEL
SERVO FLAG SET ?
YES
NO
PLACE THE JOINT POSITION COMMAND VALUES INTO THE OUTPUT BUFFER
SET THE SPECIFIED OUTPUT CHANNEL BUFFER FOR THE INTERLOCK SIGNAL TO NUMBER CORRESPONDING TO +5 VOLT LEVEL
CALL ARMOUT
TO OUTPUT THESE VALUES TO INTERFACE
RETURN

```
C --- SUBROUTINE : SERVO
C
C --- ARGUMENTS : SD      IF SD=-1 ZERO THE EIGHT OUTPUT CHANNELS
C                                    FOR THE 'SEND' FUNCTION COMMAND.
C                           IF SD=0  SEND OUT TO THE SERVOS, THE SEVEN
C                                    JOINT POSITION COMMANDS AND THE VALUE
C                                    IN JPOS(8) FOR CONTROLLING THE BRAKES.
C                              SD=   IF 'SD' IS A NON-ZERO VALUE, IT IS THE
C                                    OUTPUT CHANNEL NUMBER ON WHICH A VOLTAGE
C                                    LEVEL IS TO BE SENT AS REQUESTED BY THE
C                                    'SEND' FUNCTION.
C
C --- CALLED BY : EMOVE  RELEAS  GRASP  STLINE  BAL  DETECT
C                 PTOUCH  ACC  JOY
C
C --- CALLS SUBROUTINES :  ARMOUT  ARMIN
C
C --- INPUT DATA : SD          = THE ARGUMENT AS EXPLAINED ABOVE.
C                  PNT         = THE FLAG IN COMMON MEMORY THAT IF
C                                IT IS SET EQUAL TO ONE CAUSES THIS
C                                SUBROUTINE 'SERVO' TO MAINTAIN CONTROL
C                                UNTIL THE ACTUAL POSITION OF THE JOINTS
C                                IS WITHIN SOME MINIMUM DISTANCE OF THE
C                                COMMANDED JOINT POSITIONS.
C
C --- OUTPUT DATA : OUTBUF(1)-OUTBUF(8) = THE SEVEN COMMANDED JOINT
C                                         POSITION VALUES, AND THE
C                                         15-BIT DIGITAL CHANNEL FOR
C                                         CONTROLLING THE BRAKES.
C                   OUTBUF(9)-OUTBUF(16)= THE EIGHT OUTPUT CHANNELS THAT
C                                         CAN BE USED BY THE 'SEND'
C                                         FUNCTION COMMAND.  THIS IS USED
C                                         FOR THE INTERLOCKING OF THE ARM
C                                         EXTERNAL DEVICES.
C
C --- FUNCTION: USES THE ARGUMENT TO DETERMINE WHICH OUTPUTS SHOULD BE
C               SENT TO THE ROBOT WORK STATION.  ALL OUTPUTS FROM
C               THE CONTROL SYSTEM ARE MADE THROUGH THIS CALL.  THESE
C               INCLUDE THE JOINT POSITION COMMANDS, THE CONTROL OF
C               ALL OF THE JOINT BRAKES, AND THE VOLTAGE LEVEL SIGNALS
C               ON THE SPECIFIED OUTPUT CHANNELS.
C
      SUBROUTINE SERVO(SD)
      IMPLICIT INTEGER(B-Z)
      COMMON/ARMBUF/INBUF(64),OUTBUF(64)
      COMMON/END/PNT
      COMMON/OUT/JPOS(8)
      DIMENSION ERRT(12),ERR(6),INP(6)
      EQUIVALENCE(EABORT,INBUF(28))
      DATA ERRT/700,700,1500,700,1000,700,200,100,360,200,280,100/
C
C   THE FLAG 'PNT' IS TESTED TO SEE WHICH GROUP OF DELTA JOINT VALUES
C       FROM 'ERRT' IS TO BE USED.  IF 'PNT' IS EQUAL TO ZERO, THEN
C       THE ONLY REQUIREMENT ON THE ACTUAL POSITION OF ALL
C       OF THE JOINTS IS THAT DIFFERENCE BETWEEN THEIR PRESENT POSITION
C       AND THEIR NEXT COMMANDED POSITION NOT BE GREATER THAN THE
```

```
C       DISTANCE THAT THEY CAN MOVE IN THAT NEXT INCREMENT.  IF 'PNT' IS
C       EQUAL TO ONE, THEN THIS SUBROUTINE WILL LOOP, SENDING THE SAME
C       COMMANDED POSITION VALUES UNTIL THE ACTUAL POSITION OF
C       THE JOINTS IS WITHIN SOME MINIMUM DISTANCE OF THESE VALUES
C       (THESE MINIMUM VALUES ARE IN 'ERRT(7-12)').
C
        KS=0
        IF(PNT.EQ.0)GOTO 55
        PNT=0
        KS=6
55      CALL ARMIN
        KP=0
C
C   WHEN THE ACTUAL POSITION VALUES OF THE JOINTS ARE READ IN,
C       A CORRECTION FACTOR IS USED TO COMPENSATE FOR THE ERRORS
C       INTRODUCED IN THE CONVERSION ELECTRONICS IN THE INTERFACE
C       AND THE HARDWARE SERVO SYSTEM.
C
        INP(1)=INBUF(5)*2.+16361
        INP(2)=INBUF(8)*1.9758+16394
        INP(3)=INBUF(11)*2.002+16369
        INP(4)=INBUF(13)*2.033+16370
        INP(5)=INBUF(15)*1.9808+16270
        INP(6)=INBUF(17)*1.994+16349
        DO 710 KKK=1,6
        ERR(KKK)=INP(KKK)-OUTBUF(KKK)
        KM=KS+KKK
        IF(ERR(KKK).GT.ERRT(KM))GOTO 58
        IF(ERR(KKK).LT.-ERRT(KM))GOTO 58
710     CONTINUE
        GOTO 77
58      KP=1
77      IF(EABORT.LT.-2000)GOTO 40
C
C   A CALL IS MADE TO 'ARMIN' FOR SWITCH 34.  IF THIS SWITCH IS
C       UP, THE ARM WILL STOP AT ITS PRESENT POSITION AND REMAIN
C       THERE UNTIL THE SWITCH IS FLIPPED DOWN.
        IF(INBUF(34).LT.-2000)GOTO 55
C
C   THE VARIABLE 'SD' IS USED TO BRANCH TO THE APPROPRIATE CODE
C
        IF(SD)10,20,30
C
C   FOR 'SD'=-1, ZERO ALL OF THE OUTPUT CHANNELS USED BY THE 'SEND'
C       FUNCTION COMMAND.
C
10      DO 15 K=9,16
15      OUTBUF(K)=16383
        GOTO 35
C
C   FOR 'SD'=0, OUTPUT THE SEVEN JOINT POSITION COMMANDS AND
C       THE VALUE (JPOS(8)) FOR CONTROLLING THE BRAKES.
C
20      DO 25 JK=1,8
25      OUTBUF(JK)=JPOS(JK)
        CALL ARMOUT(8)
```

```
        GOTO 45
C
C   FOR 'SD'= SOME NUMBER BETWEEN ONE AND EIGHT, SET 'CH' EQUAL TO THIS
C       NUMBER PLUS EIGHT AND OUTPUT A +5 VOLT SIGNAL ON THIS CHANNEL.
C       HERE, THE NUMBER 'SD' WILL HAVE BEEN SPECIFIED BY THE
C       'SEND' FUNCTION COMMAND.
C
30      CH=8+SD
        OUTBUF(CH)=8192
35      CALL ARMOUT(16)
45      IF(KP.NE.0)GOTO 55
40      RETURN
        END
```

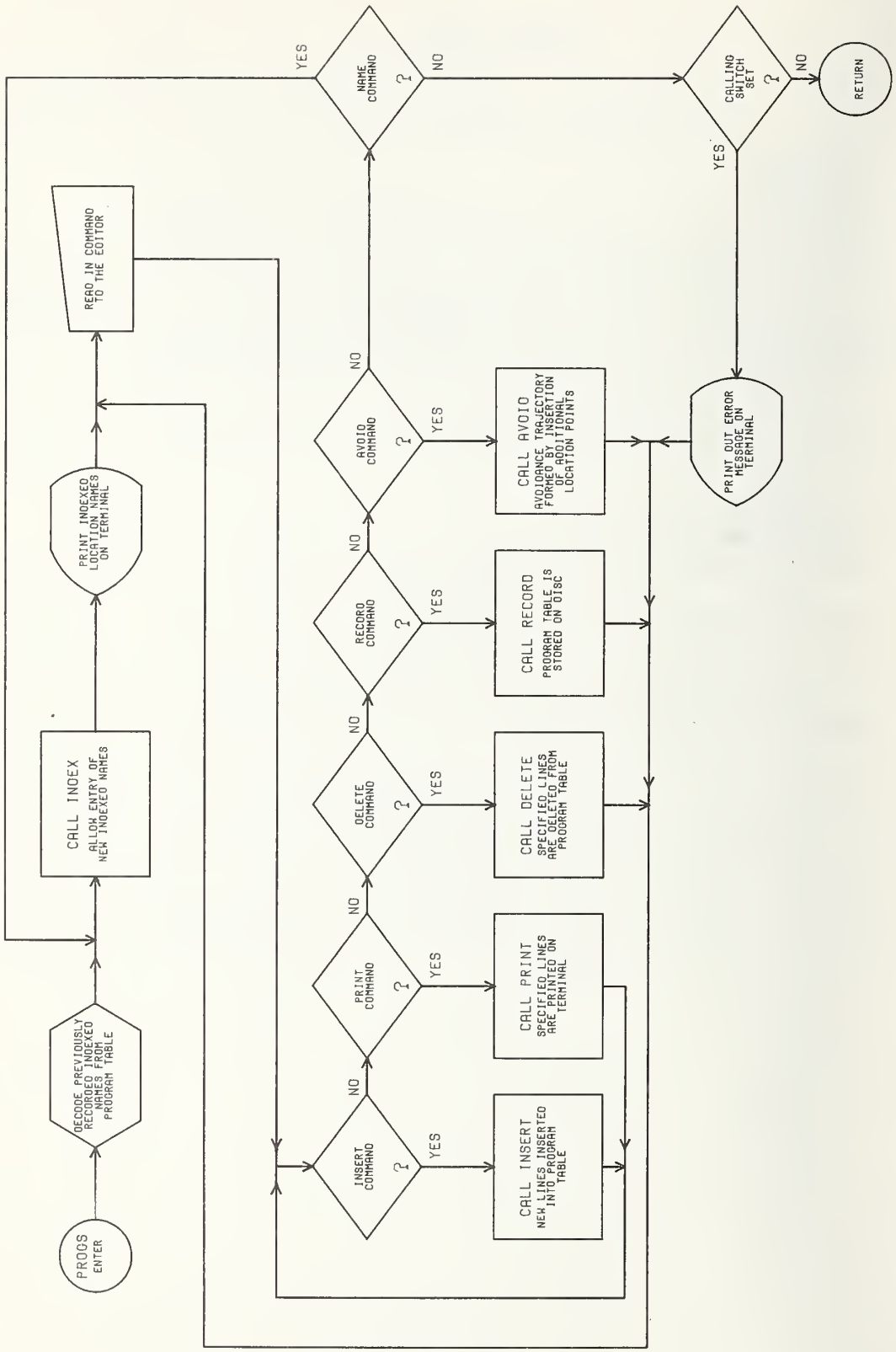
```
        .TITLE ARMOUT FOR LOW LEVEL ARM OUTPUT
        .IDENT /RMR01/
;
;       CALLING SEQUENCE IS:
;
;       CALL ARMOUT(COUNT,[BUFFER])
;                       WHERE [BUFFER] IS OPTIONAL
;                       IF NOT THERE, OUTBUF IS USED.
        .GLOBL  VAL             ;THE 11/45 IS TOO FAST, ARMIN AND
                                ;ARMOUT MUST SLOW DOWN, VAL IS A
                                ;WORD TO HELP US DO THAT.
ARMOUT::
        MOV     VAL,R0
        MOVB    @R5,R4          ;GET THE NUMBER OF ARGUMENTS
                                ;IN THE CALLING SEQUENCE
        MOV     @2(R5),R3       ;GET THE NUMBER OF POINTS TO
                                ;SEND TO THE ARM.
        MOV     #OUTBUF,R2      ;ASSUME OUTBUF IS USED
                                ;FOR ARM OUTPUT
        DEC     R4
        BEQ     1$              ;THE NUMBER OF ARGUMENTS WAS ONE
                                ;SO USE OUTBUF, OTHERWISE
        MOV     4(R5),R2        ;USE THE ARRAY SPECIFIED.
1$:     MOV     #167772,R4      ;POINT TO THE ARM INTERFACE
        MOV     #-1,@R4         ;AND RESET IT
2$:     DEC     R0
        BPL     2$
        MOV     VAL,R0          ;11/45 IS TOO FAST, SO, TIMEOUT
        MOV     (R2)+,@R4       ;OUTPUT FROM THE BUFFER
        DEC     R3              ;COUNT DOWN ON NUMBER OF POINTS
                                ;TO SEND TO THE ARM
        BGT     2$              ;THEN SEND THEM
        MOV     #-1,@R4         ;RESET AFTER
                                ;THE WRITE (6-MAY-74)
                                ;THIS CAUSES LESS CURRENT
                                ;DRAIN ON THE POWER SUPPLIES IN THE
                                ;D/A CONVERTER
        RTS     R5              ;THEN GO HOME
;
;       IMPLICIT INTEGER(A-Z)
;       COMMON/ARMBUF/INBUF(64),OUTBUF(64)
;
        .CSECT ARMBUF
INBUF=.
OUTBUF=.+200
.=.+400
        .CSECT
        .END
```

```
        .TITLE ARMIN FOR LOW LEVEL ARM INPUT
        .IDENT /RMR01/
;
; ARMIN AND ARMOUT HAVE SIMILAR DOCUMENTATION.  THE TWO PROGRAMS
; SHOULD BE READ TOGETHER AS SOME DOCUMENTATION EXISTS IN
; ONE AND NOT THE OTHER.
;
        .CSECT
;
; THE ARM IS CONTROLLED BY A DR11-C INTERFACE TO THE PDP-11/45
; UNIBUS.  ALTHOUGH ALL THE INTERRUPT FACILITY AVAILABLE WITH
; THE DR11-C IS INTACT, NO INTERRUPTS ARE USED WITH THIS VERSION
; OF THE ARM SOFTWARE.  THE GENERAL PHILOSOPHY OF THE SOFTWARE
; IS SUCH THAT DATA TRANSFER TO AND FROM THE ARM WILL OCCURE
; WHENEVER A PROGRAM WANTS AND/OR HAS THE TIME.
;
;       THE DR11-C HAS A UNIBUS ADDRESS OF 167770
; THIS IS THE CONTROL REGISTER. 167772 IS THE TRANSMIT REGISTER
; AND 167774 IS THE RECEIVE REGISTER.  THE A/D CONVERTER
; IS RESET WHENEVER IT SEES BIT 15 SET.  AT LOCATION 24, THE A/D CONVERTER
; IS RESET.
;     ALL 64 CHANNELS ARE READ INTO THE COMMON BLOCK NAMED ARMBUF
; THIS IS FORTRAN  COMPATIBLE  COMMON.  ALSO THE SUBROUTINE LINKAGE
; IS  COMPATIBLE .
;
VAL::   .WORD   25              ;TIME OUT VALUE
                                ;IS SET AT 10 WHICH CAUSES THIS
                                ;ROUTINE TO WAIT (SEE LOOP 1$)
                                ;LONG ENOUGH FOR THE A/D CONVERTER
                                ;TO SETTLE DOWN BETWEEN READS.
ARMIN::
        MOV     VAL,R0          ;THE 11/45 RUNS TOO FAST
                                ;SO WE WILL USE R0 TO
                                ;TIMEOUT BETWEEN READS.
        MOV     #167772,R2      ;LOAD THE CSR OUTPUT REG
        MOV     #-1,(R2)+       ;WITH A RESET AND POINT
                                ;TO THE INPUT SIDE
        MOV     #64.,R4         ;COUNT UP TO 64 POINTS
        MOV     #INBUF,R3       ;POINT TO THE FIRST WORD
1$:     DEC     R0              ;COUNT DOWN FOR 11/45 TIMEOUT
        BPL     1$
        MOV     VAL,R0
        MOV     @R2,(R3)+       ;LOAD THE INBUF WITH 64 POINTS
        DEC     R4
        BGT     1$
        RTS     R5              ;THEN RETURN
;
;       IMPLICIT INTEGER(A-Z)
;       COMMON /ARMBUF/INBUF(64),OUTBUF(64)
;
; WHEN USED WITH FORTRN, COMPILE WITH THE /ON SWITCH
        .CSECT ARMBUF
INBUF=.
OUTBUF=.+200
.=.+400
        .CSECT
        .END
```

Program Module
(Module #2)

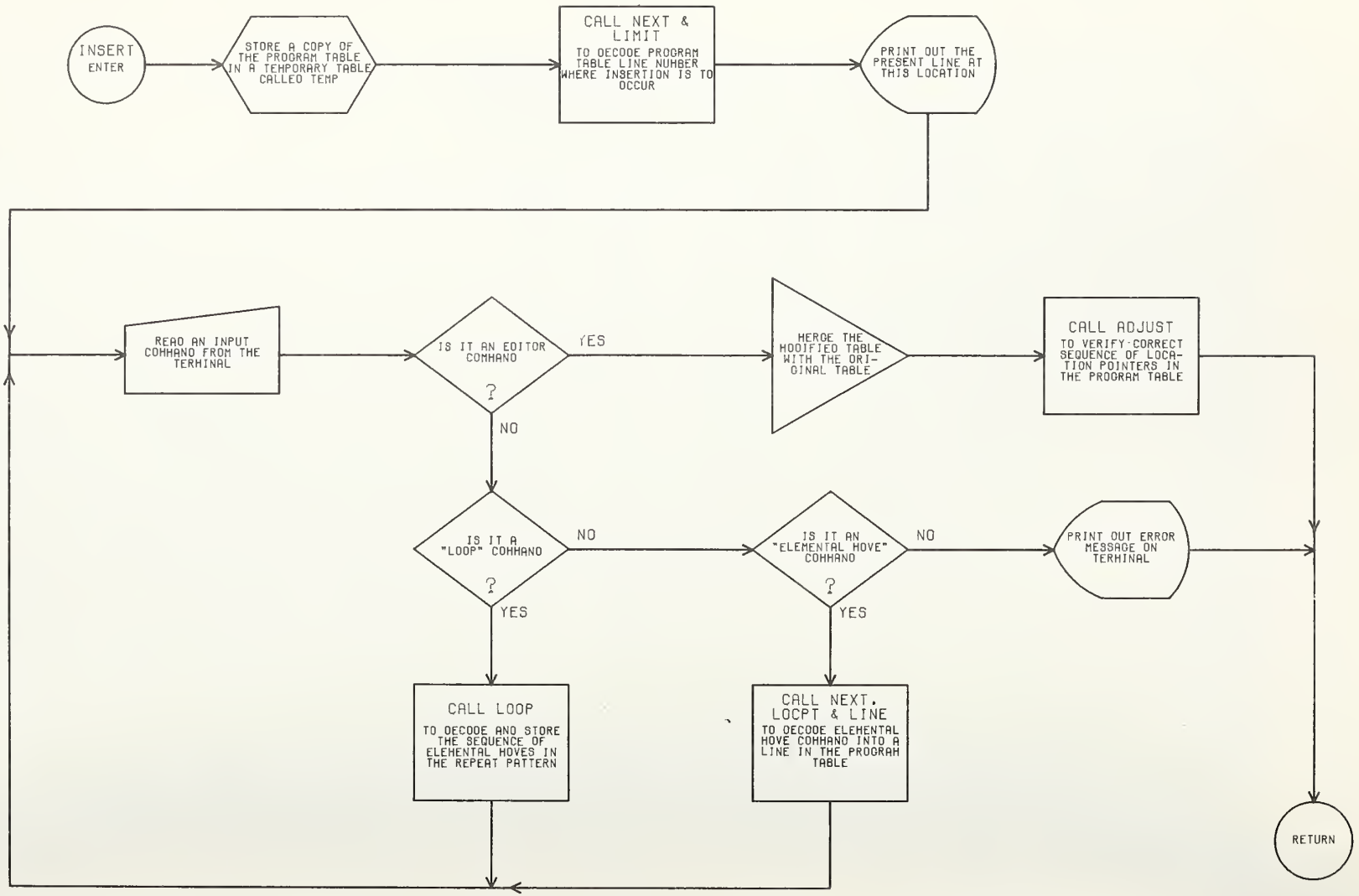
Editor Commands

Additional Support Subroutines


PROGS ENTER
DECODE PREVIOUSLY RECORDED INDEXED NAMES FROM PROGRAM TABLE
CALL INDEX
ALLOW ENTRY OF NEW INDEXED NAMES
PRINT INDEXED LOCATION NAMES ON TERMINAL
READ IN COMMAND TO THE EDITOR
INSERT COMMAND ?
YES
NO
CALL INSERT
NEW LINES INSERTED INTO PROGRAM TABLE
PRINT COMMAND ?
CALL PRINT
SPECIFIED LINES ARE PRINTED ON TERMINAL
DELETE COMMAND ?
CALL DELETE
SPECIFIED LINES ARE DELETED FROM PROGRAM TABLE
RECORD COMMAND ?
CALL RECORD
PROGRAM TABLE IS STORED ON DISC
AVOID COMMAND ?
CALL AVOID
AVOIDANCE TRAJECTORY FORMED BY INSERTION OF ADDITIONAL LOCATION POINTS
NAME COMMAND ?
CALLING SWITCH SET ?
PRINT OUT ERROR MESSAGE ON TERMINAL
RETURN

```
C --- SUBROUTINE : PROGS
C
C --- ARGUMENTS :
C
C --- CALLED BY : OPRO
C
C --- CALLS SUBROUTINES : INDEX  INSERT  DELETE  PRINT  RECORD
C                         AVOID  ARMIN
C
C --- INPUT DATA : PROG(105,6)= THE PROGRAM MODULE THAT IS TO BE MODIFIED
C                  CS(50)     = THE ASCII CHARACTER STRING THAT CONTAINS
C                               THE 'EDITOR' COMMANDS.
C
C --- OUTPUT DATA : PROG(105,6)= THE EDITED PROGRAM MODULE.
C
C --- FUNCTION: EDITS AN OLD PROGRAM FROM THE DISC, OR ENTERS A
C               NEW PROGRAM THROUGH THE USE OF THE PROPER
C               EDITOR COMMANDS.
C
        SUBROUTINE PROGS
        IMPLICIT INTEGER(B-Z)
        COMMON/PMOD/PROG(105,6),ENDP,BRNCH(6),M
        COMMON/IND/IN(30),FL(6),NAME(3),NPT(3,20)
        COMMON/CMD/CS(50),PN,PLN
        COMMON/ARMBUF/INBUF(64),OUTBUF(64)
        EQUIVALENCE(RET,INBUF(30))
C
C   THE SPECIFIED PROGRAM MODULE IS READ IN FROM THE DISC BY THE CALL
C       TO 'RDMOD' IN 'EXPRO'.  THE INDEXED NAMES ARE STORED IN THE
C       CHARACTER STRING IN(30).
C
500     M=0
        DO 204 J=101,105
        DO 204 L=1,6
        M=M+1
204     IN(M)=PROG(J,L)
        PLN=1
C
C   THE CALL TO 'INDEX' PRINTS THE PRESENT INDEXED LOCATION NAMES
C       AND ALLOWS THE ENTERING OF NEW INDEXED NAMES.
C
        CALL INDEX(0)
        WRITE(6,125)
125     FORMAT(8X'EDITOR IS NOW AVAILABLE FOR PROGRAM ENTRY',/)
C
C   THE CURRENT INDEXED NAMES ARE STORED IN THE LAST
C       FIVE LINES OF THE PROGRAM MODULE.
C
        M=0
        DO 210 J=101,105
        DO 210 L=1,6
        M=M+1
210     PROG(J,L)=IN(M)
C
C   THE COMMAND FROM THE TERMINAL IS READ INTO THE
C       CHARACTER STRING CS(50) AND TESTED TO SEE IF THE FIRST
```

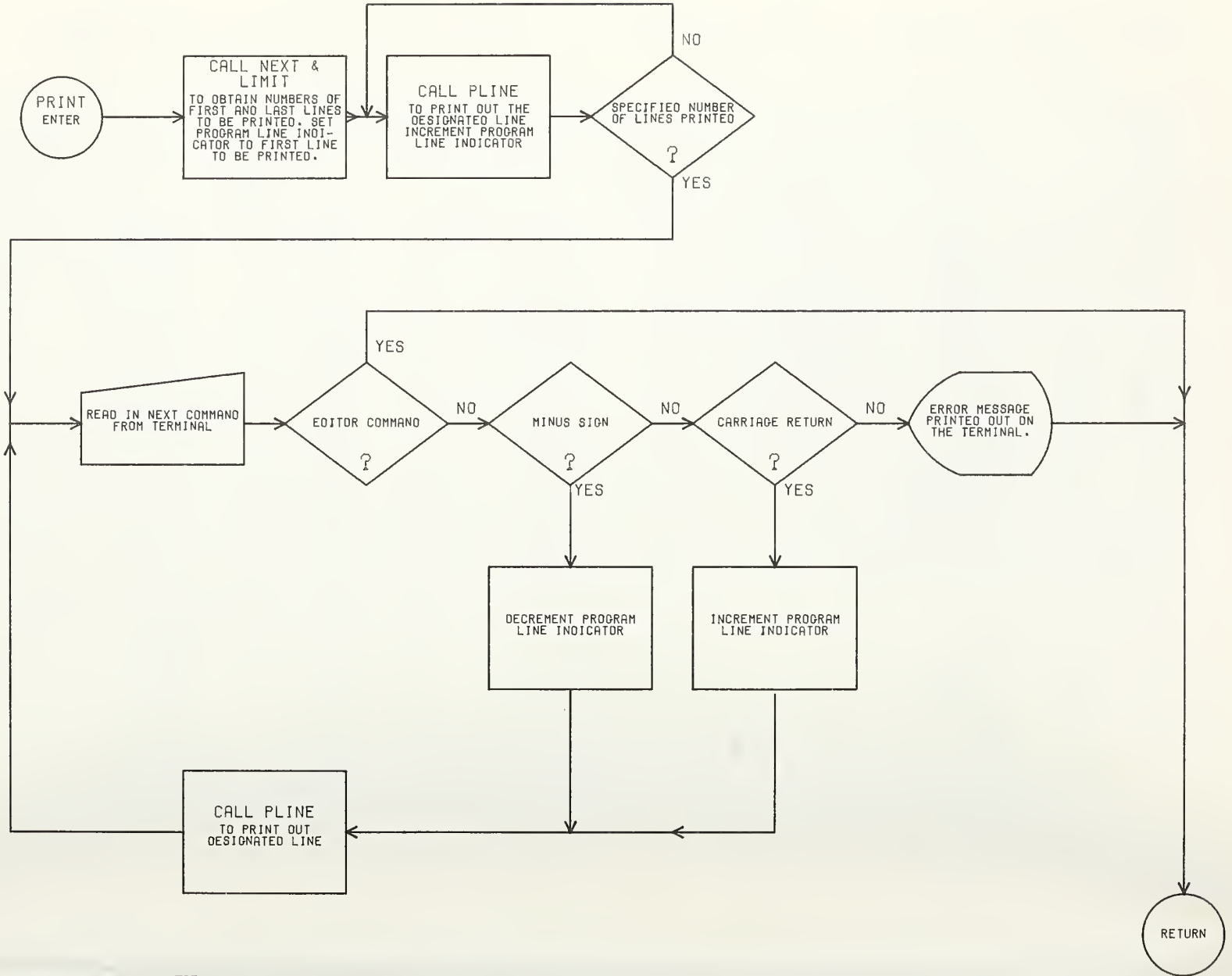
```
C       LETTER IS THE ASCII CODE NUMBER FOR I(INSERT),D(DELETE),P(PRINT),
C       R(RECORD),OR A(AVOID).  THE PROPER SUBROUTINE IS CALLED FOR
C       THE IDENTIFIED CODE LETTER.  IF NO VALID LETTER IS FOUND,
C       THE MESSAGE 'ILLEGAL COMMAND' IS PRINTED AND THE PROGRAM WAITS
C       FOR THE NEXT COMMAND.
C
99      READ(6,200)(CS(N),N=1,50)
200     FORMAT(50(A1))
        CALL ARMIN
        IF(RET.GT.-2000)GOTO 40
        PN=1
77      IF(CS(PN).NE.8265)GOTO 220
        CALL INSERT
220     IF(CS(PN).NE.8260)GOTO 221
        CALL DELETE
        GOTO 99
221     IF(CS(PN).NE.8272)GOTO 222
        CALL PRINT
        GOTO 77
222     IF(CS(PN).NE.8274)GOTO 223
        CALL RECORD(0,1)
        GOTO 405
223     IF(CS(PN).NE.8270)GOTO 224
        GOTO 500
224     IF(CS(PN).NE.8257)GOTO 255
        CALL AVOID
        GOTO 99
255     IF(RET.GT.-2000)GOTO 40
100     WRITE(6,101)
101     FORMAT(8X'ILLEGAL COMMAND')
        GOTO 99
405     WRITE(6,225)
225     FORMAT(8X,'PROGRAM HAS BEEN STORED ON DISC')
        GOTO 99
40      RETURN
        END
```


INSERT
ENTER
STORE A COPY OF THE PROGRAM TABLE IN A TEMPORARY TABLE CALLED TEMP
CALL NEXT & LIMIT
TO DECODE PROGRAM TABLE LINE NUMBER WHERE INSERTION IS TO OCCUR
PRINT OUT THE PRESENT LINE AT THIS LOCATION
READ AN INPUT COMMAND FROM THE TERMINAL
IS IT AN EDITOR COMMAND ?
YES
MERGE THE MODIFIED TABLE WITH THE ORIGINAL TABLE
CALL ADJUST
TO VERIFY CORRECT SEQUENCE OF LOCATION POINTERS IN THE PROGRAM TABLE
NO
IS IT A "LOOP" COMMAND ?
NO
IS IT AN "ELEMENTAL MOVE" COMMAND ?
NO
PRINT OUT ERROR MESSAGE ON TERMINAL
YES
YES
CALL LOOP
TO DECODE AND STORE THE SEQUENCE OF ELEMENTAL MOVES IN THE REPEAT PATTERN
CALL NEXT, LOCPT & LINE
TO DECODE ELEMENTAL MOVE COMMAND INTO A LINE IN THE PROGRAM TABLE
RETURN

```
C --- SUBROUTINE : INSERT
C
C --- ARGUMENTS :
C
C --- CALLED BY : PROGS
C
C --- CALLS SUBROUTINES : NEXT  LIMIT  PLINE  LOOP  LOCPT  LINE  ADJUST
C
C --- INPUT DATA : CS(50)      = THE CHARACTER STRING THAT CONTAINS THE
C                                 INPUT COMMAND STATEMENT.
C                   BEG        = THE LINE NUMBER IN THE PROGRAM MODULE
C                                 WHERE THE INSERTION OCCURS [LIMIT].
C                   PLN        = THE PRESENT PROGRAM LINE NUMBER,
C                                 HERE, SET EQUAL TO 'BEG'.
C                   BN         = THE LOCATION TABLE POINTER FOR THE
C                                 FIRST LOCATION IN THE PROGRAM MODULE,
C                                 ENTERED BY 'START' COMMAND [LOCPT].
C                   PROG(PLN,6)= THE DECODED NEW LINE IN THE PROGRAM
C                                 MODULE [LINE].
C
C --- OUTPUT DATA : PROG(105,6) =  THE UPDATED PROGRAM MODULE
C
C --- FUNCTION: TO INSERT NEW LINES AT THE SPECIFIED PROGRAM LINE,
C               SLIDING BACK THE REST OF THE PROGRAM TO
C               ACCOMMODATE  THESE ADDITIONAL LINES.
C
        SUBROUTINE INSERT
        IMPLICIT INTEGER(B-Z)
        COMMON/PMOD/PROG(105,6),ENDP,BRNCH(6),M
        COMMON/PTN/BN,EN
        COMMON/BED/BEG,FIN,DIF,ERR
        COMMON/CMD/CS(50),PN,PLN
        DIMENSION TEMP(100,6)
        ERR=0
C
C   STORE PRESENT PROGRAM MODULE (PROG(105,6))IN A TEMPORARY MATRIX
C       CALLED TEMP(100,6).
C
        DO 50 K=1,100
        DO 50 F=1,6
50      TEMP(K,F)=PROG(K,F)
C
C   IDENTIFY AND DECODE THE PROGRAM LINE NUMBER WHERE THE
C       INSERTION IS TO OCCUR.
C
500     CALL NEXT
        IF(PN.GT.50)GOTO 27
        CALL LIMIT
        IF(ERR.EQ.1)GOTO 100
        PLN=BEG
27      STAR=PLN
        GOTO 160
C
C   ERROR MESSAGE TO BE PRINTED OUT IF A INCORRECT COMMAND IS
C       ENTERED IN THIS SUBROUTINE.
C
```

```
100     WRITE(6,101)
101     FORMAT(8X,'ILLEGAL FORMAT FOR *** INSERT (J) ***')
        GOTO 40
C
C   PRINT OUT ON THE TERMINAL THE PRESENT PROGRAM LINE (PLN).
C
160     CALL PLINE(PLN)
C
C   READ IN A NEW LINE (CS(50)) AND TEST ON THE FIRST CHARACTER,
C       RETURN TO CALLING PROGRAM IF IT IS AN EDITOR COMMAND (INSERT,
C       NAME,AVOID,DELETE,PRINT, OR RECORD).
170     READ(6,171)(CS(L),L=1,50)
171     FORMAT(50(A1))
        PN=1
        IF(CS(PN).EQ.8265)GOTO 400
        IF(CS(PN).EQ.8270)GOTO 400
        IF(CS(PN).EQ.8257)GOTO 400
        IF(CS(PN).EQ.8260)GOTO 400
        IF(CS(PN).EQ.8272)GOTO 400
        IF(CS(PN).EQ.8274)GOTO 400
C
C   TEST FOR A 'LOOP' COMMAND.
C
        IF(CS(PN).NE.8268)GOTO 182
        CALL LOOP
        PLN=PLN+1
        GOTO 170
C
C   TEST FOR A 'START' COMMAND, IE. THE ENTERING OF THE FIRST
C       LOCATION POINT IN THE PROGRAM MODULE.
C
182     IF(CS(PN).NE.8275)GOTO 183
        CALL NEXT
        CALL LOCPT(1)
        PROG(1,1)=BN
        STAR=1
        PLN=1
        GOTO 170
C
C   TEST FOR A 'GOTO' ELEMENTAL MOVE COMMAND, CALL 'LINE' TO
C       RECORD AND DECODE THE CORRECT VALUES FOR THE PROGRAM MODULE.
C
183     IF(CS(PN).NE.8263)GOTO 100
        CALL LINE
        PLN=PLN+1
        GOTO 170
C
C   PUSH BACK THE REST OF THE PROGRAM LINES TO ACCOMMODATE THE
C       NEWLY INSERTED LINES, AND STORE UPDATED PROGRAM IN THE
C       PROGRAM MODULE PROG(105,6).
C
400     IF(PLN.NE.1)GOTO 401
        GOTO 40
401     END=PLN-STAR
        PT=101-PLN
        IF(PLN.NE.1)GOTO 60
```

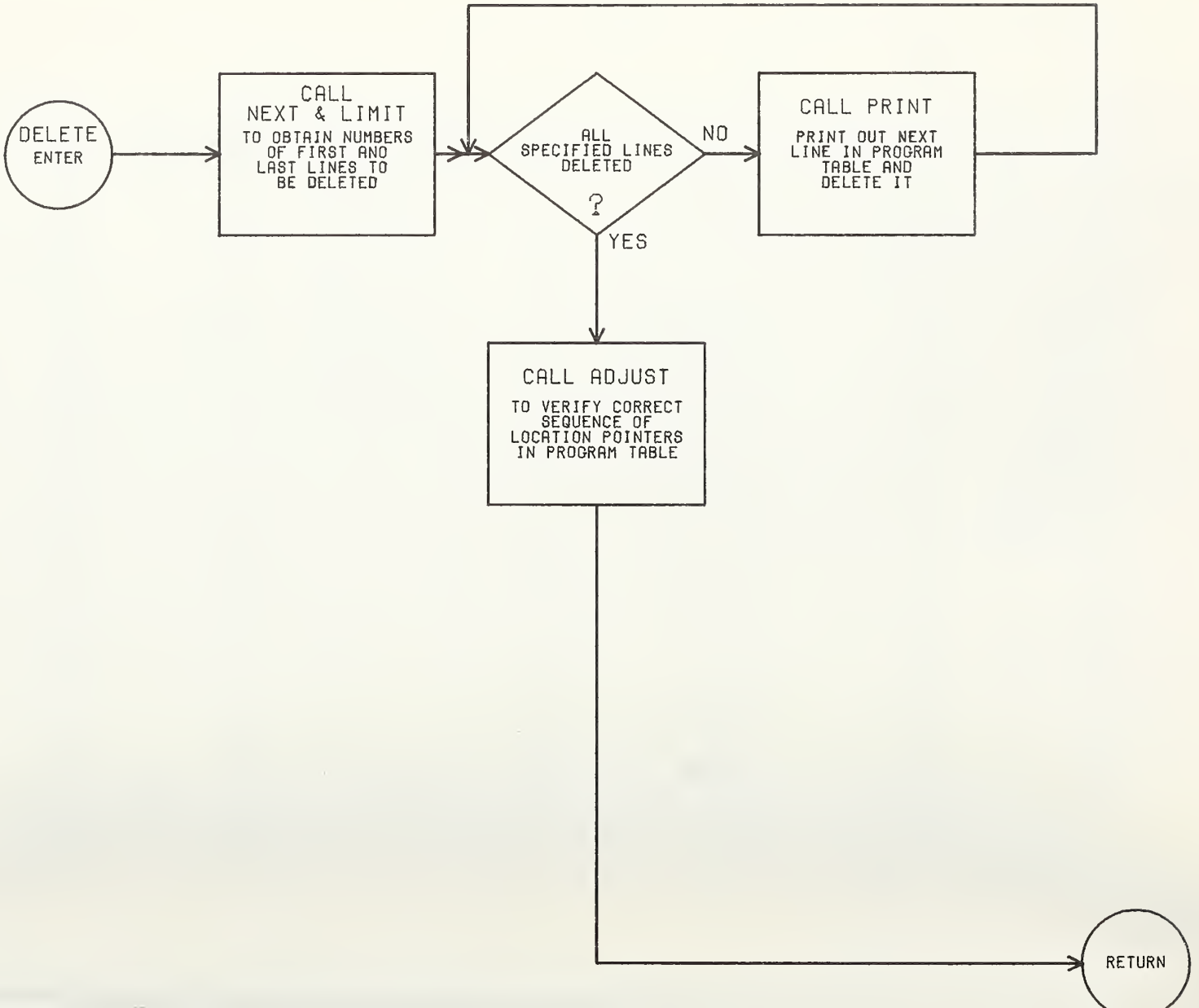
```
          PROG(1,2)=TEMP(1,1)
          END=1
          PT=99
60        DO 450 MB=1,PT
          FD=101-MB
          DIF=FD-END
          DO 450 MT=1,6
450       PROG(FD,MT)=TEMP(DIF,MT)
451       CALL ADJUST
          PLN=STAR
40        RETURN
          END
```


PRINT
ENTER
CALL NEXT & LIMIT
TO OBTAIN NUMBERS OF FIRST AND LAST LINES TO BE PRINTED. SET PROGRAM LINE INDICATOR TO FIRST LINE TO BE PRINTED.
CALL PLINE
TO PRINT OUT THE DESIGNATED LINE INCREMENT PROGRAM LINE INDICATOR
SPECIFIED NUMBER OF LINES PRINTED
?
NO
YES
READ IN NEXT COMMAND FROM TERMINAL
EDITOR COMMAND
?
YES
NO
MINUS SIGN
?
YES
NO
CARRIAGE RETURN
?
YES
NO
ERROR MESSAGE PRINTED OUT ON THE TERMINAL.
DECREMENT PROGRAM LINE INDICATOR
INCREMENT PROGRAM LINE INDICATOR
CALL PLINE
TO PRINT OUT DESIGNATED LINE
RETURN

```
C --- SUBROUTINE : PRINT
C
C --- ARGUMENTS :
C
C --- CALLED BY : PROGS
C
C --- CALLS SUBROUTINES : NEXT  LIMIT  PLINE
C
C --- INPUT DATA : CS(50)      = CHARACTER STRING THAT CONTAINS THE
C                                 EDITOR COMMAND TO PRINT A LINE(S)
C                  PLN         = THE NUMBER OF THE PRESENT PROGRAM LINE
C                                 (MAY NOT BE THE LINE SPECIFIED IN THE
C                                 'PRINT' COMMAND).
C                  BEG         = THE NUMBER (IF SPECIFIED IN THE PRINT
C                                 COMMAND) OF THE FIRST LINE TO BE
C                                 PRINTED [FROM 'LIMIT'].
C                  DIF         = THE NUMBER OF LINES TO BE PRINTED STARTING
C                                 FROM LINE 'BEG' (IF MORE THAN ONE LINE
C                                 HAS BEEN SPECIFIED IN THE COMMAND).
C
C --- OUTPUT DATA : PLN           =  THE NUMBER OF THE PROGRAM LINE TO BE
C                                     PRINTED OUT ON THE TERMINAL (SENT
C                                     TO THE SUBROUTINE 'PLINE').
C
C --- FUNCTION: DECODES THE PRINT COMMAND TO DETERMINE HOW MANY LINES ARE
C               TO BE PRINTED AND WHAT ARE THEIR LINE NUMBERS IN THE
C               PROGRAM MODULE.
C
        SUBROUTINE PRINT
        IMPLICIT INTEGER(B-Z)
        COMMON/ARMBUF/INBUF(64),OUTBUF(64)
        COMMON/BED/BEG,FIN,DIF,ERR
        COMMON/CMD/CS(50),PN,PLN
        ERR=0
C
C   TEST TO SEE IF THE LINES HAVE BEEN SPECIFIED IN THE 'PRINT'
C       COMMAND (IE. IS THEIR ANY CHARACTERS AFTER THE WORD
C       'PRINT' IN THE CHARACTER STRING CS(50)).  IF SO, THEN
C       GO TO STATEMENT 20.
C
60      CALL NEXT
        IF(PN.LT.51)GOTO 20
C
C   IF NO  LINE NUMBERS ARE SPECIFIED IN THE 'PRINT'
C       COMMAND THEN THE NUMBER OF LINES TO BE PRINTED (DIF) IS SET
C       EQUAL TO ONE .
C
        PLN=PLN-1
        DIF=1
        GOTO 70
C
C   THIS IS THE ERROR MESSAGE DISPLAYED ON THE TERMINAL IF AN INCORRECT
C       FORMAT IS USED.
C
100     WRITE(6,101)
101     FORMAT(8X'FORMAT ERROR IN  *** PRINT (J-K) *** STATEMENT')
```

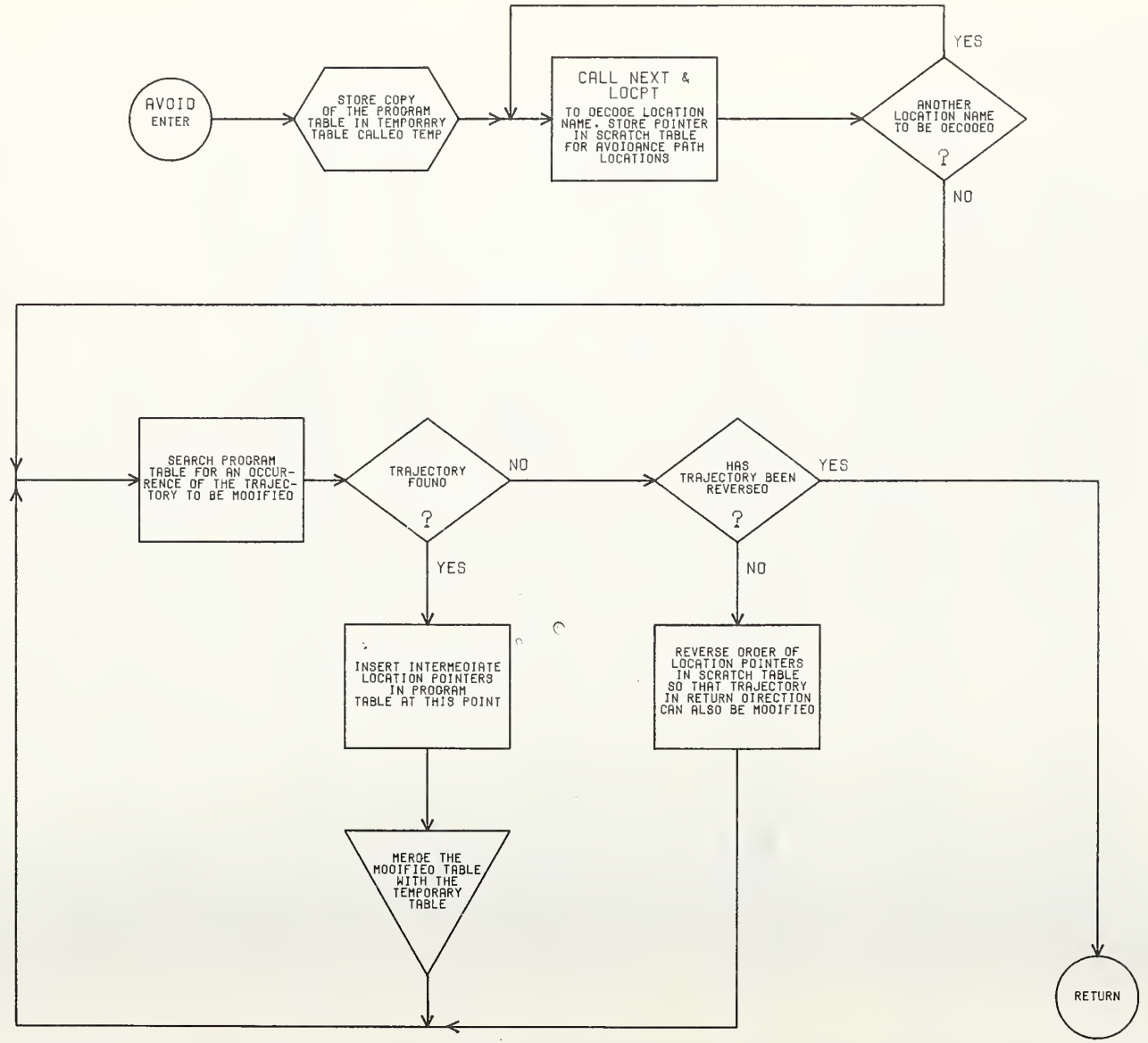
```
        GOTO 40
C
C   IF LINES ARE SPECIFIED IN THE 'PRINT' COMMAND SET PLN EQUAL
C       TO THE FIRST LINE TO BE PRINTED (IE. SET PLN EQUAL TO
C       BEG FROM 'LIMIT') AND CALCULATE THE NUMBER OF LINES TO BE
C       PRINTED 'DIF' [FROM 'LIMIT'].
C
20      CALL LIMIT
        IF(ERR.EQ.1)GOTO 100
        PLN=BEG-1
C
C   EACH TIME THROUGH THIS LOOP A LINE FROM THE PROGRAM MODULE IS
C       PRINTED OUT (BY A CALL TO 'PLINE') AND THE LINE NUMBER 'PLN'
C       IS INCREMENTED.
C
70      DO 50 K=1,DIF
        PLN=PLN+1
50      CALL PLINE(PLN)
C
C   THE NEXT COMMAND IS READ INTO CS(50) FROM THE TERMINAL AND THE
C       FIRST LETTER OF THE COMMAND WORD TESTED TO SEE IF IT IS
C       A D(DELETE), I(INSERT), N(NAME), P(PRINT), R(RECORD),
C       A(AVOID).  IF ONE OF THESE EDITOR COMMANDS, THEN
C       RETURN TO THE CALLING PROGRAM 'PROGS'.
C
65      READ(6,66)(CS(RN),RN=1,50)
66      FORMAT(50(A1))
C
C   IF THE CALLING SWITCH (SWITCH 30) IS RESET, THEN RETURN TO
C       CALLING PROGRAM.
C
        CALL ARMIN
        IF(INBUF(30).GT.-2000)GOTO 40
        PN=1
        IF(CS(PN).EQ.8260)GOTO 40
        IF(CS(PN).EQ.8265)GOTO 40
        IF(CS(PN).EQ.8270)GOTO 40
        IF(CS(PN).EQ.8272)GOTO 40
        IF(CS(PN).EQ.8274)GOTO 40
        IF(CS(PN).EQ.8257)GOTO 40
C
C   TEST IF A MINUS SIGN WAS ENTERED INDICATING THE PREVIOUS LINE IS
C       TO BE PRINTED.  THIS IS DONE BY SUBTRACTING ONE FROM THE
C       PRESENT LINE NUMBER PLN.
C
        IF(CS(PN).NE.8237)GOTO 51
        PLN=PLN-1
        GOTO 31
C
C   TEST TO SEE IF ONLY A CARRIAGE RETURN WAS ENTERED INDICATING THAT
C       THE NEXT LINE OF THE PROGRAM MODULE IS TO BE PRINTED.  THIS
C       IS DONE BY ADDING ONE TO THE PRESENT PROGRAM LINE NUMBER AND
C       CALLING 'PLINE'.
C
51      IF(CS(PN).NE.8224)GOTO 100
        PLN=PLN+1
```

```
31      CALL PLINE(PLN)
        GOTO 65
40      RETURN
        END
```


DELETE
ENTER
CALL
NEXT & LIMIT
TO OBTAIN NUMBERS
OF FIRST AND
LAST LINES TO
BE DELETED
ALL
SPECIFIED LINES
DELETED
?
NO
YES
CALL PRINT
PRINT OUT NEXT
LINE IN PROGRAM
TABLE AND
DELETE IT
CALL ADJUST
TO VERIFY CORRECT
SEQUENCE OF
LOCATION POINTERS
IN PROGRAM TABLE
RETURN

```
C --- SUBROUTINE : DELETE
C
C --- ARGUMENTS :
C
C --- CALLED BY : PROGS
C
C --- CALLS SUBROUTINES : NEXT  LIMIT  PLINE  ADJUST
C
C --- INPUT DATA : PROG(105,6)= THE PRESENT PROGRAM MODULE
C                  CS(50)     = THE CHARACTER STRING THAT CONTAINS
C                               THE 'DELETE' COMMAND.
C                  BEG        = THE NUMBER OF THE FIRST LINE TO BE
C                               DELETED [FROM 'LIMIT'].
C                  DIF        = THE NUMBER OF LINES TO BE DELETED AS
C                               SPECIFIED IN THE 'DELETE' COMMAND [FROM
C                               'LIMIT'].
C                  FIN        = THE NUMBER OF THE LAST LINE TO BE DELETED
C                               [FROM 'LIMIT'].
C                  PN         = A POINTER THAT INDICATES THE POSITION
C                               IN THE CHARACTER STRING CS(50) OF THE NEXT
C                               PIECE OF INFORMATION [FROM NEXT].
C
C --- OUTPUT DATA : PROG(105,6) =  THE MODIFIED PROGRAM MODULE AFTER
C                                   THE SPECIFIED LINES HAVE BEEN DELETED
C                                   AND THE REST OF THE PROGRAM CLOSED
C                                   UP AROUND THE DELETED LINES.
C                    PLN          =  THE POINTER TO THE PRESENT PROGRAM
C                                   LINE.
C
C --- FUNCTION: DELETES THE SPECIFIED LINES FROM THE PROGRAM, PRINTS
C               OUT THE DELETED LINES ON THE TERMINAL, AND CLOSES
C               UP THE PROGRAM MODULE AROUND THE DELETED LINES.
C
      SUBROUTINE DELETE
      IMPLICIT INTEGER(B-Z)
      COMMON/PMOD/PROG(105,6),ENDP,BRNCH(6),M
      COMMON/BED/BEG,FIN,DIF,ERR
      COMMON/CMD/CS(50),PN,PLN
      ERR=0
C
C   ADVANCE THROUGH THE CHARACTER STRING CS(50) UNTIL THE NUMBER OF THE
C       FIRST LINE OF THE PROGRAM TO BE DELETED IS REACHED BY
C       CALLING 'NEXT'.
C
      CALL NEXT
      IF(PN.LT.51)GOTO 20
C
C   ERROR MESSAGE THAT IS PRINTED OUT IF AN INCORRECT
C       FORMAT IS DETECTED IN THE 'DELETE' COMMAND.
C
100   WRITE(6,101)
101   FORMAT(8X'FORMAT ERROR IN *** DELETE (J-K)*** STATEMENT')
      GOTO 40
C
C   THE FIRST LINE SPECIFIED TO BE DELETED IS DECODED BY A CALL TO
C       'LIMIT' AND RETURNED IN THE VARIABLE 'BEG'.  THE NUMBER
```

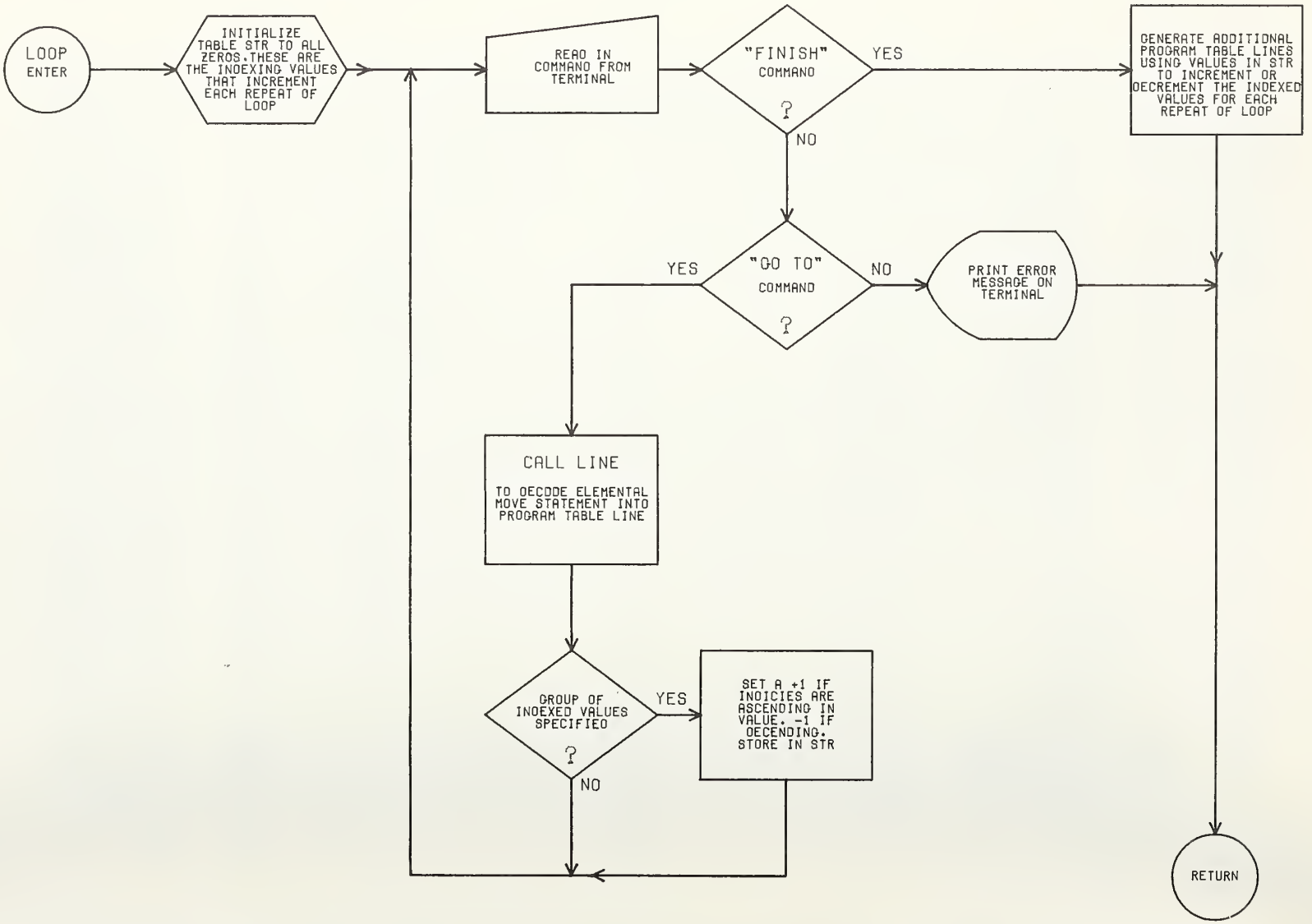
```
C       OF LINES TO BE DELETED IS ALSO DETERMINED BY 'LIMIT' AND
C       RETURNED IN THE VARIABLE 'DIF'.
C
20      CALL LIMIT
        IF(ERR.EQ.1)GOTO 100
C
C   AN INTERMEDIATE VARIABLE 'NEN' IS CREATED TO BE USED IN
C       THE CLOSING UP PROCEDURE WHERE THE REMAINDER OF THE PROGRAM
C       LINES ARE SLID FORWARD TO CLOSE THE GAP CREATED BY THE
C       DELETION OF THE SPECIFIED LINES.
C
        NEN=100-DIF
        IF(BEG.NE.1)GOTO 70
C
C   THE STARTING LOCATION OF THE PROGRAM IS SAVED IN THE VARIABLE 'TEM'
C       TO BE REINSERTED IN THE PROGRAM MODULE AFTER ALL OF
C       THE ADJUSTMENTS HAVE BEEN MADE.
C
        TEM=PROG(1,1)
C
C   THE FOLLOWING 'DO LOOP' DELETES THE SPECIFIED LINES AND CLOSES
C       UP THE PROGRAM AT THE SAME TIME BY MOVING ALL OF THE PROGRAM
C       LINES FORWARD 'DIF' NUMBER OF LINES, STARTING WITH LINE
C       NUMBER 'BEG' + 'DIF'.
C
70      DO 50 K=BEG,NEN
        JMP=K+DIF
        IF(K.GT.FIN)GOTO 30
        PLN=K
C
C   EACH LINE, AS IT IS DELETED, IS PRINTED OUT ON THE TERMINAL
C       BY A CALL TO 'PLINE'.
C
        CALL PLINE(PLN)
30      DO 50 F=1,6
50      PROG(K,F)=PROG(JMP,F)
        IF(BEG.NE.1)GOTO 71
        PROG(1,1)=TEM
C
C   ANY INCOMPLETE LINES THAT MIGHT HAVE BEEN CREATED ARE
C       ELIMINATED BY THE SUBROUTINE 'ADJUST'.
C
71      CALL ADJUST
40      RETURN
        END
```


AVOID
ENTER
STORE COPY OF THE PROGRAM TABLE IN TEMPORARY TABLE CALLED TEMP
CALL NEXT & LOCPT
TO DECODE LOCATION NAME, STORE POINTER IN SCRATCH TABLE FOR AVOIDANCE PATH LOCATIONS
ANOTHER LOCATION NAME TO BE DECODED ?
YES
NO
SEARCH PROGRAM TABLE FOR AN OCCURRENCE OF THE TRAJECTORY TO BE MODIFIED
TRAJECTORY FOUND ?
NO
YES
HAS TRAJECTORY BEEN REVERSED ?
YES
NO
INSERT INTERMEDIATE LOCATION POINTERS IN PROGRAM TABLE AT THIS POINT
REVERSE ORDER OF LOCATION POINTERS IN SCRATCH TABLE SO THAT TRAJECTORY IN RETURN DIRECTION CAN ALSO BE MODIFIED
MERGE THE MODIFIED TABLE WITH THE TEMPORARY TABLE
RETURN

```
C --- SUBROUTINE : AVOID
C
C --- ARGUMENTS :
C
C --- CALLED BY : PROGS
C
C --- CALLS SUBROUTINES : NEXT  LOCPT  ADJUST
C
C --- INPUT DATA : PROG(105,6)  = THE PRESENT PROGRAM MODULE.
C                  CS(50)       = THE CHARACTER STRING THAT CONTAINS
C                                   THE PROPER SEQUENCE OF LOCATIONS TO
C                                   BE USED IN THE AVOIDANCE PATH.
C                  BN           = THE LOCATION TABLE POINTER FOR
C                                   EACH LOCATION SPECIFIED IN THE
C                                   CHARACTER STRING CS(50) [FROM 'LOCPT'].
C --- OUTPUT DATA : PROG(105,6)    = THE MODIFIED PROGRAM MODULE WITH
C                                   ALL OF THE ADDITIONAL LINES REQUIRED
C                                   TO CREATE THE AVOIDANCE PATHS THROUGH
C                                   THE SPECIFIED LOCATIONS.
C
C --- FUNCTION: IF A TRAJECTORY FROM LOCATION 'A' TO LOCATION'B'
C               IS TO PASS THROUGH ADDITIONAL INTERMEDIATE LOCATIONS,
C               THIS SUBROUTINE INSERTS THOSE ADDITIONAL LOCATIONS IN
C               THE CORRECT SEQUENCE FOR EVERY OCCURRENCE OF 'A-B' OR
C               'B-A' IN THE PROGRAM MODULE BY MEANS OF A SINGLE
C               'AVOID' COMMAND.
C
      SUBROUTINE AVOID
      IMPLICIT INTEGER(B-Z)
      COMMON/PMOD/PROG(105,6),ENDP,BRNCH(6),M
      COMMON/PTN/BN,EN
      COMMON/CMD/CS(50),PN,PLN
      COMMON/TEMPOR/TEMP(100,6),DET(10),NDET(10)
C
C   A FLAG ('REV') IS SET EQUAL TO ZERO.  IT WILL BE USED TO
C     INDICATE IF THE PROGRAM MODULE HAS BEEN CORRECTED
C     FOR THE AVOIDANCE PATH IN THE REVERSE SEQUENCE ALSO.
C
      REV=0
C
C  THE FOLLOWING LOOP DETERMINES THE NUMBER OF LOCATION POINTS IN THE
C       AVOIDANCE PATH (AS SPECIFIED IN THE CHARACTER STRING CS(50))
C       AND DECODES THEM INTO THEIR PROPER LOCATION TABLE POINTERS
C       BY CALLING 'LOCPT'.
C
      DO 30 JK=1,10
      CALL NEXT
      IF(PN.GT.50)GOTO 20
      CALL LOCPT(1)
30    DET(JK)=BN
      GOTO 77
20    JK=JK-1
C
C   THE FOLLOWING LOOP SEQUENTIALLY TESTS EVERY LINE IN THE PROGRAM
C       MODULE FOR THE PRESENCE OF BOTH OF THE END POINTS OF
C       THE AVOIDANCE PATH.  IF BOTH IN THE SAME LINE, THEN BRANCH
```

```
C       TO CODE TO INSERT ADDITIONAL LINES.
C
77      DO 25 JB=1,100
        IF(PROG(JB,1).EQ.DET(1))GOTO 22
        GOTO 25
22      IF(PROG(JB,2).EQ.DET(JK))GOTO 50
25      CONTINUE
        GOTO 35
C
C   THE FOLLOWING LOOP STORES THE PROGRAM MODULE IN A TEMPORARY
C       TABLE WHICH ACTS AS A SCRATCH PAD UNTIL ALL OF THE REQUIRED
C       MODIFICATIONS HAVE BEEN MADE.
C
50      DO 75 K=1,100
        DO 75 F=1,6
75      TEMP(K,F)=PROG(K,F)
C
C   THIS LOOP INSERTS THE ADDITIONAL LINES FOR THE AVOIDANCE PATH,
C       ZEROS THE FUNCTION FLAGS FOR THESE INTERMEDIATE LOCATIONS AND
C       SETS THE VELOCITY (PROG(NV,3)) EQUAL TO 50 CM/SEC.
C
        VE=PROG(JB,3)
        DO 76 FT=2,JK
        NV=JB+FT-2
        DO 79 RD=3,6
        PROG(NV,RD)=0
79      PROG(NV,3)=VE
76      PROG(NV,2)=DET(FT)
C
C   THE REMAINDER OF THE PROGRAM IS NOW ADDED ON TO THE NEWLY INSERTED
C       LINES.
C
        END=NV-JB
        PT=101-NV
        DO 450 MB=1,PT
        FD=101-MB
        DIF=FD-END
        DO 450 MT=1,6
450     PROG(FD,MT)=TEMP(DIF,MT)
C
C   THE SUBROUTINE 'ADJUST' IS CALLED TO INSURE THAT
C       ALL OF THE LOCATIONS ARE IN THE PROPER SEQUENCE.
C
        CALL ADJUST
C
C   RETURN TO THE LOOP TO CONTINUE TESTING THE PROGRAM MODULE
C       FOR ANY OTHER PLACES WHERE THE AVOIDANCE PATH IS TO BE USED.
C
        GOTO 77
C
C   IF THE REVERSE FLAG ('REV') HAS NOT BEEN SET EQUAL TO ONE, THEN
C       SET IT EQUAL TO ONE AND REVERSE THE SEQUENCE OF THE LOCATION
C       POINTS IN THE AVOIDANCE PATH (DET(10)), AND GO BACK AND
C       TEST THE PROGRAM MODULE FOR ANY PLACES WHERE THE REVERSE
C       PATH IS TO BE USED.  AFTER ALL OF THESE MODIFICATIONS
C       HAVE BEEN DONE, THE REVERSE FLAG WILL BE TESTED AGAIN,
```

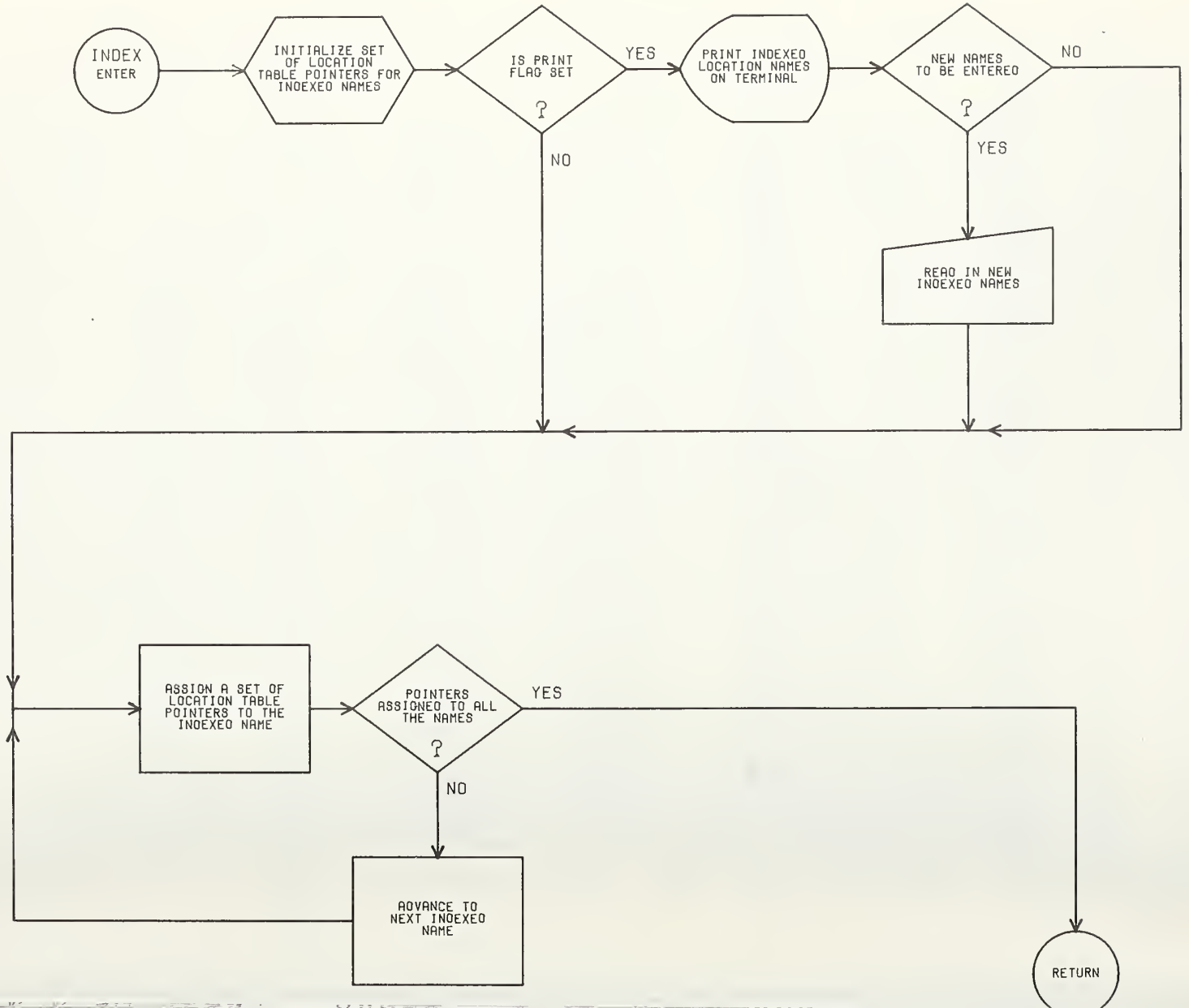
```
C       FOUND EQUAL TO ONE AND CONTROL WILL RETURN TO THE CALLING
C       PROGRAM.
C
35      IF(REV.EQ.1)GOTO 40
        REV=1
        DO 60 LL=1,JK
60      NDET(LL)=DET(LL)
        DO 61 LL=1,JK
        OL=JK+1-LL
61      DET(LL)=NDET(OL)
        GOTO 77
40      RETURN
        END
```


LOOP
ENTER
INITIALIZE TABLE STR TO ALL ZEROS. THESE ARE THE INDEXING VALUES THAT INCREMENT EACH REPEAT OF LOOP
READ IN COMMAND FROM TERMINAL
"FINISH" COMMAND
?
YES
NO
GENERATE ADDITIONAL PROGRAM TABLE LINES USING VALUES IN STR TO INCREMENT OR DECREMENT THE INDEXED VALUES FOR EACH REPEAT OF LOOP
"GO TO" COMMAND
?
YES
NO
PRINT ERROR MESSAGE ON TERMINAL
CALL LINE
TO DECODE ELEMENTAL MOVE STATEMENT INTO PROGRAM TABLE LINE
GROUP OF INDEXED VALUES SPECIFIED
?
YES
NO
SET A +1 IF INDICIES ARE ASCENDING IN VALUE. -1 IF DECENDING. STORE IN STR
RETURN

```
C --- SUBROUTINE : LOOP
C
C --- ARGUMENTS :
C
C --- CALLED BY : INSERT
C
C --- CALLS SUBROUTINES : LINE
C
C --- INPUT DATA : CS(50)        = THE CHARACTER STRING THAT CONTAINS
C                                  AN ELEMENTAL MOVE ('GOTO') COMMAND.
C                   PLN           = THE NUMBER OF THE PRESENT LINE IN THE
C                                  PROGRAM MODULE.
C                   BN            = LOCATION TABLE POINTER FOR AN ENTERED
C                                  LOCATION (IF INDEXED NAME THIS IS
C                                  THE FIRST POINTER OF THE INDEXED
C                                  SEQUENCE [FROM LINE].
C                   EN            = THE LOCATION TABLE POINTER FOR THE LAST
C                                  LOCATION SPECIFIED BY THE INDEXED
C                                  SEQUENCE [FROM LINE].
C                   PROG(105,6)   = THE PROGRAM MODULE.
C
C --- OUTPUT DATA : PROG(105,6)    = THE MODIFIED PROGRAM MODULE
C                                    WITH THE ADDITIONAL LINES GENERATED
C                                    BY THE 'LOOP' COMMAND.
C                    PLN           = THE NUMBER OF THE LAST LINE ENTERED
C                                    IN THE PROGRAM MODULE BY THE 'LOOP'
C                                    COMMAND.
C
C --- FUNCTION: TO ALLOW ENTRY OF A REPEATED SEQUENCE OF ELEMENTAL
C               MOVES WHERE ONLY THE LOCATIONS ARE
C               DIFFERENT AND ARE DESIGNATED BY INDEXED NAMES.
C
        SUBROUTINE LOOP
        IMPLICIT INTEGER (B-Z)
        COMMON/PMOD/PROG(105,6),ENDP,BRNCH(6),M
        COMMON/PTN/BN,EN
        COMMON/CMD/CS(50),PN,PLN
        DIMENSION STR(50)
        K=1
C
C   ZERO THE ARRAY STR(50)
C
        DO 55 H=1,50
55      STR(H)=0
C
C   STORE THE FIRST PROGRAM LINE NUMBER OF THE 'LOOP' SEQUENCE
C       IN THE VARIABLE 'BEGIN'.
C
        BEGIN=PLN
C
C   READ IN A COMMAND FROM THE TERMINAL.
C
99      READ(6,90)(CS(J),J=1,50)
90      FORMAT(50(A1))
        PN=1
C
```

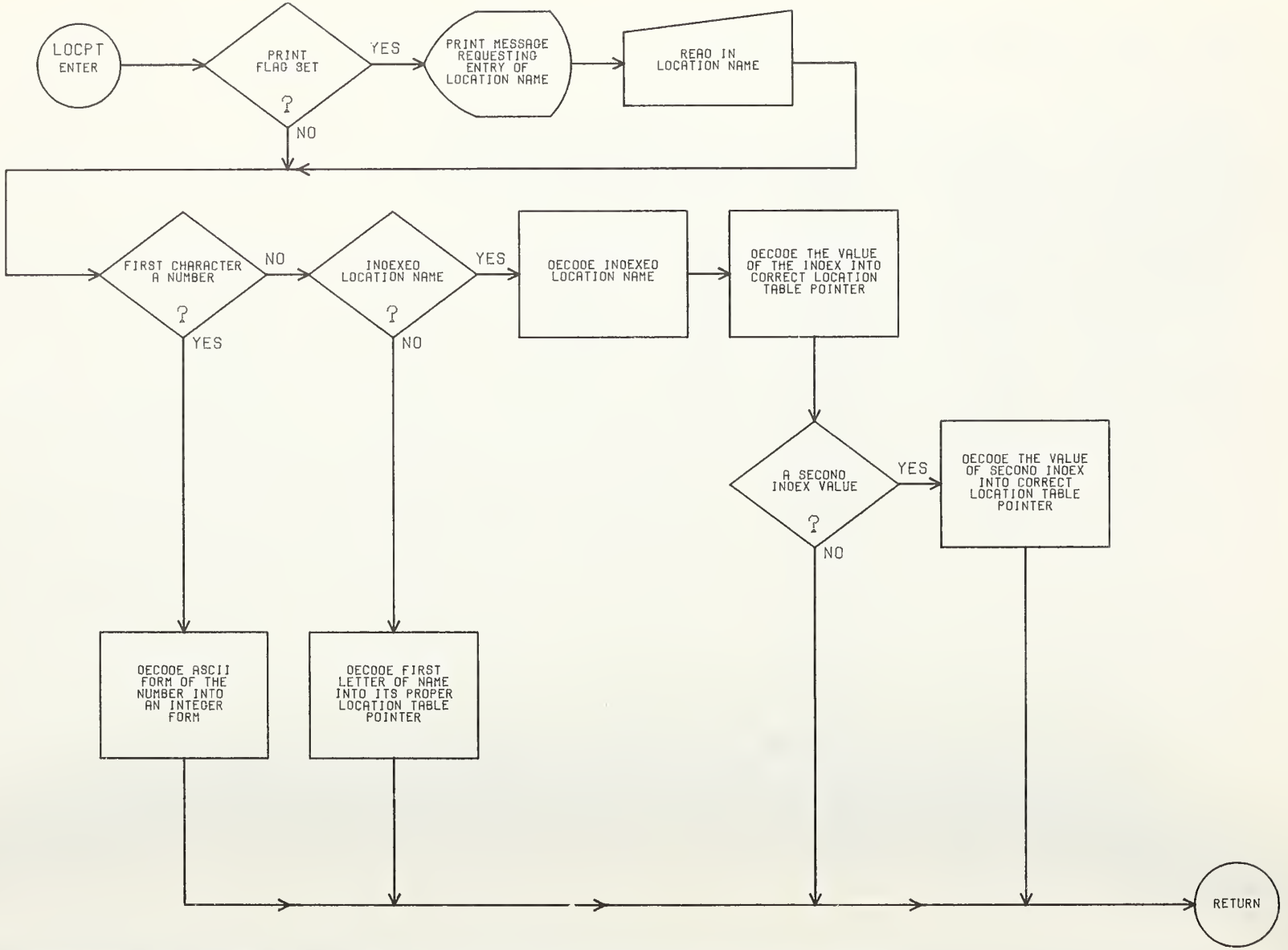
```
C    IF THE COMMAND IS A 'FINISH' STATEMENT THEN BRANCH TO
C        STATEMENT 20 TO GENERATE THE ADDITIONAL REPEATED LINES.
C
         IF(CS(PN).EQ.8262)GOTO 20
C
C    IF THE COMMAND IS A 'GOTO' STATEMENT, THEN BRANCH TO STATEMENT
C        30 TO DECODE IT.
C
         IF(CS(PN).EQ.8263)GOTO 30
C
C    IF THE COMMAND IS NEITHER OF THE ABOVE, THEN PRINT OUT THE
C        FOLLOWING ERROR MESSAGE ON THE TERMINAL AND RETURN TO THE
C        CALLING PROGRAM.
C
100      WRITE(6,101)
101      FORMAT(8X'ERROR IN ***LOOP*** INSTRUCTION')
         PLN=BEGIN
         GOTO 40
C
C    CALL THE SUBROUTINE 'LINE' TO DECODE THE 'GOTO' STATEMENT INTO
C        THE CORRECT LOCATION TABLE POINTERS AND FUNCTION FLAGS
C        AND STORE THEM IN THE PROPER LINE IN THE PROGRAM
C        MODULE.
C
30       CALL LINE
C
C    IF A RANGE OF INDEXED LOCATIONS WAS NOT GIVEN FOR THIS 'GOTO'
C        STATEMENT (IE. ONLY ONE LOCATION WAS SPECIFIED, RESULTING
C        IN 'BN' BEING EQUAL TO 'EN') THEN BRANCH TO STATEMENT
C        31 WHICH WILL  HAVE THE EFFECT OF SETTING THE CORRESPONDING
C        VALUE OF THIS LINE IN 'STR(50)' EQUAL TO ZERO.
C
         IF(EN.EQ.BN)GOTO 31
C
C    IF A RANGE OF INDEXED LOCATIONS WAS GIVEN, THEN THE LOCATION
C        TABLE POINTERS FOR THE STARTING (BN) AND ENDING (EN) INDEX
C        SPECIFIED ARE USED TO CALCULATE THE NUMBER OF REPEATS THROUGH
C        THE LOOP  'SDIF=EN-BN'.
C
         SDIF=EN-BN
         DIF=IABS(SDIF)
C
C     THE SIGN OF THE DIRECTION THROUGH THE INDEXED NAME (IE. IS THE INDEX
C        INCREASING OR DECREASING) IS STORED AT THE APPROPRIATE POSITION
C        IN THE MATRIX 'STR(50'.
C
         STR(K) =SDIF/DIF
C
C    THE PROGRAM MODULE LINE INDICATOR (PLN) IS INCREMENTED.  THEN
C        BRANCH TO STATEMENT 99 TO READ IN ANOTHER COMMAND.
C
31       K=K+1
         PLN=PLN+1
         GOTO 99
C
C    THIS SECTION OF CODE GENERATES THE ADDITIONAL REPEATED LINES,
```

```
C       INCREMENTING (OR DECREMENTING) THE INDEXED LOCATIONS FOR
C       EACH REPEAT.
C
C   CALCULATE THE NUMBER OF LINES TO BE REPEATED AND STORE IN 'LINES'.
C
20      LINES=PLN-BEGIN
C
C   REPEAT THE NUMBER OF LINES('LINES') IN THE REPEATING GROUP 'DIF'
C       TIMES, INCREMENTING THE INDEXED VALUES BY THE APPROPRIATE AMOUNT
C       (STR(KM)*LB) EACH TIME.
C
        DO 21 LB=1,DIF
        DO 21 KM=1,LINES
        ORIG=BEGIN+KM-1
        NXT=BEGIN+LINES*LB+KM-1
        DO 22 JK=1,6
22      PROG(NXT,JK)=PROG(ORIG,JK)
21      PROG(NXT,2)=PROG(NXT,2)+STR(KM)*LB
C
C   SET THE PROGRAM LINE INDICATOR ('PLN') TO THE LAST LINE ENTERED IN
C       THE 'LOOP' SEQUENCE AND RETURN TO THE CALLING PROGRAM.
C
        PLN=NXT
40      RETURN
        END
```


INDEX
ENTER
INITIALIZE SET OF LOCATION TABLE POINTERS FOR INDEXED NAMES
IS PRINT FLAG SET
?
YES
NO
PRINT INDEXED LOCATION NAMES ON TERMINAL
NEW NAMES TO BE ENTERED
?
NO
YES
READ IN NEW INDEXED NAMES
ASSION A SET OF LOCATION TABLE POINTERS TO THE INDEXED NAME
POINTERS ASSIGNED TO ALL THE NAMES
?
YES
NO
ADVANCE TO NEXT INDEXED NAME
RETURN

```
C --- SUBROUTINE : INDEX
C
C --- ARGUMENTS : I         = IF I=1  DO NOT PRINT OUT INDEXED NAMES ON
C                                     TERMINAL AND DO NOT ALLOW THE ENTRY
C                                     OF NEW INDEXED NAMES.
C                             IF I=0  PRINT OUT INDEXED NAMES ON THE
C                                     TERMINAL AND ALLOW THE ENTRY OF
C                                     NEW INDEXED NAMES.
C
C --- CALLED BY : PROGS  LOCTAB
C
C --- CALLS SUBROUTINES :
C
C --- INPUT DATA : IN(30)        = THE CHARACTER STRING THAT CONTAINS THE
C                                  ASCII FORMATTED INDEXED LOCATION NAMES.
C
C --- OUTPUT DATA : NPT(3,20)      = THE ARRAY THAT CONTAINS THE LOCATION
C                                    TABLE POINTERS FOR ALL OF THE POSSIBLE
C                                    INDEXED NAME LOCATIONS.
C                  FL(6)           = THE POSITION IN THE STRING IN(30)
C                                    OF THE FIRST AND LAST LETTER OF EACH
C                                    INDEXED NAME.
C                  NAME(3)         = THE FIRST LETTER OF EACH INDEXED NAME.
C
C --- FUNCTION: INITIALIZES THE ARRAY NPT(3,20) WHICH IS USED TO DECODE
C              AN INDEXED NAME AND ITS INDEX INTO ITS CORRECT LOCATION
C              TABLE POINTER.  STORES THE INDEXED NAMES IN THE STRING
C              IN(30) AS WELL AS DECODING PERTINENT INFORMATION INTO
C              VARIABLES IN A COMMON BLOCK FOR USE BY OTHER PROGRAMS
C              FOR IDENTIFICATION PURPOSES.
C
C
      SUBROUTINE INDEX(I)
      IMPLICIT INTEGER(B-Z)
      COMMON/IND/IN(30),FL(6),NAME(3),NPT(3,20)
C
C   INITIALIZE THE ARRAY NPT(3,20) SO THAT IT CONTAINS ALL OF THE
C      CORRECT LOCATION TABLE POINTERS FOR ALL POSSIBLE
C      INDEXED NAMES.
C
      DO 10 G=1,20
      NPT(1,G)=G
      NPT(2,G)=G+20
10    NPT(3,G)=G+40
C
C   TEST FLAG (I), IF I=0 THEN PRINT OUT THE INDEXED NAMES FROM IN(30),
C      AND REQUEST IF NEW INDEXED NAMES ARE TO BE ENTERED.
C
      IF(I.EQ.1)GOTO 120
300   WRITE(6,199)
199   FORMAT(8X'THESE ARE THE CURRENT INDEXED LOCATIONS')
      WRITE(6,200)(IN(J),J=1,30)
200   FORMAT(2X,30(A1))
      WRITE(6,201)
201   FORMAT(8X'DO YOU WANT TO ENTER NEW INDEXED NAMES ?')
      READ(6,202)JA
```

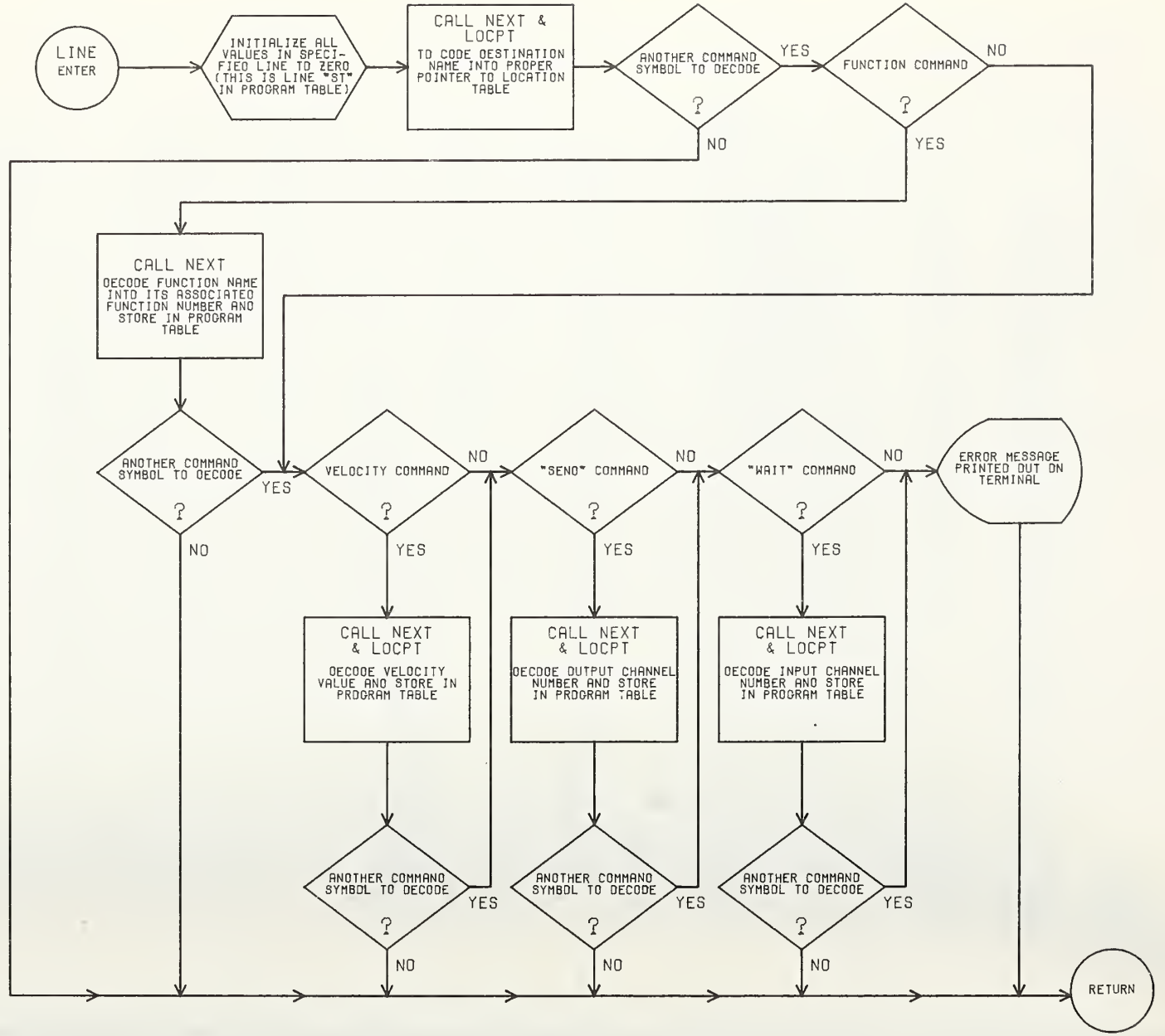
```
202     FORMAT(A1)
        IF(JA.EQ.1HY)GOTO 70
        IF(JA.EQ.1HN)GOTO 120
        WRITE(6,125)
125     FORMAT(8X'YOU DID NOT ANSWER THE QUESTION, PLEASE TRY AGAIN',//)
        GOTO 300
C
C   IF NEW INDEXED NAMES ARE TO BE ENTERED, READ THEM INTO IN(30).
C
70      WRITE(6,100)
100     FORMAT(8X'ENTER INDEXED NAMES',/
        1 8X'EG.    PALLET CUTTER STACK')
        READ(6,102)(IN(J),J=1,30)
102     FORMAT(30(A1))
120     J=0
        K=0
C
C   ADVANCE THROUGH THE STRING IN(30), NOTING THE POSITION IN THE
C       STRING WHERE EACH INDEXED NAME BEGINS AND ENDS AND STORING
C       THESE POSITION VALUES IN FL(6).  ALSO, STORE THE
C       FIRST LETTER OF EACH INDEXED NAME IN NAME(3).
C
20      J=J+1
        IF(J.GT.30)GOTO 40
        IF(IN(J).EQ.8224)GOTO 20
        K=K+1
        FL(K)=J
        D=(K+1)/2
        NAME(D)=IN(J)
15      J=J+1
        IF(J.GT.30)GOTO 40
        IF(IN(J).NE.8224)GOTO 15
        K=K+1
        FL(K)=J-1
        GOTO 20
40      RETURN
        END
```


LOCPT
ENTER
PRINT
FLAG SET
?
YES
NO
PRINT MESSAGE
REQUESTING
ENTRY OF
LOCATION NAME
READ IN
LOCATION NAME
FIRST CHARACTER
A NUMBER
?
NO
YES
INDEXED
LOCATION NAME
?
YES
NO
DECODE INDEXED
LOCATION NAME
DECODE THE VALUE
OF THE INDEX INTO
CORRECT LOCATION
TABLE POINTER
A SECOND
INDEX VALUE
?
YES
NO
DECODE THE VALUE
OF SECOND INDEX
INTO CORRECT
LOCATION TABLE
POINTER
DECODE ASCII
FORM OF THE
NUMBER INTO
AN INTEGER
FORM
DECODE FIRST
LETTER OF NAME
INTO ITS PROPER
LOCATION TABLE
POINTER
RETURN

```
C --- SUBROUTINE : LOCPT
C
C --- ARGUMENTS : D    = FLAG USED TO INDICATE IF A LOCATION NAME
C                       IS TO BE ENTERED FROM THE TERMINAL (D=0),
C                       OR IF THE LOCATION NAME IS COMING FROM
C                       THE CALLING PROGRAM (D=1)>
C
C --- CALLED BY : AVOID  LIMIT  LINE  NWTAB
C
C --- CALLS SUBROUTINES :
C
C --- INPUT DATA : CS(50)      = THE CHARACTER STRING THAT CONTAINS THE
C                                 ASCII FORMAT OF THE LOCATION NAME.
C                  PN          = THE POSITION IN THE ABOVE CHARACTER
C                                 STRING OF THE FIRST LETTER OF THE
C                                 LOCATION NAME.
C                  IN(30)      = THE CHARACTER STRING THAT CONTAINS THE
C                                 ENGLISH NAMES (IN ASCII FORMAT) OF THE
C                                 INDEXED LOCATION POINTS.
C                  NAME(3)     = THE FIRST LETTER OF THE INDEXED
C                                 NAMES.
C                  NPT(3,20)   = THE MATRIX THAT CONTAINS THE LOCATION
C                                 TABLE POINTERS FOR ALL OF THE
C                                 INDEXED POSITIONS.
C
C --- OUTPUT DATA : BN           = THE LOCATION TABLE POINTER DECODED FROM
C                                 THE ASCII FORMAT ENGLISH NAME OF THE
C                                 LOCATION POINT.
C                   EN           = THE LOCATION TABLE POINTER FOR THE
C                                 LAST INDEXED POSITION IF A SEQUENCE OF
C                                 INDEXED LOCATIONS HAS BEEN ENTERED.
C
C --- FUNCTION: TO DECODE THE ENGLISH NAME OF A LOCATION INTO ITS
C               CORRECT LOCATION TABLE POINTER.  IF A SEQUENCE OF
C               INDEXED LOCATIONS IS TO BE DECODED, THIS ROUTINE
C               RETURNS THE LOCATION TABLE POINTERS OF
C               THE FIRST AND LAST LOCATIONS IN THE SEQUENCE.  THIS
C               ROUTINE CAN ALSO BE USED TO DECODE AN ASCII FORMAT
C               NUMBER FROM 1-100 INTO ITS INTEGER VALUES.
C
      SUBROUTINE LOCPT(D)
      IMPLICIT INTEGER (B-Z)
      COMMON/IND/IN(30),FL(6),NAME(3),NPT(3,20)
      COMMON/PTN/BN,EN
      COMMON/CMD/CS(50),PN,PLN
      BN=0
      EN=0
C
C   TEST FLAG TO SEE IF LOCATION NAME TO BE ENTERED THROUGH
C       TERMINAL.
      IF(D.EQ.1)GOTO 75
C
C   ACCEPT LOCATION NAME FROM TERMINAL.
C
500   WRITE(6,300)
300   FORMAT(2X'ENTER TRAJECTORY NAME')
```

```
        READ(6,200)(CS(J),J=1,50)
200     FORMAT(50(A1))
        J=0
10      J=J+1
        IF(CS(J).EQ.8224)GOTO 10
        PN=J
75      J=PN
C
C   TEST TO SEE IF FIRST CHARACTER IS A LETTER OR A NUMBER.  IF A
C       LETTER BRANCH TO TEST IF INDEXED NAME.
C
        ONE=CS(J)
        IF(ONE.GT.8256)GOTO 50
C
C   DECODE ASCII FORMAT NUMBERS INTO TWO DIGIT INTEGERS.
C
        TWO=CS(J+1)
        BN=(10*(ONE-8240)+TWO-8240)+90
        J=J+2
        GOTO 41
50      J=J+1
        IF(J.GT.50)GOTO 60
        IF(CS(J).NE.8224)GOTO 35
37      J=J+1
C
C   IF LETTERED NAME DOES NOT HAVE ANY INDEXING NUMBERS, BRANCH TO
C       STATEMENT 60.
C
        IF(J.GT.50)GOTO 60
        IF(CS(J).EQ.8224)GOTO 37
        IF(CS(J).NE.8232)GOTO 60
35      IF(CS(J).NE.8232)GOTO 50
C
C   IF A OPEN PARENS IS DETECTED (INDICATING THE INDEXING NUMBERS),
C       THEN DECODE THE INDEXED NAME USED.
C
        DO 20 K=1,3
20      IF(ONE.EQ.NAME(K))GOTO 7
100     WRITE(6,101)
101     FORMAT(2X'INDEXED NAME ERROR')
        GOTO 40
7       J=J+1
        IF(J.LT.51)GOTO 33
102     WRITE(6,103)
103     FORMAT(2X'INCORRECT NAME')
        GOTO 40
C
C   DECODE THE INDEXING NUMBER USED WITH THE INDEXED NAME.
C
33      IF(CS(J).EQ.8224)GOTO 7
        ONE=CS(J)
        TWO=CS(J+1)
        F=(ONE-8240)*10+TWO-8240
        BN=NPT(K,F)
        J=J+2
80      IF(CS(J).EQ.8237)GOTO 82
```

```
        IF(CS(J).NE.8233)GOTO 100
        EN=BN
        GOTO 41
C
C   IF A SEQUENCE OF INDEXING NUMBERS FOR A PARTICULAR INDEXED
C       NAME IS GIVEN, DECODE THE LAST INDEXING NUMBER IN THE
C       SEQUENCE.
C
82      J=J+3
        ONE=CS(J-2)
        TWO=CS(J-1)
        F=(ONE-8240)*10+TWO-8240
C
C   LOOK UP THE LOCATION TABLE POINTER FOR THE
C       LAST INDEXING NUMBER FOR THE SPECIFIED INDEXED NAME.  EG. FOR
C       THE INDEXED NAME AND SEQUENCE NUMBERS -  BOX(13-17) - THE
C       INDEXED NAME IS 'BOX', THE FIRST INDEXING NUMBER IS '13', THE
C       LAST INDEXING NUMBER IS '17'.  IF 'BOX' WAS THE SECOND
C       INDEXED NAME STORED IN 'IN(30)' THEN THE FIRST LOCATION TABLE
C       POINTER WOULD BE FOUND AT 'NPT(2,13)', AND THE LAST LOCATION TABLE
C       POINTER WOULD BE FOUND AT 'NPT(2,17)'.
C
        EN=NPT(K,F)
        GOTO 41
C
C   DECODE THE NON-INDEXED LETTERED NAME INTO THE CORRECT LOCATION
C       TABLE POINTER.
C
60      BN=ONE-8196
        EN=BN
        J=J-1
41      PN=J
        IF(BN.LE.0)GOTO 102
        IF(BN.GT.190)GOTO 102
40      RETURN
        END
```

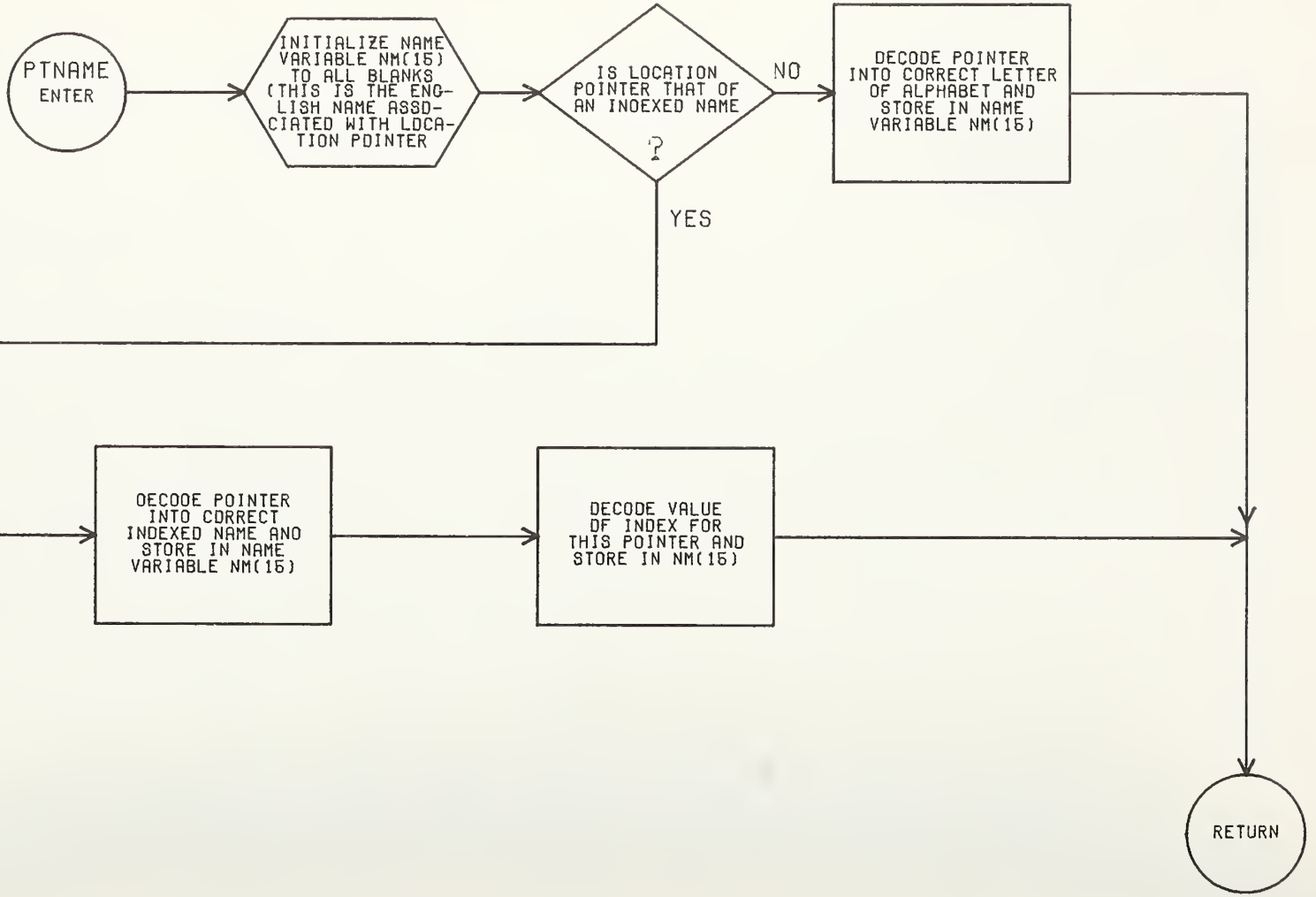

LINE ENTER
INITIALIZE ALL VALUES IN SPECI-FIED LINE TO ZERO (THIS IS LINE "ST" IN PROGRAM TABLE)
CALL NEXT & LOCPT
TO CODE DESTINATION NAME INTO PROPER POINTER TO LOCATION TABLE
ANOTHER COMMAND SYMBOL TO DECODE ?
YES
FUNCTION COMMAND ?
NO
YES
NO
CALL NEXT
DECODE FUNCTION NAME INTO ITS ASSOCIATED FUNCTION NUMBER AND STORE IN PROGRAM TABLE
ANOTHER COMMAND SYMBOL TO DECODE ?
YES
NO
VELOCITY COMMAND ?
NO
YES
"SEND" COMMAND ?
NO
YES
"WAIT" COMMAND ?
NO
YES
ERROR MESSAGE PRINTED OUT ON TERMINAL
CALL NEXT & LOCPT
DECODE VELOCITY VALUE AND STORE IN PROGRAM TABLE
CALL NEXT & LOCPT
DECODE OUTPUT CHANNEL NUMBER AND STORE IN PROGRAM TABLE
CALL NEXT & LOCPT
DECODE INPUT CHANNEL NUMBER AND STORE IN PROGRAM TABLE
ANOTHER COMMAND SYMBOL TO DECODE ?
YES
NO
ANOTHER COMMAND SYMBOL TO DECODE ?
YES
NO
ANOTHER COMMAND SYMBOL TO DECODE ?
YES
NO
RETURN

```
C --- SUBROUTINE : LINE
C
C --- ARGUMENTS :
C
C --- CALLED BY : INSERT  LOOP
C
C --- CALLS SUBROUTINES : NEXT  LOCPT
C
C --- INPUT DATA : CS(50)     = THE CHARACTER STRING THAT CONTAINS THE
C                                ELEMENTAL MOVE ('GOTO' STATEMENT) COMMAND.
C                  PLN        = THE NUMBER OF THE LINE IN THE PROGRAM
C                                MODULE THAT THE ABOVE STATEMENT WILL BE
C                                CODED INTO.
C                  BN         = THE INTEGER REPRESENTATION OF AN ASCII
C                                CODED NUMBER IN THE CHARACTER STRING
C                                CS(50), USED TO GENERATE LOCATION
C                                TABLE POINTERS, VELOCITY VALUES, AND
C                                CHANNELS FOR INTERLOCK (SEND AND WAIT)
C                                SIGNALS [FROM 'LOCPT'].
C --- OUTPUT DATA : PROG(PLN,6) = THE NEW LINE OF THE PROGRAM MODULE
C                                 THAT THE ELEMENTAL MOVE HAS BEEN
C                                 CODED INTO.
C
C --- FUNCTION: TAKES THE ENGLISH TEXT ELEMENTAL MOVE  STATEMENT
C               TYPED INTO THE TERMINAL, CODES IT INTO THE CORRECT
C               POINTERS AND FLAGS, AND STORES THEM IN THEIR PROPER
C               POSITIONS IN THE SPECIFIED LINE IN THE
C               PROGRAM MODULE.
C
        SUBROUTINE LINE
        IMPLICIT INTEGER(B-Z)
        COMMON/PMOD/PROG(105,6),ENDP,BRNCH(6),M
        COMMON/PTN/BN,EN
        COMMON/CMD/CS(50),PN,PLN
        COMMON/BED/BEG,FIN,DIF,ERR
C
C   ZERO ALL OF THE VALUES IN LINE PLN, SET THE VELOCITY INDICATOR
C       EQUAL TO 50 CM/SEC AS A DEFAULT VALUE.
C
25      DO 125 J=2,6
125     PROG(PLN,J)=0
        PROG(PLN,3)=50
C
C   ADVANCE THROUGH THE CHARACTER STRING UNTIL THE DESTINATION NAME
C       IS REACHED(BY A CALL TO 'NEXT').
C
        CALL NEXT
        IF(PN.LT.51)GOTO 41
C
C   THIS IS THE ERROR MESSAGE PRINTED IF ANY MISTAKE IN FORMAT
C       IS DETECTED.
C
100     WRITE(6,101)
101     FORMAT(8X'ERROR IN *** GOTO ***STATEMENT')
        GOTO 42
C
```

```
C    THE ASCII REPRESENTATION OF THE DESTINATION IS DECODED
C        INTO THE CORRECT LOCATION TABLE POINTER BY A CALL TO 'LOCPT'.
C        THIS POINTER IS THEN STORED IN THE PROGRAM MODULE.
C
41       CALL LOCPT(1)
         PROG(PLN,2)=BN
         BBN=BN
         EEN=EN
C
C    ADVANCE THROUGH THE ASCII CHARACTER STRING UNTIL A FUNCTION
C        COMMAND IS FOUND (BY A CALL TO 'NEXT').
C
15       CALL NEXT
         IF(PN.GT.50)GOTO 42
C
C    THIS IS A SERIES OF TESTS ON THE ASCII FORM OF THE FIRST LETTER OF THE
C        FUNCTION COMMAND TO DETERMINE IF IT IS A G(GRASP),R(RELEASE),
C        P(PROXIMITY),D(DETECT),B(BALANCE),U(UNSTACK),T(TOUCH),
C        L(LINE),E(EDGE).
C
C
C    'GRASP'
C
         IF(CS(PN).NE.8263)GOTO 60
         PROG(PLN,6)=1
         GOTO 80
C
C    'RELEASE'
C
60       IF(CS(PN).NE.8274)GOTO 61
         PROG(PLN,6)=2
80       PN=PN+1
C
C    'PROXIMITY'
C
         IF(CS(PN).NE.8272)GOTO 81
82       PROG(PLN,6)=PROG(PLN,6)+2
81       CALL NEXT
         IF(PN.GT.50)GOTO 42
C
C    'PROXIMITY'
C
         IF(CS(PN).NE.8272)GOTO 85
         GOTO 82
C
C    'DETECT'
C
61       IF(CS(PN).NE.8260)GOTO 62
         PROG(PLN,6)=5
         GOTO 81
C
C    'BALANCE'
C
62       IF(CS(PN).NE.8258)GOTO 63
         PROG(PLN,6)=6
         GOTO 81
```

```
C
C   'UNSTACK'
C
63      IF(CS(PN).NE.8277)GOTO 64
        PROG(PLN,6)=7
        GOTO 81
C
C   'TOUCH'
C
64      IF(CS(PN).NE.8276)GOTO 65
        PROG(PLN,6)=8
        GOTO 81
C
C   'LINE'
C
65      IF(CS(PN).NE.8268)GOTO 66
        PROG(PLN,6)=9
        GOTO 81
C
C   'EDGE'
C
66      IF(CS(PN).NE.8261)GOTO 85
        PROG(PLN,6)=10
        CALL NEXT
        CALL LIMIT
        PROG(PLN,5)=BEG
        GOTO 81
C
C   TEST FOR A 'VELOCITY' COMMAND, IF THERE IS ONE THEN DECODE THE
C       ASCII FORM OF THE SPECIFIED VELOCITY (CM/SEC) BY A
C       CALL TO 'LOCPT'.  MULTIPLY THIS INTEGER NUMBER BY 10 AND STORE
C       AT POSITION 3 IN THE LINE IN THE PROGRAM MODULE.
C
85      IF(CS(PN).NE.8278)GOTO 90
        CALL NEXT
        IF(PN.GT.50)GOTO 100
        CALL LOCPT(1)
        PROG(PLN,3)=(BN-90)
        CALL NEXT
        IF(PN.GT.50)GOTO 42
C
C   TEST FOR A 'SEND' COMMAND.  IF THERE IS ONE, DECODE
C       THE ASCII FORM OF THE OUTPUT CHANNEL NUMBER THAT IS TO
C       BE SET HIGH, MULTIPLY THIS INTEGER BY 100 AND STORE
C       IN POSITION 4 OF THE SPECIFIED LINE OF THE PROGRAM
C       MODULE.
C
90      IF(CS(PN).NE.8275)GOTO 91
        CALL NEXT
        IF(PN.GT.50)GOTO 100
        CALL LOCPT(1)
        PROG(PLN,4)=(BN-90)*100
        CALL NEXT
        IF(PN.GT.50)GOTO 42
C
C   TEST FOR A 'WAIT' COMMAND.  IF THERE IS ONE, STORE THE NUMBER
```

```
C       OF THE INPUT CHANNEL THAT MUST GO HIGH FOR THE PROGRAM
C       TO CONTINUE.
C
91      IF(CS(PN).NE.8279)GOTO 100
        CALL NEXT
        IF(PN.GT.50)GOTO 100
        CALL LOCPT(1)
        PROG(PLN,4)=PROG(PLN,4)+BN-90
        CALL NEXT
        IF(PN.LT.51)GOTO 100
42      BN=BBN
        EN=EEN
40      RETURN
        END
```


PTNAME
ENTER
INITIALIZE NAME VARIABLE NM(15) TO ALL BLANKS (THIS IS THE ENG-LISH NAME ASSD-CIATED WITH LOCA-TION POINTER
IS LOCATION POINTER THAT OF AN INOEXED NAME
?
NO
YES
DECODE POINTER INTO CORRECT LETTER OF ALPHABET AND STORE IN NAME VARIABLE NM(15)
OECOOE POINTER INTO CORRECT INDEXED NAME ANO STORE IN NAME VARIABLE NM(15)
DECODE VALUE OF INDEX FOR THIS POINTER AND STORE IN NM(15)
RETURN

```
C --- SUBROUTINE : PTNAME
C
C --- ARGUMENTS : NU        = A LOCATION TABLE POINTER SPECIFIED BY
C                             THE CALLING PROGRAM.
C
C --- CALLED BY : PLINE
C
C --- CALLS SUBROUTINES :
C
C --- INPUT DATA : FL(6)         = THE LOCATIONS IN THE CHARACTER STRING
C                                  IN(30) OF THE FIRST AND LAST LETTER
C                                  OF THE INDEXED LOCATION NAMES.
C                  NAME(3)       = THE FIRST LETTERS OF THE INDEXED
C                                  LOCATION NAMES.
C                  NPT(3,20)     = THE ARRAY OF VALUES FOR THE POINTERS
C                                  FOR THE INDEXED LOCATION NAMES.
C
C --- OUTPUT DATA : NM(15)        = THE CHARACTER STRING THAT CONTAINS
C                                  THE SEQUENCE OF LETTERS (IN ASCII
C                                  FORMAT) THAT MAKES UP THE NAME THAT
C                                  CORRESPONDS TO THE LOCATION TABLE
C                                  POINTER 'NU'.
C
C --- FUNCTION: DECODES THE LOCATION TABLE POINTER 'NU' INTO THE
C               CORRESPONDING ASCII FORMATED NAME.
C
      SUBROUTINE PTNAME(NU)
      IMPLICIT INTEGER(B-Z)
      COMMON/NAM/NM(15)
      COMMON/IND/IN(30),FL(6),NAME(3),NPT(3,20)
C
C   FILL THE CHARACTER STRING NM(15) WITH BLANKS.
C
      DO 10 J=1,15
10    NM(J)=8224
C
C   TEST IF POINTER 'NU' IS A SINGLE LETTERED NAME OR
C       AN INDEXED LOCATION NAME.  IF INDEXED NAME THEN BRANCH TO
C       STATEMENT NUMBER 60.
C
      IF(NU.LT.61)GOTO 60
C
C   DECODE THE LETTERED NAME LOCATION TABLE POINTER INTO ITS
C       APPROPRIATE ASCII FORMAT LETTER.  STORE THIS VALUE IN THE STRING
C       NM(30) AND RETURN TO THE CALLING PROGRAM.
C
      NM(1)=NU+8196
      GOTO 40
C
C   SEPARATE OUT THE TEN'S DIGIT AND THE UNIT'S DIGIT FROM
C       THE POINTER 'NU' FOR THE INDEXED NAME.
C
60    DO 31 K=1,7
      TEN=K-1
      UNIT=NU-10*TEN
      IF(UNIT.LT.10)GOTO 80
```

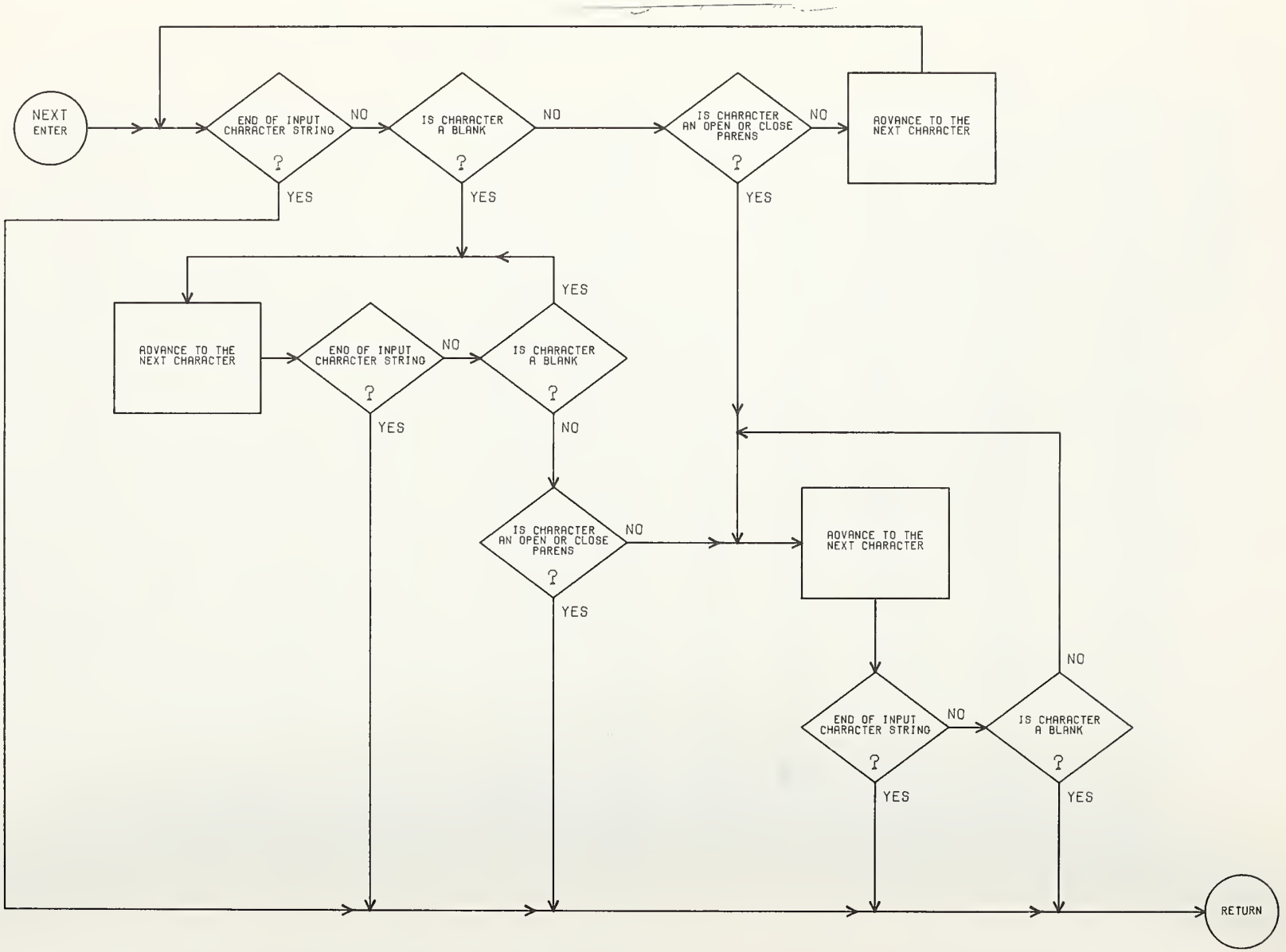
```
31      CONTINUE
C
C   THIS SECTION DECODES THE LOCATIONS IN THE STRING IN(30) OF THE
C       FIRST AND LAST LETTER OF THE INDEXED NAME SPECIFED BY THE
C       POINTER 'NU'.  THEN STORES THE LETTERS OF THIS INDEXED NAME
C       IN THE STRING NM(15).
C
80      J=2*((NU-1)/20)+1
        BE=FL(J)
        EN=FL(J+1)
        DO 82 LJ=1,15
        NN=BE+LJ-1
        IF(NN.GT.EN)GOTO 83
82      NM(LJ)=IN(NN)
C
C   AN OPEN PARENS IS NOW INSERTED IN THE STRING NM(15) AFTER THE
C       INDEXED NAME.
C
83      NM(LJ)=8232
C
C   THE CORRECT INDEX NUMBER IS DECODED AND STORED IN NM(15).
C
        NTEN=((TEN+1)*2)/(J+1)-1
        IF(TEN.NE.(J+1))GOTO 85
        NTEN=2
85      NM(LJ+1)=NTEN+8240
        NM(LJ+2)=UNIT+8240
C
C   A CLOSE PARENS (IE THE ASCII FORMAT VALUE FOR A CLOSE PARENS) IS
C       PLACED AFTER THE INDEX NUMBER IN NM(15) AND CONTROL RETURNS TO
C       THE CALLING PROGRAM.
C
        NM(LJ+3)=8233
40      RETURN
        END
```

PLINE
ENTER
PRINT FIRST
LINE OF PROGRAM
TABLE
?
YES
NO
CALL PTNAME
TO DECODE POINTER
VALUE INTO ITS
CORRESPONDING
LOCATION NAME
PRINT STARTING
LOCATION NAME
ON TERMINAL
CALL PTNAME
TO DECODE
DESTINATION POINTER
FROM SPECIFIED
LINE INTO ITS
CORRESPONDING
LOCATION NAME
DECODE VELOCITY,
INTERLOCK CHANNEL
NUMBERS, AND
FUNCTION FLAGS
FOR THIS SPECIFIED
PROGRAM TABLE LINE
PRINT OUT LINE
IN ITS ELEMENTAL
MOVE FORMAT
RETURN

```
C --- SUBROUTINE : PLINE
C
C --- ARGUMENTS : LN            = THE NUMBER OF THE PARTICULAR LINE
C                                 IN THE PROGRAM MODULE TO BE
C                                 PRINTED OUT.
C
C --- CALLED BY : INSERT  DELETE  PRINT  SAMPLE
C
C --- CALLS SUBROUTINES : PTNAME
C
C --- INPUT DATA : PROG(105,6)  = THE PROGRAM MODULE.
C                  NM(15)       = THE ASCII FORMATTED
C                                 CHARACTER STRING THAT CONTAINS THE
C                                 NAME OF THE LOCATION 'NU' [FROM 'PTNAME']
C
C --- OUTPUT DATA : THE DECODED LINE FROM THE PROGRAM MODULE PRINTED
C                   OUT ON THE TERMINAL.
C
C --- FUNCTION: DECODES A SPECIFIED LINE FROM THE PROGRAM MODULE
C               INTO THE ENGLISH LANGUAGE CHARACTER STRING ORIGINALLY
C               USED TO ENTER IT, AND DISPLAYS IT ON A TERMINAL.
C
        SUBROUTINE PLINE(LN)
        IMPLICIT INTEGER(B-Z)
        COMMON/PMOD/PROG(105,6),ENDP,BRNCH(6),M
        COMMON/NAM/NM(15)
        COMMON/CMD/CS(50),PN,PLN
        DIMENSION FN(6,11)
        DATA FN/6*'   ',
        2 'GR','AS','P ',3*'   ',
        3 'RE','LE','AS','E ',2*'   ',
        4 'GR','AS','P ','PR','OX','   ',
        5 'RE','LE','AS','E ','PR','OX',
        6 'DE','TE','CT',3*'   ',
        7 'BA','LA','NC','E ',2*'   ',
        8 'UN','ST','AC','K ',2*'   ',
        9 'TO','UC','H ',3*'   ',
        1 'LI','NE',4*'   ',
        1 'ED','GE',4*'   '/
C
C   IF THE FIRST LINE OF THE PROGRAM MODULE IS TO BE PRINTED OUT,
C       THEN DECODE THE STARTING LOCATION BY A CALL TO 'PTNAME'
C       WHICH RETURNS THE ASCII FORMATTED NAME IN THE STRING 'NM(15)'.
C       THIS NAME IS THEN PRINTED OUT WITH THE WORD 'START'.
C
        IF(LN.NE.1)GOTO 10
        NU=PROG(1,1)
        IF(NU.EQ.0)GOTO 40
        CALL PTNAME(NU)
        WRITE(6,200)(NM(J),J=1,15)
200     FORMAT(6X,'START',1X,15(A1))
C
C   THE DESTINATION POINTER 'NU' IS DECODED BY 'PTNAME' AND RETURNED
C       IN 'NM(15)'.
C
10      NU=PROG(LN,2)
```

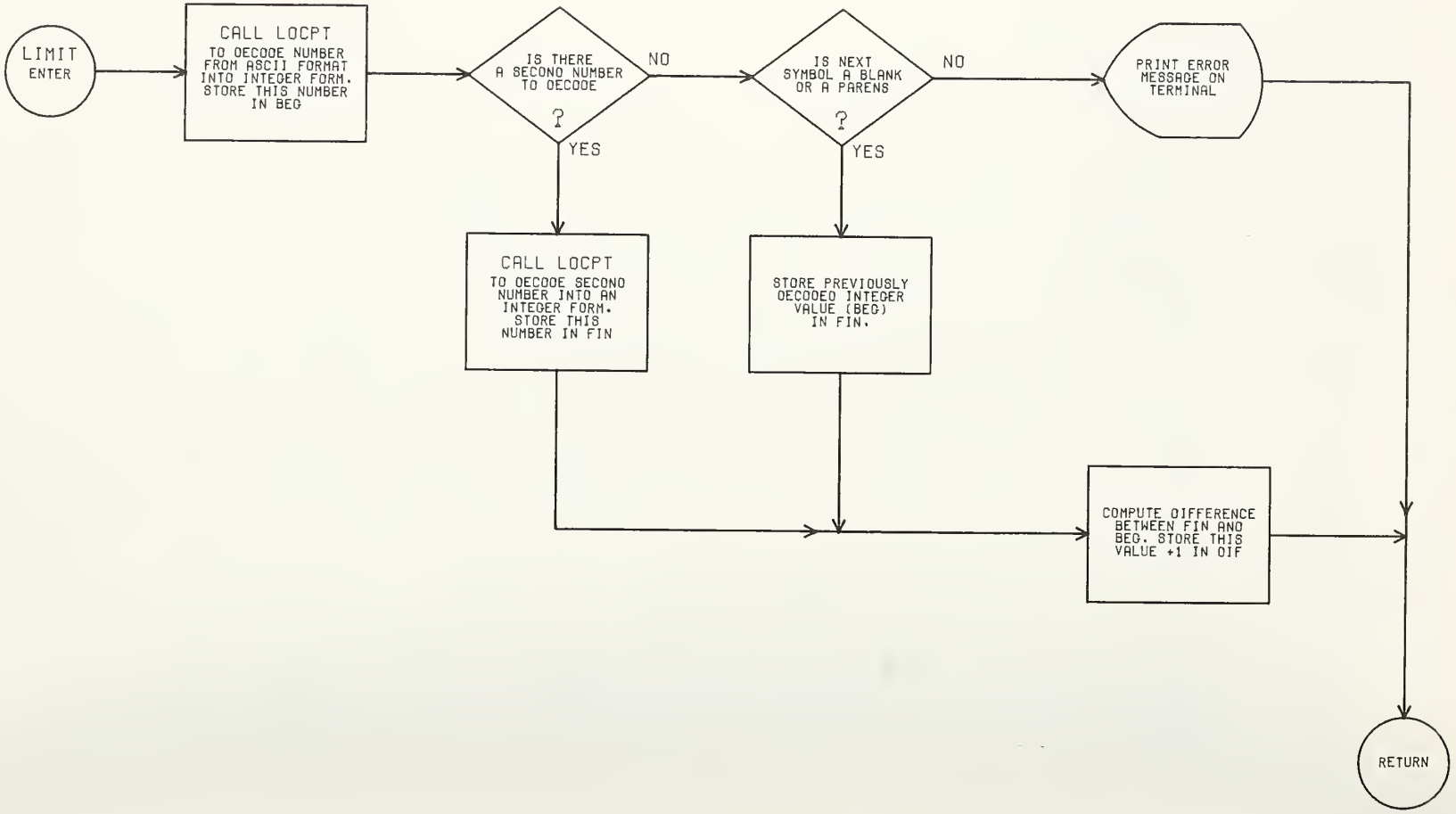
```
        IF(NU.EQ.0)GOTO 40
        CALL PTNAME(NU)
C
C   THE FLAG 'F' IS CALCULATED FROM THE FUNCTION FLAG OF THE
C       OF THE PROGRAM LINE AND IS USED TO SPECIFY THE PROPER LINE
C       OF THE MATRIX 'FN(6,11)' IN ORDER TO PRINT OUT THAT FUNCTION
C       NAME.
C
        F=PROG(LN,6)+1
C
C   IF AN OUTPUT SIGNAL IS TO BE SENT OUT ON A CHANNEL, THEN THE
C       CHANNEL NUMBER IS DECODED AND STORED IN 'SEN'.
C
        SEN=PROG(LN,4)/100
C
C   IF THE PROGRAM IS TO WAIT FOR A SIGNAL ON AN INPUT CHANNEL, THEN
C       THE INPUT CHANNEL NUMBER IS DECODED AND STORED IN 'WA'.
C
        WA=PROG(LN,4)-SEN*100
C
C   THE VELOCITY VALUE IN CM/SEC IS DECODED AND STORED IN 'VEL'.
C
        VEL=PROG(LN,3)
C
C   THE DECODED PROGRAM LINE IS NOW PRINTED OUT ON THE TERMINAL.
C
        IF(PROG(LN,5).EQ.0)GOTO 140
        LE=PROG(LN,5)
        WRITE(6,300)LN,(NM(L),L=1,15),(FN(E,F),E=1,6),LE,VEL,SEN,WA
300     FORMAT('+'I5,' GOTO ',15(A1),1X,6(A2),T34,I2,T41,' VELOCITY(',I3,
        1 ') SEND(',I2,') WAIT(',I2,')')
        GOTO 40
140     WRITE(6,301)LN,(NM(L),L=1,15),(FN(E,F),E=1,6),VEL,SEN,WA
301     FORMAT('+',I5,' GOTO ',15(A1),1X,6(A2),' VELOCITY(',I3,
        1 ') SEND(',I2,') WAIT(',I2,')')
40      RETURN
        END
```

%


NEXT ENTER
END OF INPUT CHARACTER STRING ?
NO
YES
IS CHARACTER A BLANK ?
NO
YES
IS CHARACTER AN OPEN OR CLOSE PARENS ?
NO
YES
ADVANCE TO THE NEXT CHARACTER
ADVANCE TO THE NEXT CHARACTER
END OF INPUT CHARACTER STRING ?
NO
YES
IS CHARACTER A BLANK ?
YES
NO
IS CHARACTER AN OPEN OR CLOSE PARENS ?
NO
YES
ADVANCE TO THE NEXT CHARACTER
END OF INPUT CHARACTER STRING ?
NO
YES
IS CHARACTER A BLANK ?
NO
YES
RETURN

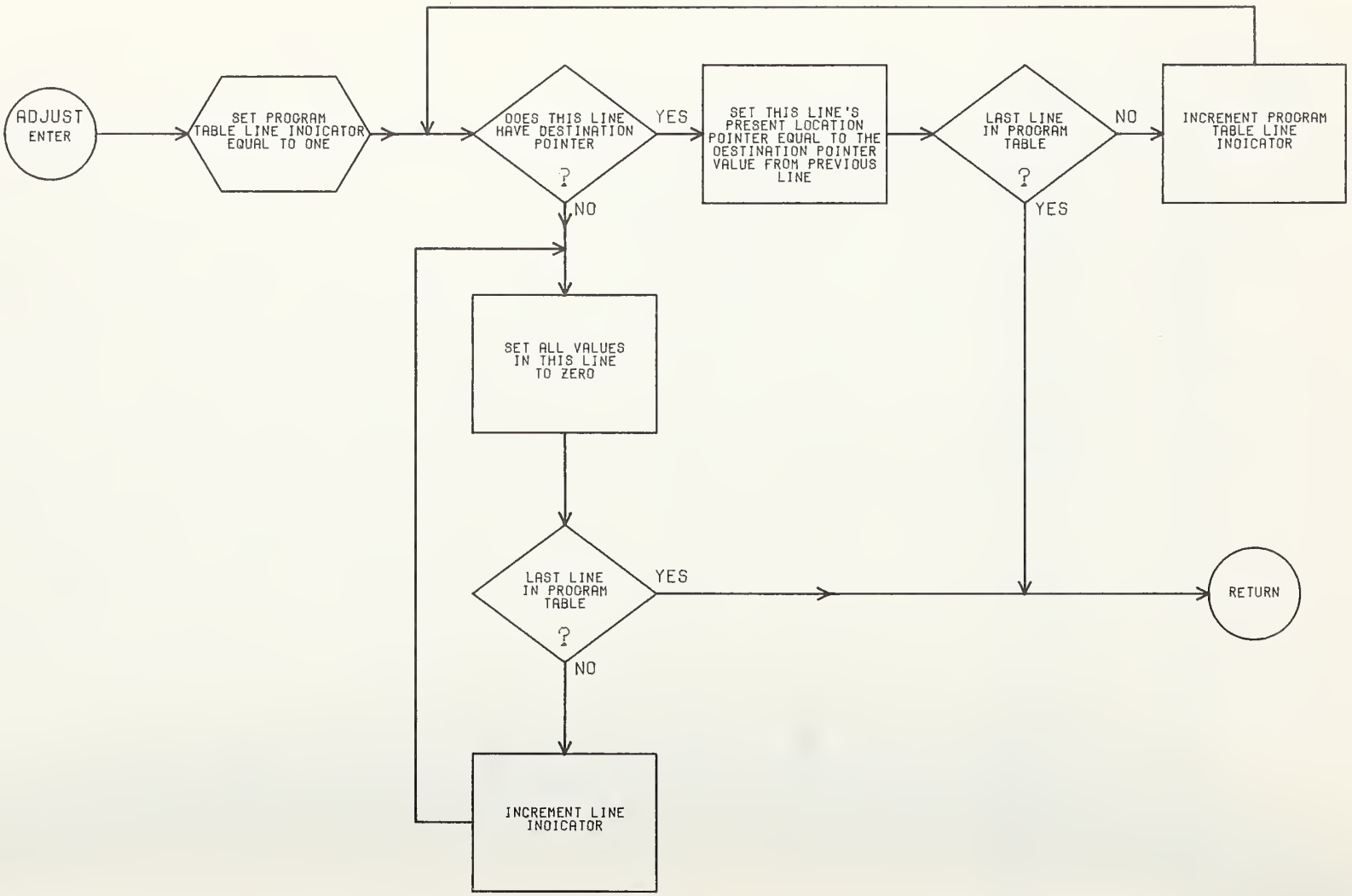
```
C --- SUBROUTINE : NEXT
C
C --- ARGUMENTS :
C
C --- CALLED BY : PRINT  INSERT  DELETE  LINE
C
C --- CALLS SUBROUTINES :
C
C --- INPUT DATA : CS(50)      = THE CHARACTER STRING THAT CONTAINS THE
C                                ASCII CODED INPUT INFORMATION.
C                   PN         = THE INDICATOR DESIGNATING THE PRESENT
C                                POSITION IN THE CHARACTER STRING
C                                CS(50).
C
C --- OUTPUT DATA : PN           =  THE POSITION INDICATOR IN THE STRING
C                                    CS(50) THAT INDICATES THE BEGINNING OF
C                                    THE NEXT PIECE OF INFORMATION.
C
C --- FUNCTION: ADVANCES THE POSITION INDICATOR 'PN' UNTIL REACHES
C               THE BEGINNING OF THE NEXT PIECE OF INFORMATION IN THE
C               CHARACTER STRING CS(50).
C
        SUBROUTINE NEXT
        IMPLICIT INTEGER(B-Z)
        COMMON/CMD/CS(50),PN,PLN
        GOTO 21
48      PN=PN+1
21      IF(PN.GT.50)GOTO 40
C
C   THIS (PN) POSITION IN THE CHARACTER STRING IS CHECKED TO SEE IF
C       IT IS A BLANK, AN OPEN PARENS, OR A CLOSE PARENS.
C
        IF(CS(PN).EQ.8224)GOTO 41
        IF(CS(PN).EQ.8232)GOTO 50
        IF(CS(PN).EQ.8233)GOTO 50
C
C   IF THE PRESENT CHARACTER IS NONE OF THE ABOVE THEN CYCLE
C       BACK TO STATEMENT 48, STEP TO THE NEXT CHARACTER
C       IN THE STRING AND TEST AGAIN.
C
        GOTO 48
C
C   ONCE A BLANK IS DETECTED, BRANCH TO THIS SECTION OF CODE AND
C       CONTINUE STEPPING THROUGH THE CHARACTER STRING TESTING
C       FOR BLANKS, OPEN PARENS, OR CLOSE PARENS.
C
41      PN=PN+1
        IF(PN.GT.50)GOTO 40
        IF(CS(PN).EQ.8224)GOTO 41
        IF(CS(PN).EQ.8232)GOTO 50
C
C   IF A CHARACTER IS DETECTED THAT IS NOT A BLANK, OPEN PARENS,
C       OR A CLOSE PARENS THEN IT IS ASSUMED THAT THIS IS THE
C       BEGINNING OF THE NEXT PIECE OF INFORMATION AND RETURNS
C       TO THE CALLING PROGRAM.
C
```

```
        IF(CS(PN).NE.8233)GOTO 40
C
C   IF EITHER A OPEN OR CLOSE PARENS IS DETECTED THEN ADVANCE TO THE
C       NEXT CHARACTER AND TEST FOR A BLANK.
C
50      PN=PN+1
        IF(PN.GT.50)GOTO 40
C
C   IF A BLANK IS FOUND, CONTINUE STEPPING THROUGH THE STRING
C       UNTIL THE FIRST NON-BLANK CHARACTER IS DETECTED THEN RETURN
C       TO THE CALLING PROGRAM.
C
        IF(CS(PN).EQ.8224)GOTO 50
40      RETURN
        END
```


LIMIT
ENTER
CALL LOCPT
TO DECODE NUMBER FROM ASCII FORMAT INTO INTEGER FORM. STORE THIS NUMBER IN BEG
IS THERE A SECOND NUMBER TO DECODE
?
NO
YES
IS NEXT SYMBOL A BLANK OR A PARENS
?
NO
YES
PRINT ERROR MESSAGE ON TERMINAL
CALL LOCPT
TO DECODE SECOND NUMBER INTO AN INTEGER FORM. STORE THIS NUMBER IN FIN
STORE PREVIOUSLY DECODED INTEGER VALUE (BEG) IN FIN.
COMPUTE DIFFERENCE BETWEEN FIN AND BEG. STORE THIS VALUE +1 IN DIF
RETURN

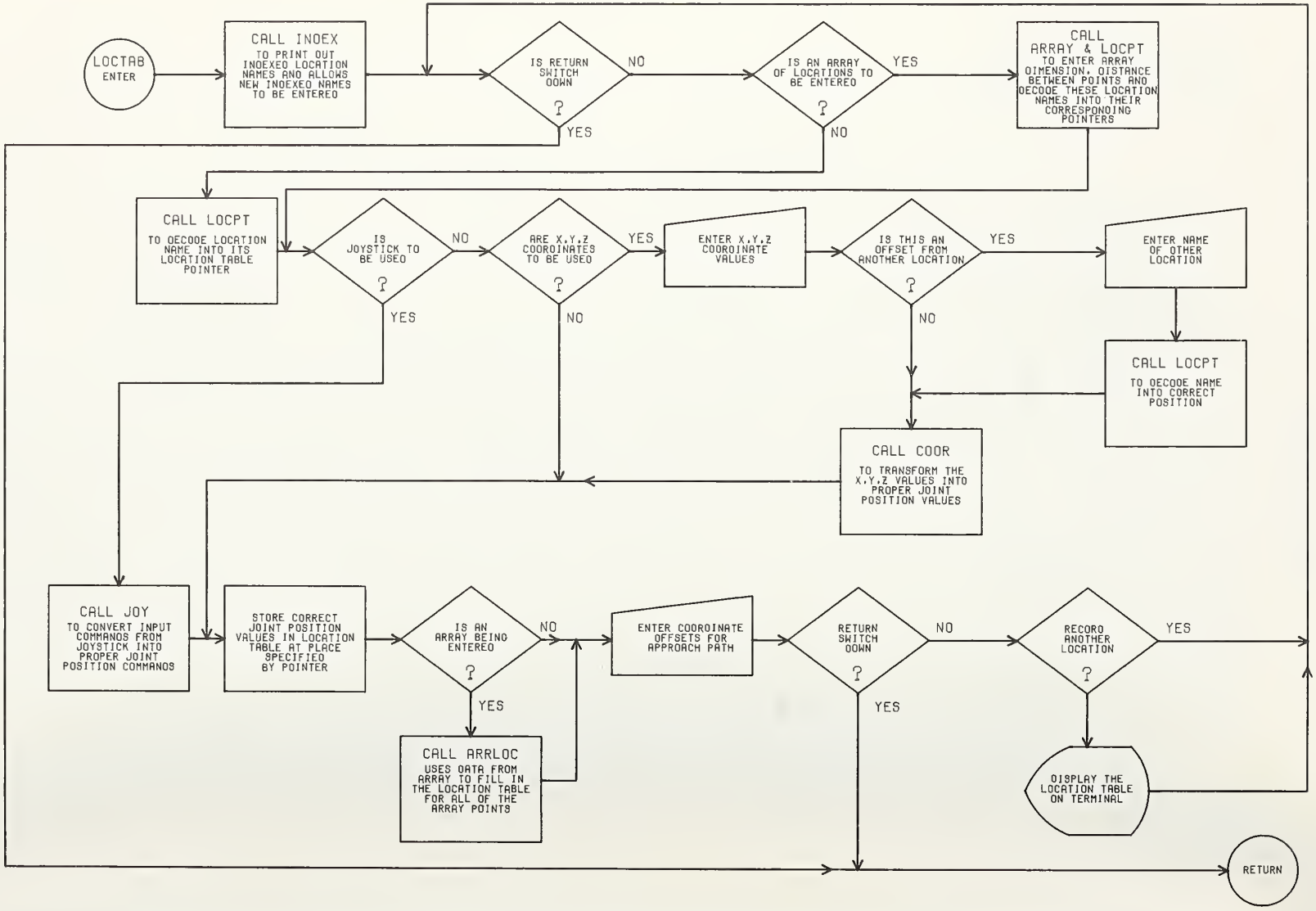
```
C --- SUBROUTINE : LIMIT
C
C --- ARGUMENTS :
C
C --- CALLED BY : INSERT  DELETE  PRINT
C
C --- CALLS SUBROUTINES : LOCPT  NEXT
C
C --- INPUT DATA : CS(50)       = THE CHARACTER STRING THAT CONTAINS A
C                                  NUMBER OR A RANGE OF NUMBERS TO BE
C                                  DECODED FROM ASCII INTO INTEGER FORM.
C                  PN           = THE POSITION IN THE STRING CS(50) OF
C                                  THE FIRST LINE NUMBER TO BE DECODED.
C                  BN           = THE INTEGER VALUE OF THE LINE NUMBER
C                                  DECODE FROM THE ASCII FORMAT BY A CALL
C                                  TO THE SUBROUTINE 'LOCPT'.
C
C --- OUTPUT DATA : BEG          = THE VARIABLE IN THE COMMON BLOCK THAT
C                                  RETURNS THE INTEGER VALUE OF THE FIRST
C                                  PROGRAM MODULE LINE NUMBER SPECIFIED
C                                  IN THE EDITOR COMMAND.
C                   FIN          = THE VARIABLE IN THE COMMON BLOCK THAT
C                                  RETURNS THE INTEGER VALUE OF THE LAST
C                                  PROGRAM MODULE LINE NUMBER SPECIFIED
C                                  IN THE EDITOR COMMAND.
C                   DIF          = THE DIFFERENCE (THE NUMBER OF LINES)
C                                  BETWEEN 'BEG' AND 'FIN'.
C
C --- FUNCTION: TO RETURN THE BEGINNING AND ENDING PROGRAM MODULE LINE
C               NUMBERS THAT THE PARTICULAR EDITOR COMMAND SPECIFIES
C               FOR ITS OPERATION.
C
        SUBROUTINE LIMIT
        IMPLICIT INTEGER(B-Z)
        COMMON/PTN/BN,EN
        COMMON/BED/BEG,FIN,DIF,ERR
        COMMON/CMD/CS(50),PN,PLN
        ERR=0
C
C   TRANSLATE THE ASCII FORMAT BEGINNING LINE NUMBER INTO AN
C       INTEGER FORMAT AND STORE IN 'BEG' BY A CALL TO
C       'LOCPT'.
C
        CALL LOCPT(1)
        BEG=BN-90
        IF(BEG.LE.0)GOTO 100
        IF(BEG.GT.100)GOTO 100
C
C   TEST FOR A DASH(-), CLOSE PARENS()), OR A BLANK( ) AND BRANCH
C       TO THE APPROPRIATE STATEMENT.
C
        IF(CS(PN).EQ.8237)GOTO 30
        IF(CS(PN).EQ.8233)GOTO 45
        IF(CS(PN).EQ.8224)GOTO 45
100     WRITE(6,101)
101     FORMAT(8X'FORMAT ERROR IN PROGRAM MODULE LINE NUMBER')
```

```
        ERR=1
        GOTO 40
C
C   IF A CLOSE PARENS WAS DETECTED THEN THIS INDICATES ONLY ONE LINE
C       WAS SPECIFIED THEREFORE SET 'FIN' EQUAL TO 'BEG' .
C
45      FIN=BEG
        GOTO 60
C
C   IF A DASH WAS DETECTED, THIS INDICATES THAT AN ENDING
C       NUMBER IS ALSO SPECIFIED THEREFORE ADVANCE TO THE NEXT NUMBER
C       IN THE STRING CS(50) (BY A CALL TO 'NEXT') AND TRANSLATE IT INTO
C       THE INTEGER FORMAT AND STORE IN 'FIN' (BY A CALL TO 'LOCPT')
C
30      PN=PN+1
        IF(CS(PN).NE.8224)GOTO 31
        CALL NEXT
31      CALL LOCPT(1)
        FIN=BN-90
C
C   TEST TO MAKE CERTAIN THAT THE LINE NUMBERS ARE WITHIN THE
C       ALLOWABLE RANGE.
C
60      IF(FIN.LE.0)GOTO 100
        IF(FIN.GT.100)GOTO 100
C
C   CALCULATE THE NUMBER OF LINES SPECIFIED AND STORE IN THE
C       COMMON VARIABLE 'DIF'.
C
        DIF=FIN-BEG+1
40      RETURN
        END
```


ADJUST
ENTER
SET PROGRAM TABLE LINE INDICATOR EQUAL TO ONE
DOES THIS LINE HAVE DESTINATION POINTER
?
YES
NO
SET THIS LINE'S PRESENT LOCATION POINTER EQUAL TO THE DESTINATION POINTER VALUE FROM PREVIOUS LINE
LAST LINE IN PROGRAM TABLE
?
NO
YES
INCREMENT PROGRAM TABLE LINE INDICATOR
SET ALL VALUES IN THIS LINE TO ZERO
LAST LINE IN PROGRAM TABLE
?
YES
NO
RETURN
INCREMENT LINE INDICATOR

```
C --- SUBROUTINE : ADJUST
C
C --- ARGUMENTS :
C
C --- CALLED BY : INSERT  DELETE  AVOID
C
C --- CALLS SUBROUTINES :
C
C --- INPUT DATA : PROG(105,6)   = THE PRESENT PROGRAM MODULE.
C
C --- OUTPUT DATA : PROG(105,6)    = THE VERIFIED PROGRAM MODULE.
C
C --- FUNCTION: MAKES CERTAIN THAT FOR EVERY LINE IN THE PROGRAM
C               MODULE, THE PRESENT LOCATION POINTER IS THE SAME
C               AS THE DESTINATION POINTER OF THE PREVIOUS LINE
C               THEREBY ASSURING CONTINUITY IN THE COMMANDED
C               POINT-TO-POINT MOTION.
C
        SUBROUTINE ADJUST
        IMPLICIT INTEGER(B-Z)
        COMMON/PMOD/PROG(105,6),ENDP,BRNCH(6),M
C
C   LOOP THROUGH THE PROGRAM MODULE, TESTING TO DETERMINE IF
C       EACH PROGRAM LINE HAS A DESTINATION POINTER.  IF IT
C       DOES, THEN SET THAT LINE'S PRESENT LOCATION POINTER EQUAL
C       TO THE PREVIOUS LINE'S DESTINATION POINTER.  IF THE PROGRAM
C       LINE DOES NOT HAVE A DESTINATION POINTER, THEN ZERO ITS
C       PRESENT LOCATION POINTER AND BRANCH TO STATEMENT 207.
C
200     DO 201 M=2,100
        IF(PROG(M,2).NE.0)GOTO 201
        PROG(M,1)=0
        GOTO 207
201     PROG(M,1)=PROG(M-1,2)
C
C   THIS LOOP ZEROS OUT THE REMAINDER OF THE PROGRAM MODULE TO ELIMINATE
C       ANY EXTRANEOUS DATA.
C
207     DO 208 RM=M,100
        DO 208 MM=1,6
208     PROG(RM,MM)=0
40      RETURN
        END
```

Location Module
(Module #3)


LOCTAB ENTER
CALL INOEX
TO PRINT OUT INOEXED LOCATION NAMES AND ALLOWS NEW INOEXED NAMES TO BE ENTERED
IS RETURN SWITCH DOWN ?
NO
YES
IS AN ARRAY OF LOCATIONS TO BE ENTERED ?
YES
NO
CALL ARRAY & LOCPT
TO ENTER ARRAY DIMENSION, DISTANCE BETWEEN POINTS AND DECODE THESE LOCATION NAMES INTO THEIR CORRESPONDING POINTERS
CALL LOCPT
TO DECODE LOCATION NAME INTO ITS LOCATION TABLE POINTER
IS JOYSTICK TO BE USED ?
NO
YES
ARE X,Y,Z COORDINATES TO BE USED ?
YES
NO
ENTER X,Y,Z COORDINATE VALUES
IS THIS AN OFFSET FROM ANOTHER LOCATION ?
YES
NO
ENTER NAME OF OTHER LOCATION
CALL LOCPT
TO DECODE NAME INTO CORRECT POSITION
CALL COOR
TO TRANSFORM THE X,Y,Z VALUES INTO PROPER JOINT POSITION VALUES
CALL JOY
TO CONVERT INPUT COMMANDS FROM JOYSTICK INTO PROPER JOINT POSITION COMMANDS
STORE CORRECT JOINT POSITION VALUES IN LOCATION TABLE AT PLACE SPECIFIED BY POINTER
IS AN ARRAY BEING ENTERED ?
NO
YES
CALL ARRLOC
USES DATA FROM ARRAY TO FILL IN THE LOCATION TABLE FOR ALL OF THE ARRAY POINTS
ENTER COORDINATE OFFSETS FOR APPROACH PATH
RETURN SWITCH DOWN ?
NO
YES
RECORD ANOTHER LOCATION ?
YES
DISPLAY THE LOCATION TABLE ON TERMINAL
RETURN

```
C --- SUBROUTINE : LOCTAB
C
C --- ARGUMENTS :
C
C --- CALLED BY : SAMPLE
C
C --- CALLS SUBROUTINES : ARMIN  INDEX  LOCPT  ARRAY  ARRLOC  JOY  POS
C                         COOR
C
C --- INPUT DATA : INBUF(28)    = WHEN THIS SWITCH IS UP, CAUSES THE
C                                 PROGRAM TO JUMP BACK TO THE PREVIOUS
C                                 DATA ENTRY REQUEST.
C                  INBUF(29)    = WHEN THIS SWITCH IS DOWN, CAUSES THE
C                                 CONTROL TO RETURN TO THE CALLING
C                                 PROGRAM.
C                  INBUF(32)    = WHEN THIS SWITCH IS UP, THE ARM IS
C                                 UNDER JOYSTICK CONTROL.
C                  INBUF(34)    = WHEN THIS SWITCH IS UP, THE LOCATION IS
C                                 TO BE ENTERED AS AN 'X,Y,Z' COORDINATE
C                                 POSITION.
C
C --- OUTPUT DATA : LTAB(90,4,7)  = THE UPDATED LOCATION TABLE MODULE.
C
C --- FUNCTION: USES A QUESTION AND ANSWER SYSTEM TO OBTAIN THE
C              NECESSARY INFORMATION FOR THE DESCRIPTION OF A
C              LOCATION AND THE CONSTRUCTION OF THE LOCATION TABLE
C              MODULE.
C
      SUBROUTINE LOCTAB
      IMPLICIT INTEGER(B-R)
      IMPLICIT INTEGER(T-Z)
      COMMON/ARMBUF/INBUF(64),OUTBUF(64)
      COMMON/LMOD/LTAB(90,4,7),PRES,DEST
      COMMON/OUT/JPOS(8)
      COMMON/ARR/B1,E1,J1,J2,AD1(3),AD2(3),DA
      COMMON/IND/IN(30),FL(6),NAME(3),NPT(3,20)
      COMMON/PTN/BN,EN
      COMMON/CORT/AC(6),AXEP,AYEP,AZEP
      EQUIVALENCE(EABORT,INBUF(28))
      EQUIVALENCE(NRE,INBUF(29)),(JSTK,INBUF(32)),(XYZ,INBUF(34))
      CALL ARMIN
C
C   MONITOR SWITCH 29 ('NRE') TO DETERMINE WHEN CONTROL SHOULD BE
C       RETURNED TO CALLING PROGRAM.
C
500   IF(NRE.GT.-2000)GOTO 40
C
C   INITIALIZE THE FLAG 'DA' AT ZERO.  USED TO INDICATE WHEN AN ARRAY
C       OF LOCATION POINTS ARE TO BE STORED.
C
      DA=0
C
C   THE FOLLOWING CODE READS THE PRESENT INDEXED NAMES OUT OF THE
C       LOCATION TABLE MODULE, AND BY A CALL TO 'INDEX', PRINTS
C       THESE INDEXED NAMES ON THE TERMINAL AND PROVIDES THE OPERATOR
C       THE OPPORTUNITY TO CHANGE THESE NAMES.
```

```
C
        M=0
        DO 104 J=89,90
        DO 104 K=1,4
        DO 104 L=1,7
        M=M+1
        IF(M.EQ.31)GOTO 98
104     IN(M)=LTAB(J,K,L)
98      CALL INDEX(0)
C
C   THE FOLLOWING CODE PRINTS OUT THE FUNCTIONS OF THE VARIOUS CONTROL
C       SWITCHES AND ASKS THE OPERATOR IF A SINGLE LOCATION OR AN ARRAY
C       OF LOCATIONS IS TO BE ENTERED.
C
303     WRITE(6,300)
300     FORMAT(8X'---SW 32 UP--- FOR JOYSTICK CONTROL',/
        1 8X'---SW 34 UP--- FOR XYZ ENTRY',/
        1 8X'---SW 32 & 34 DOWN---FOR MANUAL ENTRY',//)
        WRITE(6,201)
201     FORMAT(8X'***TYPE +1*** TO RECORD AN ARRAY OF INDEXED LOCATIONS',/
        1 8X'***TYPE [CR]*** TO RECORD LOCATIONS ONE AT A TIME')
        READ(6,202)DA
202     FORMAT(I3)
        CALL ARMIN
        IF(NRE.GT.-2000)GOTO 40
        IF(EABORT.LT.-2000)GOTO 500
C
C   THE OPERATORS RESPONSE IS TESTED AND IF AN ARRAY OF POINTS IS
C       TO BE ENTERED, THEN THE SUBROUTINE 'ARRAY' IS CALLED.
C
        IF(DA.EQ.0)GOTO 275
        CALL ARRAY
C
C   THE LOCATION TABLE POINTER FOR THE FIRST LOCATION TO BE ENTERED
C       IS STORED IN THE VARIABLE 'BN' ('B1' IS THE FIRST LOCATION OF
C       THE ARRAY DECODED AND RETURNED FROM THE CALL TO 'ARRAY').
C
        BN=B1
        GOTO 321
C
C   IF ONLY A SINGLE LOCATION IS TO BE ENTERED, THEN THE VALUE OF ITS
C       LOCATION TABLE POINTER IS DECODED BY A CALL TO 'LOCPT',
C       RETURNED IN THE VARIABLE 'BN' AND STORED IN BOTH
C       'B1' AND 'E1', THE VARIABLES THAT ARE THE FIRST AND LAST
C       LOCATIONS TO BE ENTERED.
C
275     CALL LOCPT(0)
        B1=BN
        E1=BN
C
C   THE VALUE OF THE SWITCH FOR JOYSTICK CONTROL IS CHECKED AND IF UP,
C        THE SUBROUTINE 'JOY' IS CALLED TO TRANSFER CONTROL TO THE
C       JOYSTICK BOX.
C
321     IF(JSTK.GT.-2000)GOTO 400
        CALL JOY
```

```
        GOTO 401
C
C   THE VALUE OF THE SWITCH FOR 'X,Y,Z' ENTRY IS CHECKED AND IF UP, THEN
C       THE 'X,Y,Z' COORDINATE VALUES FOR THE LOCATION ARE
C       REQUESTED.
C
400     IF(XYZ.GT.-2000)GOTO 401
765     WRITE(6,304)
304     FORMAT(8X'ENTER X,Y,Z VALUES (DECIMAL CENTIMETERS)',/
        1 8X'OF THE LOCATION TO BE RECORDED')
        READ(6,305)A1X,A1Y,A1Z
305     FORMAT(3(F10.3))
        CALL ARMIN
        IF(EABORT.LT.-2000)GOTO 303
C
C   THE OPERATOR IS NOW ASKED IF THESE COORDINATES ARE ABSOLUTE
C       VALUES OR DELTA OFFSETS FROM THE PREVIOUS VALUES FOR
C       THE PRESENT LOCATION OR FROM ANOTHER PREVIOUSLY ENTERED
C       LOCATION.
C
766     WRITE(6,306)
306     FORMAT(8X'***TYPE +1***IF THESE ARE ABSOLUTE VALUES',/
        1 8X'***TYPE [CR]***IF OFFSETTING PRESENT LOCATION BY THESE AMOUNTS',/
        1 8X'***TYPE -1***IF THESE ARE OFFSETS FROM ANOTHER LOCATION')
        READ(6,307)RAD
307     FORMAT(I3)
        CALL ARMIN
        IF(EABORT.LT.-2000)GOTO 765
        DD=RAD
        IF(RAD.NE.-1)GOTO 176
C
C    IF OFFSET FROM ANOTHER PREVIOUSLY RECORDED LOCATION, HERE, THAT
C       LOCATION NAME IS ENTERED.
C
        WRITE(6,308)
308     FORMAT(8X'WHAT IS THE OTHER LOCATION YOU ARE OFFSETTING FROM ?')
        RAD=0.
        MNN=BN
C
C   THIS OTHER LOCATION NAME IS IDENTIFIED BY A CALL TO 'LOCPT',
C       AND THE VALUE OF ITS LOCATION TABLE POINTER RETURNED IN THE
C       COMMON VARIABLE 'BN'.
C
        CALL LOCPT(0)
C
C   THE VALUE OF THE JOINT POSITIONS FOR THIS OTHER POINT ('BN') ARE
C       STORED IN THE COMMON VARIABLE 'AC(1-6)' TO BE USED BY THE
C       COORDINATE TRANSFORMATION ROUTINE 'COOR' AS THE REFERENCE POINT
C       FROM WHICH TO OFFSET THE NEW LOCATION BY THE DELTA COORDINATE
C       VALUES ('A1X','A1Y','A1Z').
C
176     DO 402 JX=1,6
402     AC(JX)=LTAB(BN,1,JX)
        CALL COOR(A1X,A1Y,A1Z,RAD,0)
        IF(DD.NE.-1)GOTO 479
        BN=MNN
```

```
        GOTO 479
C
C   THE PRESENT VALUES OF THE JOINT
C       POSITION INDICATORS ARE ENTERED AS THE JOINT POSITION VALUES
C       FOR THIS DESIGNATED LOCATION BY A CALL TO 'POS' TO READ IN
C       THESE VALUES.
C
401     CALL POS
        IF(EABORT.LT.-2000)GOTO 500
C
C   THE PRESENT JOINT POSITION INDICATOR VALUES FROM THE CALL TO 'POS'
C       ARE ENTERED IN THE LOCATION TABLE FOR THIS
C       DESIGNATED LOCATION.
C
479     DO 415 JX=1,6
        AC(JX)=JPOS(JX)
415     LTAB(BN,1,JX)=JPOS(JX)
        IF(DA.NE.1)GOTO 711
        DA=0
        CALL ARRLOC
C
C   IF THE START OF THE APPROACH PATH IS TO BE A DELTA X,Y,Z AWAY
C       FROM THE END POINT, THOSE DELTA VALUES ARE ENTERED HERE.
C
711     WRITE(6,521)
521     FORMAT(8X'ENTER DELTA X,Y,Z (DECIMAL CM) TO START OF APPROACH PATH')
        READ(6,522)ADX,ADY,ADZ
522     FORMAT(3(F7.3))
        CALL ARMIN
        IF(EABORT.LT.-2000)GOTO 303
        HX=ADX*100
        HY=ADY*100
        HZ=ADZ*100
C
C   THE OPERATOR NOW SUPPLIES THE INFORMATION AS TO WHETHER THE HAND COMES
C       TO A STOP AT THIS LOCATION (FX=1) OR IT DOES NOT STOP
C       HERE (FX=0).
C
630     WRITE(6,109)
109     FORMAT(8X'***TYPE +1***IF HAND WILL STOP AT THIS LOCATION',/
        1 8X'***TYPE[CR]***IF HAND DOES NOT STOP HERE')
        READ(6,110)FX
110     FORMAT(I3)
        CALL ARMIN
        IF(NRE.GT.-2000)GOTO 40
        IF(EABORT.LT.-2000)GOTO 303
        DO 620 J=B1,E1
        LTAB(J,2,7)=FX
        LTAB(J,3,4)=HX
        LTAB(J,3,5)=HY
620     LTAB(J,3,6)=HZ
C
C   THE OPERATOR IS GIVEN THE OPTION OF DISPLAYING THE LOCATION
C       TABLE ON THE TERMINAL OR ENTERING A NEW LOCATION IN THE
C       TABLE.
C
```

```
        WRITE(6,111)
111     FORMAT(8X'***TYPE [CR]*** FOR ANOTHER POINT',/
        1 8X'***TYPE +1*** TO DISPLAY TABLE THEN RETURN TO BEGINNING')
        READ(6,112)HH
112     FORMAT(I3)
        CALL ARMIN
        IF(EABORT.LT.-2000)GOTO 303
        M=0
C
C   THESE NEXT LOOPS SCAN THE LOCATION TABLE AND IF A DELTA OFFSET IS
C       GIVEN TO THE START OF AN APPROACH PATH, THEN THE
C       JOINT POSITION VALUES OF THE START OF THIS APPROACH PATH
C       IS CALCULATED BY A CALL TO 'COOR' AND THESE VALUES
C       ENTERED IN THE TABLE ('LTAB(J,4,1-6)').  IF THERE IS NO
C       APPROACH PATH, THEN THE JOINT POSITION VALUES OF
C       THE END POINT ARE ALSO STORED IN 'LTAB(J,4,1-6).
C       THE NAMES OF THE INDEXED LOCATIONS ARE STORED IN THE LAST TWO
C       POSITONS IN THE LOCATION TABLE.
C
        DO 850 J=1,86
        IF(LTAB(J,1,1).EQ.0)GOTO 850
        AX=LTAB(J,3,4)/100.
        AY=LTAB(J,3,5)/100.
        AZ=LTAB(J,3,6)/100.
        DO 830 JB=1,6
830     AC(JB)=LTAB(J,1,JB)
        CALL COOR(0.,0.,0.,0,1)
        LTAB(J,2,1)=AXEP*100
        LTAB(J,2,2)=AYEP*100
        LTAB(J,2,3)=AZEP*100
        IF(AX.NE.0.)GOTO 2500
        IF(AY.NE.0.)GOTO 2500
        IF(AZ.NE.0.)GOTO 2500
        DO 2400 KBL=1,6
2400    LTAB(J,4,KBL)=LTAB(J,1,KBL)
        GOTO 850
2500    CALL COOR(AX,AY,AZ,0,0)
        DO 820 JA=1,6
820     LTAB(J,4,JA)=JPOS(JA)
850     CONTINUE
C
C   HERE THE NAMES OF THE INDEXED LOCATIONS ARE STORED IN THE LAST
C       TWO POSITIONS OF THE TABLE.
C
772     DO 704 J=89,90
        DO 704 K=1,4
        DO 704 L=1,7
        M=M+1
        IF(M.EQ.31)GOTO 798
704     LTAB(J,K,L)=IN(M)
C
C   IF 'HH' EQUAL TO ZERO (IE. A CARRIAGE RETURN WAS ENTERED), THEN
C       RETURN TO BEGINNING OF PROGRAM TO ENTER ANOTHER LOCATION.
C       IF 'HH'=1, THEN PRINT THE LOCATION TABLE OUT ON THE
C       TERMINAL.
C
```

```
798     IF(NRE.GT.-2000)GOTO 40
        IF(HH.NE.1)GOTO 303
        DO 120 KL=1,90
        CALL ARMIN
        IF(EABORT.LT.-2000)GOTO 500
        DO 120 NK=1,4
        WRITE(6,1201)KL,(LTAB(KL,NK,J),J=1,7)
1201    FORMAT(8(2X,I6))
120     CONTINUE
        GOTO 500
40      RETURN
        END
```

ARRAY
CNTER
REQUESTS INDEXED LOCATION NAME AND INDEXED VALUES OF POINTS TO BE RECORDED
READS IN NAME AND INDEX VALUES SPECIFIED BY OPERATOR
REQUESTS THE DIMENSIONS OF THE ARRAY
READS IN THE ARRAY DIMENSIONS
REQUESTS X,Y,Z DELTA OFFSETS BETWEEN THE LOCATION POINTS ALONG BOTH DIMENSIONS OF THE ARRAY
READS IN THESE DELTA OFFSET VALUES
DECODES INDEXED LOCATION NAME DETERMINES POINTERS FOR FIRST AND LAST INDEX VALUES SPECIFIED.
RETURN

```
C --- SUBROUTINE : ARRAY
C
C --- ARGUMENTS :
C
C --- CALLED BY : LOCTAB
C
C --- CALLS SUBROUTINES :
C
C --- INPUT DATA : PA(30)        = THE CHARACTER STRING FROM THE TERMINAL
C                                  THAT CONTAINS THE INDEXED NAME AND THE
C                                  STARTING AND ENDING INDEXED LOCATION
C                                  VALUE FOR THIS PARTICULAR ARRAY [FROM
C                                  THE TERMINAL].
C                  J1,J2         = THE DIMENSIONS OF THE ARRAY [FROM THE
C                                  TERMINAL].
C                  AD1(3)        = THE DELTA 'X,Y,Z' DISTANCES (IN
C                                  CENTIMETERS) BETWEEN THE LOCATION
C                                  POINTS ALONG THE FIRST DIMENSION ('J1')
C                                  OF THE ARRAY [FROM THE TERMINAL].
C                  AD2(3)        = THE DELTA 'X,Y,Z' DISTANCES BETWEEN THE
C                                  LOCATION POINTS ALONG THE SECOND
C                                  DIMENSION OF THE ARRAY [FROM THE
C                                  TERMINAL].
C
C --- OUTPUT DATA : J1,J2           = AS DESCRIBED ABOVE.
C                   AD1(3)          = AS DESCRIBED ABOVE.
C                   AD2(3)          = AS DESCRIBED ABOVE.
C                   B1              = THE LOCATION TABLE POINTER FOR THE
C                                     FIRST INDEXED LOCATION SPECIFIED
C                                     IN THE ARRAY.
C                   E1              = THE LOCATION TABLE POINTER FOR THE
C                                     LAST INDEXED LOCATION SPECIFIED IN
C                                     THE ARRAY.
C                   DA              = A FLAG SET EQUAL TO ONE TO INDICATE
C                                     THAT AN ARRAY OF LOCATION POINTS
C                                     HAS BEEN ENTERED.
C
C --- FUNCTION: TO READ IN FROM THE TERMINAL THE NECCESARY INFORMATION
C              TO CONSTRUCT THE CORRECT LOCATION TABLE VALUES FOR THE
C              ARRAY OF LOCATIONS SPECIFIED.  THE ACTUAL ENTERING
C              OF THE VALUES INTO THE TABLE WILL BE DONE BY THE
C              SUBROUTINE 'ARRPT'.
C
      SUBROUTINE ARRAY
      IMPLICIT INTEGER(B-Z)
      COMMON/ARMBUF/INBUF(64),OUTBUF(64)
      COMMON/ARR/B1,E1,J1,J2,AD1(3),AD2(3),DA
      COMMON/IND/IN(30),FL(6),NAME(3),NPT(3,20)
      EQUIVALENCE(EABORT,INBUF(28))
      DIMENSION PA(30)
C
C   READ IN FROM THE TERMINAL, THE INDEXED NAME AND THE RANGE OF
C      LOCATIONS, THE DIMENSIONS OF THE ARRAY, AND THE DELTA COORDINATE
C      OFFSETS ALONG EACH DIMENSION.
C
500   WRITE(6,200)
```

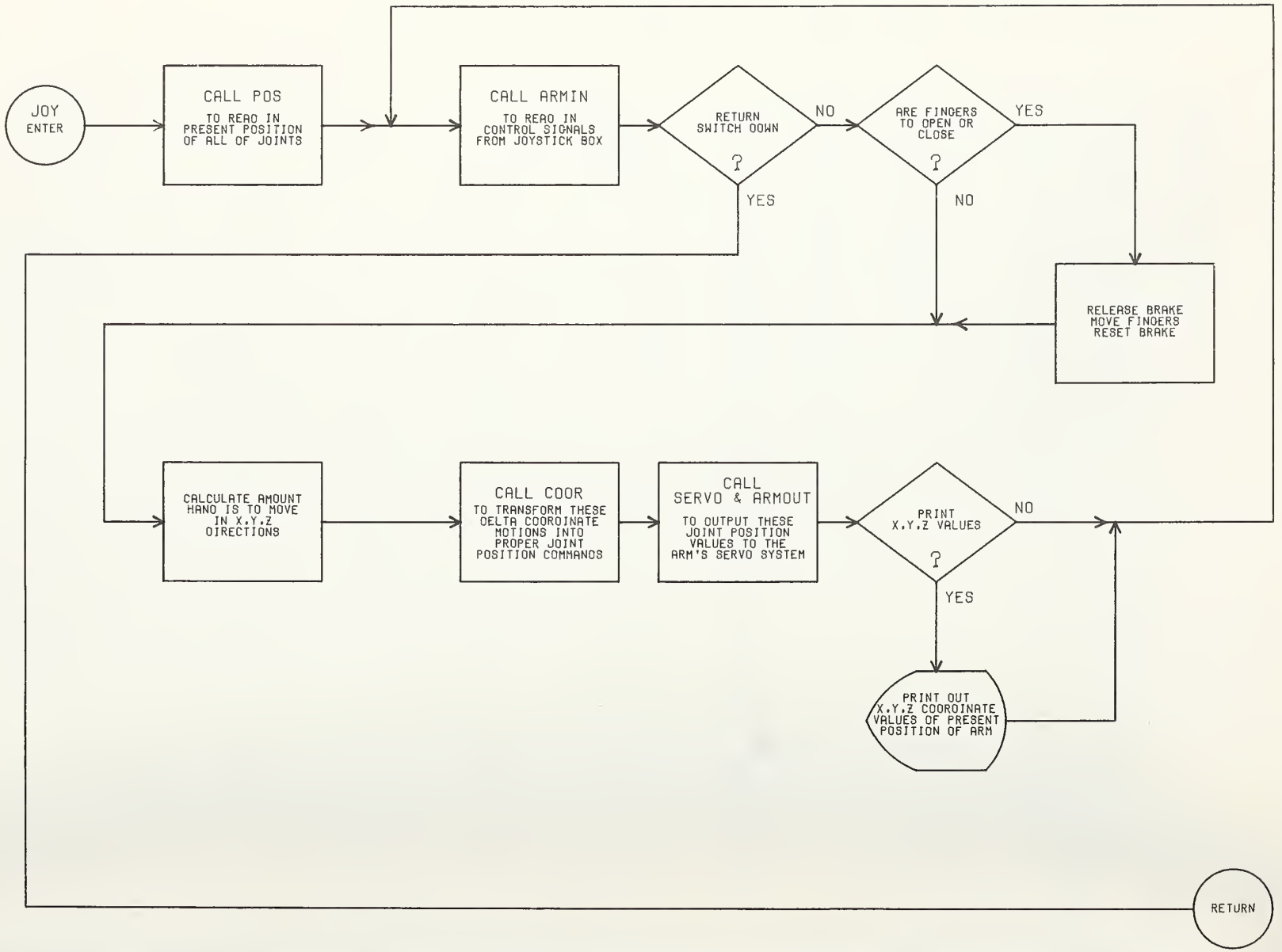
```
200     FORMAT(2X'ENTER LOCATION NAME AND INDEX NUMBERS',/
        1 2X'EG.       PALLET(01-20)')
        READ(6,201)(PA(J),J=1,30)
201     FORMAT(30(A1))
        WRITE(6,202)
202     FORMAT(2X'ENTER ARRAY DIMENSIONS',/
        1 2X'(EG.      4,5        )')
        READ(6,203)J1,J2
203     FORMAT(2(I3))
        WRITE(6,204)
204     FORMAT(2X'ENTER X,Y,Z DELTAS (DECIMAL CM) BETWEEN ',/
        1 2X'  ARRAY POSITIONS ALONG FIRST DIMENSION.')
        READ(6,205)(AD1(J),J=1,3)
205     FORMAT(3(F7.3))
        WRITE(6,214)
214     FORMAT(2X'ENTER X,Y,Z DELTAS (DECIMAL CM) BETWEEN ',/
        1 2X'  ARRAY POSITIONS ALONG SECOND DIMENSION.')
        READ(6,215)(AD2(J),J=1,3)
215     FORMAT(3(F7.3))
C
C   DECODE THE NAME AND THE SPECIFIED  INDICES  INTO THE CORRECT LOCATION
C       TABLE POINTERS.
C
        J=0
10      J=J+1
        IF(J.EQ.31)GOTO 40
        IF(PA(J).EQ.8224)GOTO 10
        DO 20 K=1,3
20      IF(PA(J).EQ.NAME(K))GOTO 7
        WRITE(6,120)PA(J),NAME(1),NAME(2),NAME(3)
120     FORMAT(4(A1))
        WRITE(6,100)
100     FORMAT(2X'CANNOT FIND THIS NAME,',/
        1 2X'***TYPE +1*** TO RE-ENTER NAME',/
        1 2X'*** TYPE [CR]*** TO ABORT.')
        READ(6,206)BOR
206     FORMAT(I3)
        IF(BOR.EQ.1)GOTO 500
        GOTO 40
C
C   DETERMINE WHAT THE FIRST SPECIFIED INDEX VALUE IS.
C
7       J=J+1
        IF(J.EQ.31)GOTO 40
        IF(PA(J).NE.8232)GOTO 7
8       J=J+1
        IF(J.EQ.31)GOTO 40
        IF(PA(J).EQ.8224)GOTO 8
        ONE=PA(J)
        TWO=PA(J+1)
        F=(10*(ONE-8240))+(TWO-8240)
        J=J+1
        IF(J.EQ.31)GOTO 40
C
C   DETERMINE WHAT THE LAST SPECIFIED INDEX VALUE IS.
C
```

```
17      J=J+1
        IF(J.EQ.31)GOTO 40
        IF(PA(J).EQ.8224)GOTO 17
        IF(PA(J).EQ.8237)GOTO 17
        ONE=PA(J)
        TWO=PA(J+1)
        L=(10*(ONE-8240))+(TWO-8240)
C
C   DECODE THE FIRST AND LAST INDEX NUMBERS INTO THEIR CORRESPONDING
C       LOCATION TABLE POINTERS AND STORE IN THE COMMON VARIABLES 'B1' AND
C       'E1'.
C
        B1=NPT(K,F)
        E1=NPT(K,L)
        DA=1
40      RETURN
        END
```

ARRLOC
ENTER
USE JOINT POSITION VALUES OF FIRST LOCATION IN THE ARRAY AS REFERENCE LOCATION FOR COORDINATE TRANSFORMATION ROUTINE
CALL COOR
TO CALCULATE JOINT POSITION VALUES FOR A LOCATION IN THE ARRAY USING DELTA OFFSET VALUES ENTERED IN ARRAY
STORE THESE JOINT POSITION VALUES IN LOCATION TABLE AT POSITION SPECIFIED BY APPROPRIATE POINTER
ALL ARRAY LOCATIONS STORED ?
YES
NO
CALCULATE TOTAL OFFSETS OF THE NEXT ARRAY LOCATION FROM FIRST POSITION IN ARRAY
RETURN

```
C --- SUBROUTINE : ARRLOC
C
C --- ARGUMENTS :
C
C --- CALLED BY : LOCTAB
C
C --- CALLS SUBROUTINES : COOR
C
C --- INPUT DATA : J1,J2          = THE DIMENSIONS OF THE ARRAY.
C                  AD1(3)         = THE DELTA 'X,Y,Z' DISTANCES (IN
C                                   CENTIMETERS) BETWEEN THE LOCATION
C                                   POINTS ALONG THE FIRST DIMENSION ('J1')
C                                   OF THE ARRAY.
C                  AD2(3)         = THE DELTA 'X,Y,Z' DISTANCES BETWEEN THE
C                                   LOCATION POINTS ALONG THE SECOND
C                                   DIMENSION OF THE ARRAY.
C                   B1            = THE LOCATION TABLE POINTER FOR THE
C                                   FIRST INDEXED LOCATION SPECIFIED
C                                   IN THE ARRAY.
C                   E1            = THE LOCATION TABLE POINTER FOR THE
C                                   LAST INDEXED LOCATION SPECIFIED IN
C                                   THE ARRAY.
C                   DA            = A FLAG SET EQUAL TO ONE TO INDICATE
C                                   THAT AN ARRAY OF LOCATION POINTS
C                                   HAS BEEN ENTERED.
C
C --- OUTPUT DATA : JPOS(1-6)       = THE JOINT POSITION VALUES FOR EACH OF
C                                     THE LOCATIONS IN THE SPECIFIED ARRAY.
C                   AC(1-6)         = THE JOINT POSITION VALUES THAT
C                                     CHARACTERIZE THE REFERENCE POINT USED
C                                     BY THE COORDINATE TRANSFORMATION
C                                     ROUTINE.
C
C --- FUNCTION: TO GENERATE AND STORE IN THE LOCATION TABLE, ALL OF THE
C               JOINT POSITION VALUES FOR THE LOCATIONS OF THE SPECIFIED
C               ARRAY.
C
      SUBROUTINE ARRLOC
      IMPLICIT INTEGER(B-Z)
      COMMON/ARMBUF/INBUF(64),OUTBUF(64)
      COMMON/LMOD/LTAB(90,4,7),PRES,DEST
      COMMON/OUT/JPOS(8)
      COMMON/CORT/AC(6),AXEP,AYEP,AZEP
      COMMON/ARR/B1,E1,J1,J2,AD1(3),AD2(3),DA
      EQUIVALENCE(EABORT,INBUF(28))
      DIMENSION AY2(3),AXY(3)
C
C   STORE THE LOCATION TABLE POINTER FOR THE FIRST INDEXED LOCATION
C       IN THE VARIABLE 'PAA'.
C
      PPA=B1
C
C   PLACE THE JOINT POSITION VALUES OF THIS FIRST INDEXED LOCATION IN THE
C       COMMON VARIABLE 'AC(1-6)' WHICH WILL BE USED AS A REFERENCE POINT
C       BY THE COORDINATE TRANSFORMATION ROUTINE 'COOR'.
C
```

```
        DO 15 J=1,6
15      AC(J)=LTAB(B1,1,J)
C
C   THESE LOOPS GENERATE THE CORRECT JOINT POSITION VALUES FOR ALL OF THE
C       LOCATIONS IN THE ARRAY. THIS IS DONE BY SPECIFYING THE
C       DELTA COORDINATE OFFSETS FOR EACH LOCATION IN THE ARRAY FROM
C       THE FIRST SPECIFIED REFERENCE LOCATION ('AC(1-6)') AND OBTAINING
C       FROM THE COORDINATE TRANSFORMATION ROUTINE THE CORRESPONDING JOINT
C       POSITION VALUES, THEN STORING THESE VALUES IN THE CORRECT
C       PLACE IN THE LOCATION TABLE.
C
        DO 110 J22=1,J2
        JS=J22-1
        DO 117 J4=1,3
117     AY2(J4)=JS*AD2(J4)
        DO 110 J11=1,J1
        JR=J11-1
        DO 120 JA=1,3
120     AXY(JA)=JR*AD1(JA)+AY2(JA)
        CALL COOR(AXY(1),AXY(2),AXY(3),0,0)
C
C   HERE THE JOINT POSITION VALUES ARE STORED IN THE LOCATION
C       TABLE UNDER THE CORRECT POINTER VALUE FOR THIS PARTICULAR NAME
C       AND INDEX NUMBER.
C
        DO 115 K=1,6
115     LTAB(PPA,1,K)=JPOS(K)
110     PPA=PPA+1
        RETURN
        END
```


JOY
ENTER
CALL POS
TO READ IN
PRESENT POSITION
OF ALL OF JOINTS
CALL ARMIN
TO READ IN
CONTROL SIGNALS
FROM JOYSTICK BOX
RETURN
SWITCH DOWN
?
NO
YES
ARE FINGERS
TO OPEN OR
CLOSE
?
YES
NO
RELEASE BRAKE
MOVE FINGERS
RESET BRAKE
CALCULATE AMOUNT
HAND IS TO MOVE
IN X,Y,Z
DIRECTIONS
CALL COOR
TO TRANSFORM THESE
DELTA COORDINATE
MOTIONS INTO
PROPER JOINT
POSITION COMMANDS
CALL
SERVO & ARMOUT
TO OUTPUT THESE
JOINT POSITION
VALUES TO THE
ARM'S SERVO SYSTEM
PRINT
X,Y,Z VALUES
?
NO
YES
PRINT OUT
X,Y,Z COORDINATE
VALUES OF PRESENT
POSITION OF ARM
RETURN

```
C --- SUBROUTINE : JOY
C
C --- ARGUMENTS :
C
C --- CALLED BY : LOCTAB
C
C --- CALLS SUBROUTINES : ARMIN  COOR  SERVO  POS
C
C --- INPUT DATA : JPOS(1-6)    = THE VALUE OF ALL OF THE JOINT POSITION
C                                 INDICATORS AT THE BEGINNING OF THIS
C                                 ROUTINE.
C                  INBUF(26)    = THE CHANNEL THAT CONTROLS THE VELOCITY
C                                 OF THE ARM WHILE UNDER JOYSTICK CONTROL.
C                  INBUF(27)    = THE CHANNEL THAT CONTROLS THE 'GRASP' AND
C                                 'RELEASE'.
C                  INBUF(31)    = THE CHANNEL THAT CALLS FOR A PRINT OUT ON
C                                 THE TERMINAL OF THE PRESENT 'X,Y,Z'
C                                 VALUES.
C                  INBUF(32)    = THE CHANNEL USED TO RETURN CONTROL TO THE
C                                 CALLING PROGRAM ('LOCTAB').
C                  INBUF(37)    = THE CHANNEL THAT CONTROLS THE WRIST
C                                 ROLL.
C                  INBUF(38)    = THE CHANNEL THAT CONTROLS THE WRIST
C                                 FLEX.
C                  INBUF(39)    = THE CHANNEL THAT CONTROLS THE HAND
C                                 ROLL.
C                  INBUF(40)    = THE CHANNEL THAT CONTROLS THE 'X' MOTION
C                                 OF THE HAND.
C                  INBUF(41)    = THE CHANNEL THAT CONTROLS THE 'Y' MOTION
C                                 OF THE HAND.
C                  INBUF(42)    = THE CHANNEL THAT CONTROLS THE 'Z' MOTION
C                                 OF THE HAND.
C
C --- OUTPUT DATA : AC(1-6)      = THE STARTING JOINT POSITION VALUES
C                                  THAT ARE USED AS A REFERENCE POINT
C                                  FOR THE COORDINATE TRANSFORMATION
C                                  CALCULATIONS.
C                  JPOS(1-8)     = THE JOINT POSITION VALUES THAT ARE
C                                  THE COMMANDED OUTPUT VALUES TO THE
C                                  SERVOS THAT WILL CAUSE THE HAND
C                                  TO MOVE IN THE DIRECTION INDICATED
C                                  BY THE JOYSTICK.
C
C --- FUNCTION: READS IN THE VALUES OF THE JOYSTICK POTENTIOMETERS,
C               CALCULATES THE PROPER JOINT POSITION VALUES
C               TO ACCOMPLISH THE COMMANDED MOVES AND OUTPUTS THESE
C               JOINT POSITION VALUES TO THE SERVOS.
C
      SUBROUTINE JOY
      IMPLICIT INTEGER (D-R)
      COMMON/ARMBUF/INBUF(64),OUTBUF(64)
      COMMON/OUT/JPOS(8)
      COMMON/CORT/AC(6),AXEP,AYEP,AZEP
      EQUIVALENCE(EABORT,INBUF(28))
      COMMON/XYZO/A04,A05,A06
      DIMENSION KIN(5),KBN(5),KMD(3)
```

```
      DATA KIN/37,38,39,27,42/
C
C   THE FLAG 'PTC' IS SET EQUAL TO ZERO.  THIS FLAG IS USED IN THE GRASP
C     AND RELEASE FUNCTIONS TO CONTROL THE SETTING OF THE HAND BRAKE.
C
      PTC=0
C
C   THE DELTA COORDINATE OFFSETS ('AX','AY','AZ') AND THE DELTA
C     ROTATIONAL OFFSETS ('A04','A05','A06') FOR THE WRIST AND HAND
C     ARE INITIALIZED TO ZERO.
C
      AX=0.
      AY=0.
      AZ=0.
      A04=0.
      A05=0.
      A06=0.
C
C   THE PRESENT JOINT POSITION VALUES ARE READ IN AND STORED IN THE
C     COMMON VARIABLE 'AC(1-6)' TO BE USED AS THE REFERENCE
C     POINT FOR ALL OF THE COORDINATE TRANSFORMATION CALCULATIONS
C     TO BE DONE BY 'COOR'.
C
      CALL POS
      DO 525 JJ=1,6
525   AC(JJ)=JPOS(JJ)
C
C   THE MESSAGE THAT TELLS WHICH SWITCHES CONTROL WHICH FUNCTIONS
C     IS SENT TO THE TERMINAL.
C
      WRITE(6,526)
526   FORMAT(3X'TO RETURN TO THE TEACH TRAJECTORY MODE,',/
      1 3X'FLIP SWITCH 32 TO THE DOWN POSITION.'/
      1 3X'FLIP SW 31 UP TO DISPLAY X Y Z VALUES.')
C
C   THE VALUES OF ALL OF THE INPUTS FROM THE JOYSTICK BOX ARE
C     READ IN.
C
20    CALL ARMIN
C
C   THE VARIABLE 'RT' CONTAINS THE VALUE OF THE SWITCH USED TO
C     RETURN TO THE CALLING PROGRAM ('LOCTAB').
C
      RT=-INBUF(32)
      IF(RT-2000)40,527,527
C
C   THE INPUT VALUES FOR THE 'X' AND 'Y' MOTION ARE STORED IN THE
C     VARIABLES 'IX' AND 'IY'.
C
527   IX=-INBUF(40)
      IY=-INBUF(41)
C
C   THE THREE ROTATIONAL ANGLES OF THE HAND-WRIST, THE FINGER CLOSING,
C     AND THE 'Z' MOTION INDICATOR ARE DECODED INTO THE NUMBER '2'
C     FOR POSITIVE MOTION, OR THE NUMBER '-2' FOR NEGATIVE MOTION,
C     OR THE NUMBER '0' FOR NO MOTION.  THESE CODED NUMBERS FOR
```

```
C       EACH OF THESE FIVE MOTIONS ARE STORED IN THE VARIABLE 'KBN(1-5)'.
C
        DO 100 KL=1,5
        FLL=KIN(KL)
100     KBN(KL)=((INBUF(FLL)+1850)/1000)*(-2)
C
C   THE VALUE OF THE POTENTIOMETER ON CHANNEL 26 IS READ IN AND USED AS
C       A CONTROL FOR THE SIZE OF THE COMMANDED MOVE FOR EACH OUTPUT,
C       THEREBY CONTROLING THE VELOCITY.  THIS CONTROL VALUE
C       IS STORED IN THE VARIABLE 'SPD'.
C
        SPD=((-INBUF(26)-2150)/10)+1
C
C   A VELOCITY FACTOR FOR EACH OF THE THREE ROTATIONAL ANGLES OF THE
C       HAND-WRIST IS GENERATED BY MULTIPYING THE VELOCITY CONTROL
C       VALUE ('SPD') BY THE PARTICULAR JOINT'S MOTION INDICATOR
C       NUMBER ('KBN(1-3)').
C
        DO 200 J=1,3
200     KMD(J)=KBN(J)*SPD
C
C   EACH OF THE THREE ANGLES IS OFFSET BY ITS CORRESPONDING VELOCITY
C       FACTOR ('KMD(1-3)').
C
        AO4=AO4+KMD(1)
        AO5=AO5+KMD(2)
        AO6=AO6+KMD(3)
C
C   THE FINGER MOTION INDICATOR IS CHECKED TO DETERMINE IFTHE
C       FINGERS ARE TO BE OPENED OR CLOSED.  IF NEITHER,
C       THEN BRANCH TO STATEMENT 300.
C
        IF(KBN(4).EQ.0)GOTO 300
C
C   THE FLAG 'PTC' IS SET EQUAL TO ONE, AND WILL BE USED TO SET THE BRAKE
C       ON THE FINGERS AFTER THIS FINGER MOTION IS FINISHED.
C
        IF(PTC.EQ.1)GOTO 75
        PTC=1
C
C   THE CORRECT VALUE IS SET IN 'JPOS(8)' TO RELEASE THE BRAKE ON THE
C       FINGERS.
C
        JPOS(8)=32767
        CALL SERVO(0)
75      IF(KBN(4).EQ.2)GOTO 301
C
C   HERE, THE COMMANDED JOINT POSITION VALUE TO THE FINGERS IS
C       INCREMENTED FOR EACH OUTPUT TO CAUSE THE FINGERS TO OPEN.
C
        IF(JPOS(7)-30000)305,310,310
305     JPOS(7)=JPOS(7)+200
        GOTO 310
C
C   HERE, THE COMMANDED JOINT POSITION VALUE TO THE FINGERS IS
C       DECREMENTED FOR EACH OUTPUT TO CAUSE THE FINGERS TO CLOSE.
```

```
C
301     IF(JPOS(7)-3500)310,308,308
308     JPOS(7)=JPOS(7)-200
310     CALL SERVO(0)
        GOTO 20
C
C   THE BRAKE FLAG IS CHECKED TO SEE IF IT IS SET ('PTC'=1).  IF IT IS,
C       THEN THE PROPER NUMBER IS PLACED IN 'JPOS(8) TO CAUSE THE
C       BRAKE ON THE FINGERS TO BE SET.  THE FLAG IS ALSO RESET TO
C       ZERO.
C
300     IF(PTC.EQ.0)GOTO 720
        PTC=0
        JPOS(8)=32703
        CALL SERVO(0)
C
C   A FLOATING POINT VARIABLE 'ASP' IS DEFINED AS THE VELOCITY FACTOR
C       'SPD' MULTIPLIED BY 1/300 OF A CENTIMETER.  THE NUMBER OF
C       THESE DELTA INCREMENTS TO BE MOVED IN THE 'X', 'Y', AND 'Z'
C       DIRECTIONS IS DETERMINED BY MULTIPLYING THIS DELTA STEP BY
C       A DECODED VALUE FROM THE INPUT CHANNEL FOR EACH COORDINATE AXIS.
C
720     ASP=SPD/300.
        MX=(IX-1070)/75
        BX=MX*ASP
        AX=AX+BX
        MY=(1070-IY)/75
        BY=MY*ASP
        AY=AY+BY
        BZ=KBN(5)*ASP
        AZ=AZ+BZ
C
C   ONCE THE DELTA MOTIONS IN EACH OF THE THREE COORDINATE AXES HAS BEEN
C       CALCULATED, THEN THESE DELTA 'X', 'Y', 'Z' VALUES ARE GIVEN TO THE
C       COORDINATE TRANSFORMATION ROUTINE SO THAT THE JOINT POSITIONS
C       FOR THESE MOTIONS CAN BE CALCULATED.
C
        CALL COOR(AX,AY,AZ,0,0)
C
C   THESE CALCULATED JOINT POSITION VALUES ARE THEN OUTPUTED TO
C       THE SERVOS.
C
        CALL SERVO(0)
C
C   IF SWITCH 31 IS SET THEN THE ABSOLUTE 'X', 'Y', 'Z' COORDINATE
C       POSITIONS ARE PRINTED OUT ON THE TERMINAL.
C
        OHH=INBUF(31)
        IF(OHH.GT.-2000)GOTO 20
        WRITE(6,591)AXEP,AYEP,AZEP
591     FORMAT(2X'X=',F10.2,/,2X'Y=',F10.2,/,2X'Z=',F10.2)
425     CALL ARMIN
        OHH=INBUF(31)
        IF(OHH.LT.-2000)GOTO 425
        GOTO 20
40      RETURN

        END
```

POS
ENTER
CALL ARMIN
TO READ IN
VALUE OF JOINT
POSITION INDICATORS
SCALE ALL THE
INPUT VALUES
AND STORE IN
JPOS(1-7)
RETURN

```
C --- SUBROUTINE : POS
C
C --- ARGUMENTS :
C
C --- CALLED BY : EMOVE  LOCTAB  JOY
C
C --- CALLS SUBROUTINES : ARMIN
C
C --- INPUT DATA : INBUF(5,8,11,13,15,17) = THE PRESENT VALUES OF THE
C                                           SIX JOINT POSITION INDICATORS.
C
C --- OUTPUT DATA : JPOS(1-6)      = THE SCALED VALUES OF THE SIX JOINT
C                                     POSITON INDICATORS AS CALCULATED IN
C                                     THIS SUBROUTINE.
C
C --- FUNCTION: SCALES THE JOINT POSITION INDICATOR VALUES INTO
C               A RANGE OF 0 TO 32767.
C
        SUBROUTINE POS
        IMPLICIT INTEGER(A-Z)
        COMMON/ARMBUF/INBUF(64),OUTBUF(64)
        COMMON/OUT/JPOS(8)
C
C   A CALL TO THE SUBROUTINE 'ARMIN' IS MADE TO READ IN THE VALUES
C       OF THE JOINT POSITION INDICATORS.
C
        CALL ARMIN
C
C
C   THE FOLLOWING TWO 'DO LOOPS' ARE USED TO
C       GENERATE THE CORRECT INPUT CHANNEL NUMBERS
C       (5,8,11,13,15,17,19) FOR THE SEVEN JOINT
C       POTENTIOMETERS.
C
        JPOS(1)=INBUF(5)*2+16383
        JPOS(2)=INBUF(8)*2+16383
        K=9
        DO 51 I=3,6
        K=K+2
        JPOS(I)=INBUF(K)*2+16383
51      CONTINUE
        JPOS(1)=(JPOS(1)*1.)-22.
        JPOS(2)=(JPOS(2)*.9879)+210.
        JPOS(3)=(JPOS(3)*1.001)-30.
        JPOS(4)=(JPOS(4)*1.0165)-283.
        JPOS(5)=(JPOS(5)*.9904)+45.
        JPOS(6)=(JPOS(6)*.997)+15
C       OUTBUF(7)=(OUTBUF(7)*1.0227)-400.
        RETURN
        END
```

NBS-114A (REV. 7-73)

U.S. DEPT. OF COMM. BIBLIOGRAPHIC DATA SHEET	1. PUBLICATION OR REPORT NO. NBS SP 500-23	2. Gov't Accession No.	3. Recipient's Accession No.

4. TITLE AND SUBTITLE

COMPUTER SCIENCE & TECHNOLOGY:

AN ARCHITECTURE FOR A ROBOT HIERARCHICAL CONTROL SYSTEM

5. Publication Date: December 1977

6. Performing Organization Code

7. AUTHOR(S) Anthony J. Barbera

8. Performing Organ. Report No.

9. PERFORMING ORGANIZATION NAME AND ADDRESS

NATIONAL BUREAU OF STANDARDS
DEPARTMENT OF COMMERCE
WASHINGTON, D.C. 20234

10. Project/Task/Work Unit No.

11. Contract/Grant No.

12. Sponsoring Organization Name and Complete Address *(Street, City, State, ZIP)*

Same as item 9.

13. Type of Report & Period Covered

14. Sponsoring Agency Code

15. SUPPLEMENTARY NOTES

Library of Congress Catalog Card Number: 77-17960

16. ABSTRACT *(A 200-word or less factual summary of most significant information. If document includes a significant bibliography or literature survey, mention it here.)*

Complex automation systems, such as industrial robots, require a computer-based control system for the effective utilization of this advanced technology. This report describes such a control system developed at the National Bureau of Standards. The approach has been to partition the control system into a hierarchy of different functional levels. This has proven to be a powerful technique in obtaining sensor-controlled robot behavior at a minimum cost of programming time and computer size. Further, this partitioning has greatly simplified the implementation of additional functions and sensors. This report discusses the control system, its implementation and use, and provides a documented listing of all of the control programs.

17. KEY WORDS *(six to twelve entries; alphabetical order; capitalize only the first letter of the first key word unless a proper name; separated by semicolons)*

Adaptive; automation; computer; control; goal-oriented; hierarchical control; robot; sensors

18. AVAILABILITY

[X] Unlimited

[ ] For Official Distribution. Do Not Release to NTIS

[X] Order From Sup. of Doc., U.S. Government Printing Office Washington, D.C. 20402, SD Cat. No. C13.10:500-23.

[ ] Order From National Technical Information Service (NTIS) Springfield, Virginia 22151

19. SECURITY CLASS (THIS REPORT) UNCLASSIFIED

21. NO. OF PAGES 227

20. SECURITY CLASS (THIS PAGE) UNCLASSIFIED

22. Price $4.25

USCOMM-DC 29042-P74

☆ U. S. GOVERNMENT PRINTING OFFICE :1977--261-238/424

ANNOUNCEMENT OF NEW PUBLICATIONS ON COMPUTER SCIENCE & TECHNOLOGY

Superintendent of Documents,
Government Printing Office,
Washington, D. C. 20402

Dear Sir:

Please add my name to the announcement list of new publications to be issued in the series: National Bureau of Standards Special Publication 500-.

Name _____

Company _____

Address _____

City _____ State _____ Zip Code _____

(Notification key N-503)

NBS TECHNICAL PUBLICATIONS

PERIODICALS

JOURNAL OF RESEARCH—The Journal of Research of the National Bureau of Standards reports NBS research and development in those disciplines of the physical and engineering sciences in which the Bureau is active. These include physics, chemistry, engineering, mathematics, and computer sciences. Papers cover a broad range of subjects, with major emphasis on measurement methodology, and the basic technology underlying standardization. Also included from time to time are survey articles on topics closely related to the Bureau's technical and scientific programs. As a special service to subscribers each issue contains complete citations to all recent NBS publications in NBS and non-NBS media. Issued six times a year. Annual subscription: domestic $17.00; foreign $21.25. Single copy, $3.00 domestic; $3.75 foreign.

Note: The Journal was formerly published in two sections: Section A "Physics and Chemistry" and Section B "Mathematical Sciences."

DIMENSIONS/NBS (formerly Technical News Bulletin)—This monthly magazine is published to inform scientists, engineers, businessmen, industry, teachers, students, and consumers of the latest advances in science and technology, with primary emphasis on the work at NBS. The magazine highlights and reviews such issues as energy research, fire protection, building technology, metric conversion, pollution abatement, health and safety, and consumer product performance. In addition, it reports the results of Bureau programs in measurement standards and techniques, properties of matter and materials, engineering standards and services, instrumentation, and automatic data processing.

Annual subscription: Domestic, $12.50; Foreign $15.65.

NONPERIODICALS

Monographs—Major contributions to the technical literature on various subjects related to the Bureau's scientific and technical activities.

Handbooks—Recommended codes of engineering and industrial practice (including safety codes) developed in cooperation with interested industries, professional organizations, and regulatory bodies.

Special Publications—Include proceedings of conferences sponsored by NBS, NBS annual reports, and other special publications appropriate to this grouping such as wall charts, pocket cards, and bibliographies.

Applied Mathematics Series—Mathematical tables, manuals, and studies of special interest to physicists, engineers, chemists, biologists, mathematicians, computer programmers, and others engaged in scientific and technical work.

National Standard Reference Data Series—Provides quantitative data on the physical and chemical properties of materials, compiled from the world's literature and critically evaluated. Developed under a world-wide program coordinated by NBS. Program under authority of National Standard Data Act (Public Law 90-396).

NOTE: At present the principal publication outlet for these data is the Journal of Physical and Chemical Reference Data (JPCRD) published quarterly for NBS by the American Chemical Society (ACS) and the American Institute of Physics (AIP). Subscriptions, reprints, and supplements available from ACS, 1155 Sixteenth St. N.W., Wash., D.C. 20056.

Building Science Series—Disseminates technical information developed at the Bureau on building materials, components, systems, and whole structures. The series presents research results, test methods, and performance criteria related to the structural and environmental functions and the durability and safety characteristics of building elements and systems.

Technical Notes—Studies or reports which are complete in themselves but restrictive in their treatment of a subject. Analogous to monographs but not so comprehensive in scope or definitive in treatment of the subject area. Often serve as a vehicle for final reports of work performed at NBS under the sponsorship of other government agencies.

Voluntary Product Standards—Developed under procedures published by the Department of Commerce in Part 10, Title 15, of the Code of Federal Regulations. The purpose of the standards is to establish nationally recognized requirements for products, and to provide all concerned interests with a basis for common understanding of the characteristics of the products. NBS administers this program as a supplement to the activities of the private sector standardizing organizations.

Consumer Information Series—Practical information, based on NBS research and experience, covering areas of interest to the consumer. Easily understandable language and illustrations provide useful background knowledge for shopping in today's technological marketplace.

Order **above** *NBS publications from: Superintendent of Documents, Government Printing Office, Washington, D.C. 20402.*

Order **following** *NBS publications—NBSIR's and FIPS from the National Technical Information Services, Springfield, Va. 22161.*

Federal Information Processing Standards Publications (FIPS PUB)—Publications in this series collectively constitute the Federal Information Processing Standards Register. Register serves as the official source of information in the Federal Government regarding standards issued by NBS pursuant to the Federal Property and Administrative Services Act of 1949 as amended, Public Law 89-306 (79 Stat. 1127), and as implemented by Executive Order 11717 (38 FR 12315, dated May 11, 1973) and Part 6 of Title 15 CFR (Code of Federal Regulations).

NBS Interagency Reports (NBSIR)—A special series of interim or final reports on work performed by NBS for outside sponsors (both government and non-government). In general, initial distribution is handled by the sponsor; public distribution is by the National Technical Information Services (Springfield, Va. 22161) in paper copy or microfiche form.

BIBLIOGRAPHIC SUBSCRIPTION SERVICES

The following current-awareness and literature-survey bibliographies are issued periodically by the Bureau:

Cryogenic Data Center Current Awareness Service. A literature survey issued biweekly. Annual subscription: Domestic, $25.00; Foreign, $30.00.

Liquified Natural Gas. A literature survey issued quarterly. Annual subscription: $20.00.

Superconducting Devices and Materials. A literature survey issued quarterly. Annual subscription: $30.00. Send subscription orders and remittances for the preceding bibliographic services to National Bureau of Standards, Cryogenic Data Center (275.02) Boulder, Colorado 80302.

U.S. DEPARTMENT OF COMMERCE
National Bureau of Standards
Washington, D.C. 20234

OFFICIAL BUSINESS

Penalty for Private Use, $300

POSTAGE AND FEES PAID
U.S. DEPARTMENT OF COMMERCE
COM-215

SPECIAL FOURTH-CLASS RATE
BOOK