DESIGN AND IMPLEMENTATION

OF

FLEXIBLE MICROPROCESSOR CONTROL

FOR

RETROFITTING

TO

FIRST GENERATION ROBOTIC DEVICES

by

J Middleton

A Master's Thesis submitted in
partial fulfilment of the
requirements for the award of
Master of Science of the
Loughborough University of Technology

February 1982

ACKNOWLEDGEMENTS

I would like to express my sincere gratitude to my Supervisor, Dr. R.H. Weston, whose help and encouragement made this work a success. Thanks are also due to Mr. G. Charles, Mr. N. Carpenter and Mr. P.D. Hanlon for their advice.

J.MIDDLETON

DESIGN AND IMPLEMENTATION OF FLEXIBLE MICROPROCESSOR CONTROL FOR RETROFITTING TO FIRST GENERATION ROBOTIC DEVICES

BY JANET MIDDLETON

This Master of Science project concerns the design and development of a flexible microprocessor-based controller for a Versatran Industrial Robot. The software and hardware are designed in modules to enhance the flexibility of the controller so that it can be used as the control unit for other forms of workhandling equipment.

The hardware of the designed controller is based on the Texas Instruments single board computer and interface printed circuit boards although some specially designed interface hardware was required. The software is developed in two major categories, which are "real-time" modules and "operator communication" modules. The real-time modules were for the control of the hydraulic servo-valves, pneumatic actuators and interlock switches, whilst the operator communication modules were used to assit the operator in programming "handling" sequences". The main advantages of the controller in its present form can be summarised thus:-

(i) The down-time between program changes is significantly reduced;

(ii) There can be many more positions programmed in a "handling sequence";

(iii) Greater control over axis dynamics can be achieved.

The software and hardware structure adopted has sufficient flexibility to allow many future enhancements to be provided. For example, as part of a subsequent research project additional facilities are being implemented as follows: a teach hand held pendant is being installed to improve still further the ease with which "handling sequences" can be programmed; improved control algorithms are being implemented and these will facilitate contouring; communication software is being included so that the controller can access via a node a commercially available local area network.

LIST OF CONTENTS

		PAGE
CHAPTER 1	INTRODUCTION	1
CHAPTER 2	A LITERATURE SURVEY OF ROBOTIC STRUCTURES	2
2.1	Reasons for Robots	2
2.2	Robot Components	3
2.3	Classification of Robot Components	3
2.3.1	Robot Anatomy	4
2.3.1.1	Arm Geometry	4
2.3.1.2	Wrist Assemblies	4
2.3.1.3	Three Fingered Hand	7
2.3.2	Drive Systems	9
2.3.3.1	Limited Sequence Robots	13
2.3.3.2	Playback Robots - with Point-to-Point Control	16
2.3.3.3	Playback Robots - with Continuous Path Control	18
2.4	Second Generation Robots	19
2.4.1	Proximity Sensor Technology for Manipulator End Effectors	20
2.4.2	Learning Systems in Manipulator Control	21
2.4.3	Classification of Robot Vision Systems	23
2.4.4	Existing First Generation Vision Systems	25
2.4.5	Current Second Generation Vision Systems	25
2.4.6	Tactile Sensing	25
2.4.6.1	A Compliant Device for Inserting a Peg in a Hole	26
2.4.7	Man-Robot Voice Communication	27
2.4.8	Total Self-Diagnostic Fault Tracing	27
2.4.9	Inherent Safety	27
CHAPTER 3	SOFTWARE SURVEY	28
3.1	Hierarchy of Software Structure	30
3.2	Special Robot Languages	36
3.2.1	MAL	37
3.2.2	WAVE	38
3.2.3	AL	39
3.2.4	Cincinnati Developments in Software	40
3.2.5	VAL	44
3.2.6	LAMA	44

	3.2.7	AUTOPASS	46
	3.2.8	EMILY	46
	3.2.9	SIGLA	47
	3.2.10	FREDDY	47
	3.2.11	CAMAC	47
	3.2.12	Adaptive Control System	48
	3.2.13	SAMMIE	48
	3.2.14	GRASP	48
	3.2.15	HELP	49
	3.2.16	RAPT	49
CHAPTER 4		THE COMPUTER, MICROCOMPUTER, TMS 9900 MICROPROCESSOR AND SOFTWARE/HARDWARE DEVELOPMENT SYSTEMS	52
	4.1	Historical Development of the Computer	52
	4.1.1	The Impact of Transistors	54
	4.2	Historical View of the Microcomputer	52
	4.3	Properties of Hardware and Software for the Implementation of Data Processing Algorithms	57
	4.4	Hardware/Software Tools for Microprocessors	57
	4.4.1	Hardware Aids	57
	4.4.2	Small Support Environments	58
	4.4.3	Simple Software Support	58
	4.4.4	Software Development Systems	59
	4.4.5	In-circuit Emulation	59
	4.4.6	Logic Analysers	60
	4.4.7	Software	60
	4.5	The Texas 9900 Family	63
	4.5.1	The Hardware Family	63
	4.6	The Software and Development System Support	67
	4.7	The Microcomputer	67
	4.8	Hardware	70
	4.8.1	The Central Processing Unit	70
	4.8.2	Memory Devices	72
	4.8.3	Input/Output Devices	72
CHAPTER 5		THE VERSATRAN ROBOT AND CONTROL SYSTEMS	74
	5.1	Control of the Versatran	77
	5.2	Hydraulic System	77
	5.3	Servo System	79

	5.4	Design of the Hardware for the Micro-processor Based Controller	79
	5.4.1.1	Introduction to the TMS9900 Microprocessor	79
	5.4.1.2	Programmable Systems Interface	81
	5.4.1.3	User Accessible Registers on the CPU	82
	5.4.1.4	Input/Output	83
	5.4.2	Interface Printed Circuit Boards	84
	5.5	Software and Hardware Development Facilities for use with the Texas Instruments 16 bit Microprocessor	86
CHAPTER 6		THE DEVELOPMENT OF REAL-TIME CONTROL SOFTWARE	88
	6.1	Introduction	88
	6.2	Closed-Loop Control for the Robot	88
	6.3.1	Control of One Axis	94
	6.3.2	Program TRY 1	94
	6.3.3	Program TRY 2	100
	6.3.4	Program TRY 3	102
	6.4	Program VERTHREE	107
	6.4.1	Intruction Format	107
	6.4.2	Description of VERTHREE	110
	6.4.2.1	Robot Intructions - Real Time Control Routines	114
	6.5	Programs Using Continuous Closed-Loop Control	132
	6.5.1	Program INT 1	132
	6.5.2	Program INT 2	147
	6.5.3	Program INT 3 and DATA	147
	6.5.4	Program INT4	147
CHAPTER 7		TESTS OBSERVATIONS AND RESULTS	150
	7.1	Robot Testing	150
	7.2	Controller Output and System Response Analysis	151
	7.3	Positional Repeatability	152
	7.4	Positional Repeatability	152
	7.4.1	Volumetric Accuracy Mapping (VAM)	160
CHAPTER 8		DEVELOPMENT OF A SOFTWARE LIBRARY	168
	8.1	Real Time Control Modules	168
	8.2	Task Programming Operator Communcations Modules	173
CHAPTER 9		FURTHER DEVELOPMENT OF AN OPERATOR COMMUNICATIONS MODULE	175

9.1	Program DATA INPUT	175
CHAPTER 10	PROJECT CONCLUSIONS AND FURTHER WORK	188
	References	190
Appendix 1	Robot Economics	
2	Specifications for Industrial Robots	
3	Mobile Robots	
4	Vision Systems	
5	Microprocessor Score Card	
6	Analogue to Digital and Digital to Analogue Converters	
7	Program Listings	
8	Texas Instruments TM990/101M Single Board Computer	
9	Versatran Specifications	

LIST OF FIGURES

		PAGE
2.1	Robot Arm Configurations	5
2.2	Typical Wrist Articulations	6
2.3	Model of the Crosseley's Hand	8
2.4	Detail of Finger Turnbuckle Mechanism	8
2.5	Semi-Rotary Actuator (Rotary Vane)	11
2.6	Semi-Rotary Actuator (Rack and Pinion)	12
2.7	Schematic Arrangement of a Typical Limited Sequence Robot	14
2.8	Analog Servo System	17
2.9	Hierarchy Feedback Loop Structure	22
3.1	Hierarchical Control System for Robot Installation	31
3.2	Hierarchical Control System for Sensory Information	35
3.3	Hand Held Teach Pendant	41
3.4	CRT and Keyboard	42
3.5	Alignment Fixture	43
3.6	Joint, World and Tool Modes	45
4.1	Organisation of a Logic Analyser	62
4.2	The 9900 Family	64
4.3	9900 Family CPUs	64
4.4	Microcomputer Support Components	65
4.5	TM990 Board Module and Software Support	66
4.6	990 Minicomputers	66
4.7	The 9900 Family Software and Development Systems	68
4.8	Microcomputer General Arrangement	69
4.9	CPU Block Diagram	71
4.10	ALU	71
4.11	Action of an interrupt	73
5.1	Dimensions of Versatran Robot	75
5.2	Mechanical Unit and Console	76
5.3	Versatran Interfacing Scheme	78
5.4	Servo-Valve Configuration	80
5.5	Servo-Amplifier Module	85
6.1	Open-Loop System	89
6.2	Closed-Loop System	89

6.3	Schematic Diagram of Closed-Loop Control	91
6.4	Output from DAC	92
6.5	Servo-Amplifier Module	93
6.6	Velocity Control	94
6.7	Overall Software for Program TRY 1	95
6.8	Detailed Flow Chart for Program TRY 1	96
6.9	Flow Chart for Program TRY 2 to input positions	101
6.10	Flow Chart for Program TRY 2	101
6.11	Flow Chart for Program TRY 3	103
6.12	Detailed Flow Chart for Program TRY 3	104
6.13	Controller Configuration - VERTHREE	111
6.14	Instruction Read and Decode	112
6.15	Select Axis to be Moved Routine	115
6.16	Initialise a Move in Swing	116
6.17	Initialise a Move in Vertical	116
6.18	Initialise a Move in Horizontal	117
6.19	Convert Analog Feedback and Calculate	118
6.20	Continue Cycle Routine	122
6.21	Stop Routine	123
6.22	Jump Routine	124
6.23	Move Wrist Movement	125
6.24	Open and Close Gripper Routine	127
6.25	Time Delay Routine	128
6.26	Overall Flow Chart for INT1	133
6.27	Detailed Flow Chart for INT1	136
6.28	Flow Chart to Open/Close Jaws	148
7.1	Trace Recording System	153
7.2	Description of Test Traces	154
7.3.1	Polar Coordinate Robot Work Volume	161
7.3.1	XYZ, Coordinate Robot Work Volume	161
7.4	Nodal Magnitudes and Machine Coordinates	162
7.5	Stand for Nodal Measurement	165
7.6	Accuracy Distribution Cirve	167
8.1	Software Structure for Versatran Robot	169
8.2	Op-codes for Versatran Robot	170
9.1	Flow Chart for Program DATA INPUT	176
9.2	Print Out from Program DATA INPUT	180
9.3	Flexible Operator Communications Module	182

LIST OF TABLES

		PAGE
3.1	Summary of Robot Languages	51
4.1	Comparison Between the Use of Hardware and Software for Implementing Algorithms	57
4.2	Survey of Available Microcomputer Development Systems	61
6.1	Programmable Instruction for Verthree	109
7.1	Output and Response Test Results	155
7.2	Test Results	158
7.3	Average Nodal X Readings	164

CHAPTER 1

INTRODUCTION

The main objective of this Master of Science project is to design, develop, and install a microprocessor-based controller for a Versatran Industrial Robot which demonstrates enhanced facilities when compared with the original control equipment. A high priority was attributed to the flexibility demonstrated by the designed controller, to allow the same hardware and software structure to be utilised for a wider range of applications in the control of work-handling equipment.

The Versatran Industrial Robot, which represents a fairly complex example of a first generation robot has six degrees of freedom. The three major axes, horizontal, vertical and swing, are controlled by closed-loop hydraulic servo-valves and the three wrist movements, yaw swing and pneumatically controlled to end stops. The original controller consisted of a rotating drum with pegs inserted to control delays and the movements of the wrist and a bank of potentiometers to control the positions of the major axes. The speed of the hydraulically controlled axes could be selected as either fast or slow and no intermediate speeds available. (The fast and slow speeds being related by a factor of four).

The microprocessor controller used for retrofitting is based on a Texas Instruments Single Board Computer and incorporates memory expansion and interface printed circuit boards to complete the hardware structure. The interface boards included digital to analogue and analogue to digital converters, servo amplifiers and solenoid drivers. A number of software modules were designed and implemented within two major categories - "real time control" software and "operator communication" software.

Software in the first category was configured to control the actual movement of the robot and was written in Assembly Language whilst the software in the second category was developed in both Pascal and Assembly Languages and facilitates the ease of programming the robot "handling" sequences.

A series of positional accuracy tests were conducted for the controller/ robot combination to evaluate the suitability of the approach adopted when programming robot handling sequences.

CHAPTER 2

A LITERATURE SURVEY ON ROBOTIC STRUCTURES

Robotics entered the English vocabulary with the translation of Karel Capek's play Rossum's Universal Robots in 1923, robot when translated means "worker".

Isaac Asimov[1] in 1940 had published a series of robotic stories. Asimov postulated roboticists with the wisdom to design robots that contained inviolable control circuitry to insure their always "keeping their place". The Three Laws of Robotics remain worthy design standards:

1 A robot must not harm a human being, nor through inaction allow one to come to harm.
2 A robot must always obey human beings, unless that is a conflict with the first law.
3 A robot must protect itself from harm, unless that is a conflict with the first or second laws.

To most people, the word "robot" brings to mind the robots from motion pictures such as "Star Wars". Thanks to the excellence of special photography, these manually operated robots appeared to us as true robots instead of as the hollow shells they were. Man has tried to duplicate nature by fashioning mechanical replicas of himself, of animals and even of birds. The development of these mechanical automatons from the mechanical fortune tellers and musical devices of the early 1900's back to the mechanical and musical clocks of the 16th and 17th centuries.

2.1 REASONS FOR ROBOTS

In a particular application the use of robots should be justified on either economic or humanitarian grounds[5]. Economic justification can be made if a process can be carried out cheaper and more efficiently than can be accomplished by humans[2], (see Appendix 1 for the factors to be considered). Robots should be employed on humanitarian grounds for boring and repetitive jobs which are psychologically damaging to humans and for dangerous or uncomfortable tasks in confined or hazardous environments which may be physically damaging to humans[3].

It would be socially irresponsible and financially unsound to attempt to replace all craftsmen with robots. The human hand-eye-brain co-ordination will not be surpassed by machines this century, if at all. In some cases a man-machine compound - the telechir could be used. The work "telechir" means 'hands at a distance' which aptly describes a system whereby a machine at one end of a cable slavishly copies the movements of the human operator at the other end. These machines could amplify human actions or diminutise them, as in the case of micro-manipulators. Furthermore they could incorporate some robotic elements where, for example, part of the task could be under control of an in-machine microprocessor, with a human operator overriding this for the more complex operations.

2.2 ROBOT COMPONENTS

All robots consist of the following components:-[6,7].

(i) there are the moving parts, chiefly comprising the arm, wrist and hand elements. The moving system is often referred to as the manipulator, but this term can be misleading, because it is easily confused with one of the robot's "near relations", the telecheric device;

(ii) the drive system which can be either hydraulic, pneumatic, electrical or a combination of these;

(iii) the control system, which at its simplest may consist of a series of adjustable mechanical stops and limit switches. At the other extreme are high technology computer based control systems which give the robot a programmable memory and which allows the robot drives to follow a path that is accurately defined all along its length by a series of continuously specified coordinates, and which can also be coupled with another computer or machine control system to synchronize the robot with its environment to increase efficiency and safety.

2.3 CLASSIFICATION OF ROBOT COMPONENTS

A description of the various robotic components are outlined in the following sections.

2.3.1 Robot Anatomy

Appendix 2 highlights some of the major features of available industrial robots. In observing the structure of industrial robots various observations can be made.

2.3.1.1 Arm Geometry

The robot's sphere of influence is based upon the volume into which the robot's arm can deliver the wrist subassembly. The robot arm configurations can be classified into:-

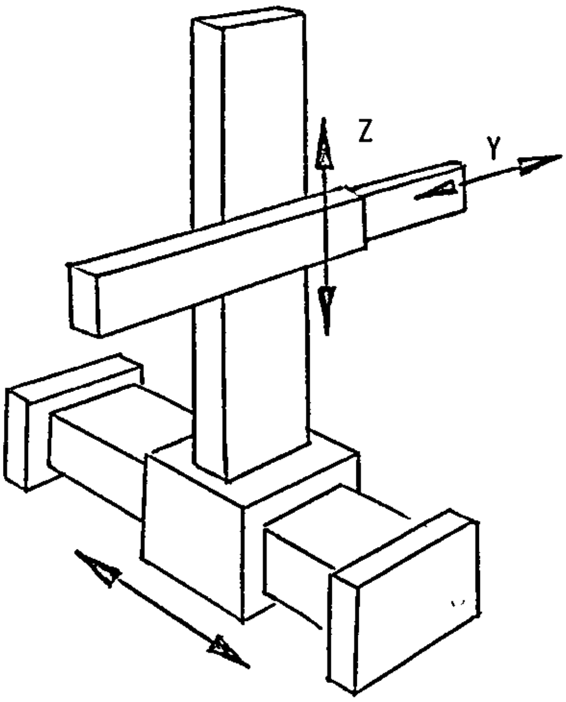
- (i) Cartesian coordinates
- (ii) Cylindrical coordinates
- (iii) Polar coordinates
- (iv) Revolute coordinates

Sketches of the typcial embodiments are shown in figure 2.1. Evidently each of these configurations offers a different shape to its sphere of influence, the total volume of which depends upon arm link lengths. For different applications the appropriate configuration can be used. For example, a revolute arm might be best for reaching into a tub, while a cylindrical arm might be best suited to a straight thrust between the dies of a press. (See Appendix 3 for the examination of mobile robots).

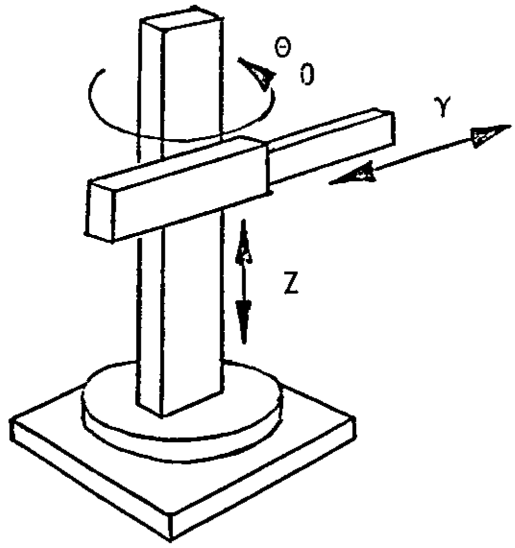
2.3.1.2 Wrist Assemblies

In every case the arm carries a wrist assembly to orient its end effector as demanded by the workpiece placement. Commonly, the wrist provides three articulations that offer motions labeled pitch, yaw and roll (analogous with aircraft technology as illustrated in figure 2.2).

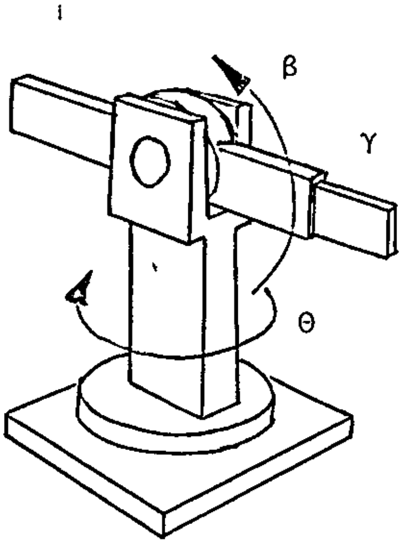
As robot hands are less adaptable than human hands, they have to be chosen or designed specially for a particular industrial application. Whereas the robots themselves have earned the reputation of being general purpose automation, the hands are not quite so flexible and may have to be included along with special tooling requirements for a specific task.



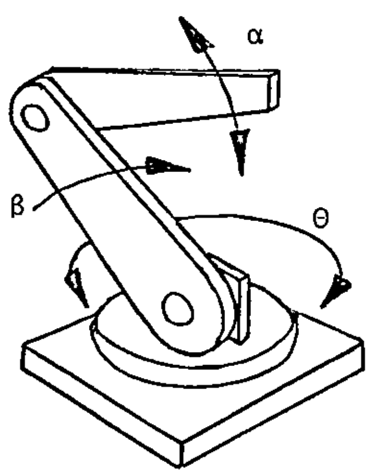
CARTESIAN COORDINATES



CYLINDRICAL COORDINATES



POLAR COORDINATES



REVOLUTE COORDINATES

FIGURE 2.1. ROBOT ARM CONFIGURATIONS

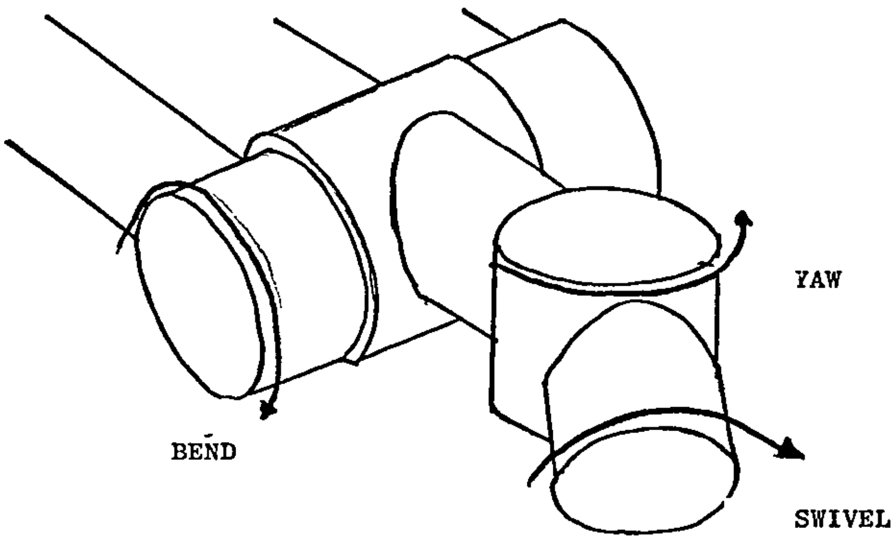


FIGURE 2.2 TYPICAL WRIST ARTICULATIONS

2.3.1.3 Three Fingered Hand

After considering the various functions that are performed by a hand Crossley et al[26] designed the following three-fingered hand.

The end effector had three digits, that is, a thumb and two fingers. The third finger needs to be separately motorized for trigger action. If the thumb and index finger are to work in opposition, one motor will suffice for these. However, if the hand is to provide the "hook" or "baggage lift" capability, the thumb needs to be left fully open while the index finger closes. The hand followed the anthropomorphic model as much as possible (see figure 2.3). The main transverse axis of the palm was chosen at 45^{o} to the longitudinal axis of the fore-arm and wrist. A method of bending the interphalangeal joints was used which had two important advantages. The mechanical advantage is upheld from the motor right to the joint, the velocity reduction and force augmentation being at the last possible moment and secondly the high forces to be encountered in the joint are combined with their reactions into a small triangle at each pivot. Figure 2.4 shows the scheme of these joints. The two phalanges, being of channel form, are directly hinged. The two are connected by a turnbuckle, with right and left-hand threaded eye-bolts. The buckle itself is a pinion, and driven by another pinion through a flexible cable within the finger. The other end of the cable can be driven directly by the motor through a reduction gear. By this design the moment of any lateral force imposed at the finger tip is carried by the structure of each phalanx and the joints, but it is not felt by the finger drive mechanism, except as a much reduced torque, and then only when the pinion turns, for the pitch of the screw makes the drive irreversible.

The parallel-jaw end effector was utilised which consisted of a set of parallelogram 4-bar linkages in cascade mounted in the side plates of both thumb and index finger. Their effect is to maintain the inside (gripping) surfaces of the ultimate phalanges of these two digits parallel to one another and perpendicular to the surface of the palm, even while the more proximal phalanges bend to form a cylindrical grip.

The gripping surfaces of the fingertips of the hand require cushioning (to accommodate themselves to various shapes to be grasped) and to have

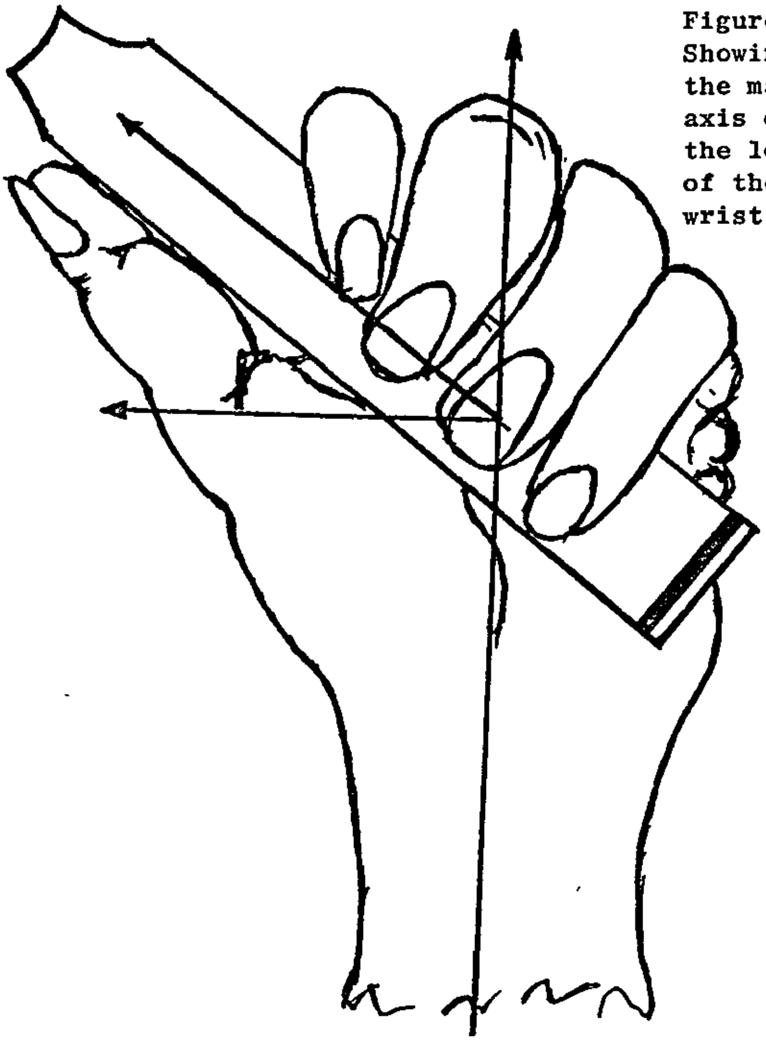


Figure 2.3
Showing the setting of the main transverse axis of the palm 45° to the longitudinal axis of the fore-arm and wrist.

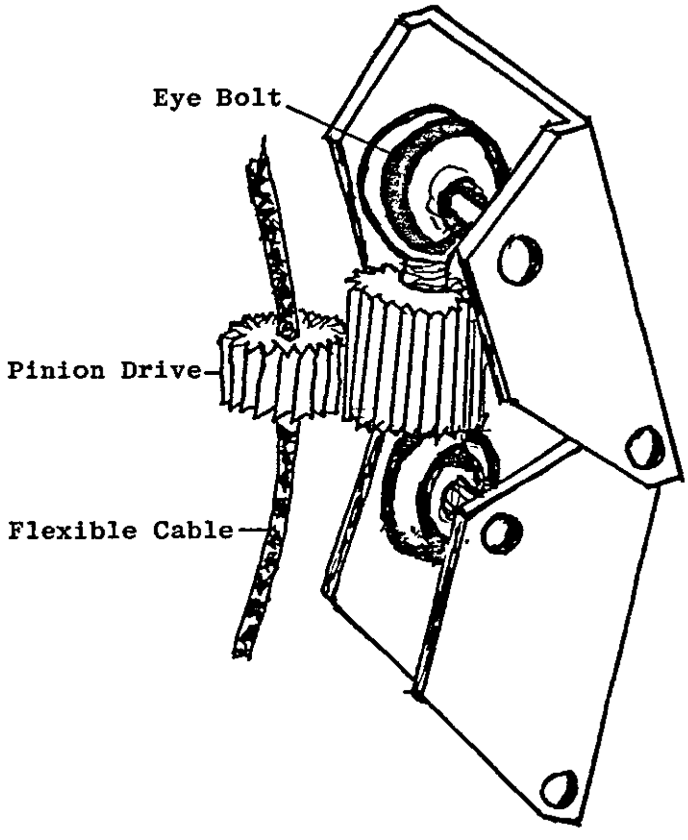


Figure 2.4
Detail of finger turnbukle mechanism

a high coefficient of friction as possible. To achieve this the inside gripping surfaces are covered with a layer about 3 mm thick of soft silicone rubber, which is cast in place. This material does not adhere to a metal surface, therefore before casting, the metal pressure plate is drilled with many holes and the plastic cast as a "sandwich" on both sides of the metal. Using this method the padding is held firmly in place even when heavily strained.

2.3.2 Drive Systems

A drive system is required for each robot articulation. In addition to driving the arm, hand and wrist, the grippers also need a drive mechanism for the functions of holding and releasing. Robot drives can be electrical, pneumatic or hydraulic or some combination.

Pneumatic systems are found in about 30% of robots; electromechanical drives in about 20%, typical forms are servomotors, stepping motors, pulse motors, linear solenoid and rotational solenoids; hydraulic drives account for the remainder [7].

Hydraulic drives can be divided into the following categories; cylinders (or jacks), hydraulic motors, and semi motors (or rotary actuators).[71]

(i) Cylinders or Jacks

These may be either single or double rodded, the advantage of the latter being that the characteristics are the same in both directions and the flow through the valve is symetrical in both directions.

(ii) Hydraulic Motors

A hydraulic motor is similar to a pump but it allows full pressure to be applied to both parts. A motor, together with its driving gear, rack and pinion, lead screw etc, will normally be appreciably more expensive than a jack. Its advantage lies in its small inertia and greater rigidity, giving a more positive action and one less influended by any disturbing force. Among the hydraulic motors there is a choice of piston, gear, vane and ball configurations. The choice is determined by several factors, such as application, whether the motion required is linear or rotary, performance, cost, reliability etc. The best choice is generally the simplest device that will do the job satisfactorily.

(iii) Semi-Motors or Rotary Actuators

Figure 2.5 shows a rotary vane contained in a circular housing. With the single vane shown, the maximum angle of rotation is about 300° but by having a double vane, with two inlets and outlets, the power for a given size can be doubled, whilst reducing the angle of rotation to about 100°.

Another type of semi-rotary actuator embodies a rack and pinion, the rack being actuated by one or more cylinders. Whichever system is chose, an electro-hydraulic servo valve is required, which is show in figure 2.6.

The present generation of robots which have electrical drives use rotational motors. These motors also require gearing or ball screws and a servo power amplifier to provide a complete actuation system.

The motor driven robot will have a much higher maintenance cost than the simpler cylinder (or jack) driven robots, not only because of the many more expensive components, but because of localized wear in gears and ball screws by fretting corrosion during active servoing.

In certain applications, such as paint spraying the environment may present an explosion hazard and the robot must either be explosion proof or intrinsically safe so as not to ignite the combustable environment. Here the hydraulically driven robot has the advantage over the electrical system as the electrical energy from the feedback devices and the energy to drive servo valves can be small enough not to ignite the explosive fuel-air mixture.

Another advantage for hydraulics is that this power method lends itself to robot applications because energy can easily be stored in an accumulator and released when a burst of robot activity is called for. As there is no convenient means to store electric energy, the electrically driven robots tend to underpower the drives.

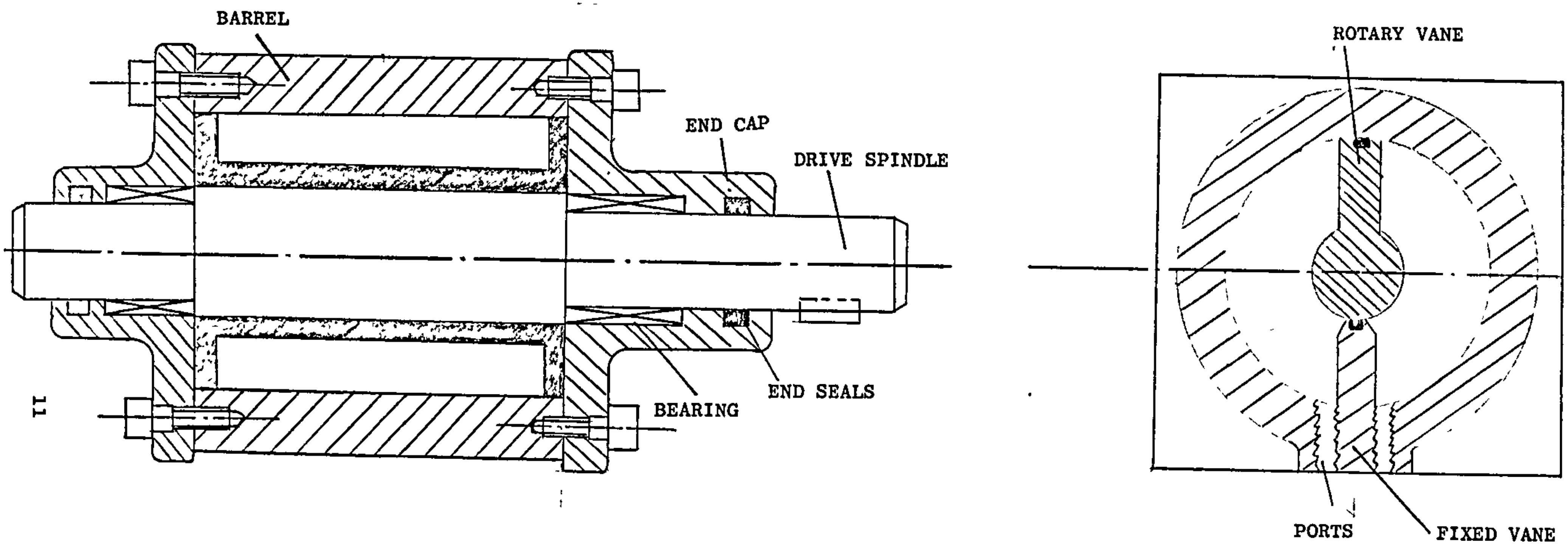


FIGURE 2.5 SEMI-ROTARY ACTUATOR (ROTARY VANE)

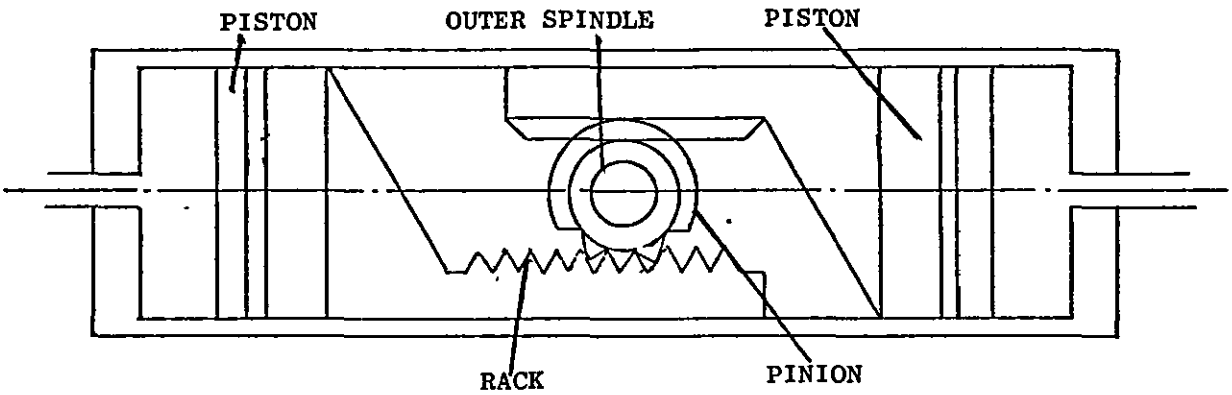


FIGURE 2.6. SEMI-ROTARY ACTUATOR (RACK AND PINION)

2.3.3 Control Systems

Introduction

The control system can be, broadly speaking, divided into the following three categories. Comparison between any two robots that belong to one of the categories could easily reveal that quite different drive systems had been employed to achieve roughly the same end. Control systems are likely to correspond more closely between robots in the same category.

2.3.3.1 Limited Sequence Robots

As its name implies, a limited sequence robot is at the least sophisticated end of the robot scale. Typically, these robots use a system of mechanical stops and limit switches to control the movements of arm and hand (see figure 2.7). Operation sequences can often be set up by means of adjustable plugboards, which are themselves associated with electromechanical switching, (usually this electromechnical switching is achieved by using a combination of relays and rotary or stepping switches). As a result of this type of control, only the end positions of robot limbs can be specified and controlled. The arm, for example, can be taken from point A to B, but the path between is not defined. The controls simply switch the drives on and off at the end of travel. This mode of operation has earned such machines the name of 'pick and place' robots.

The use of mechanical stops and limit switches gives good positional accuracy, which is typically repeatable to better than $\pm$ 0.5 mm. Limited sequence robots have been used successfully in a variety of applications, including die-casting press loading, plastic moulding and as part of special-purpose automation. This type of robot is used in applications where low cost is of major significance. Thus, historically their associated control equipment has been of corresponding low cost and inherent limited capability. This situation will be improved with many additional control features being available through the use of large scale integrated (LSI) devices without an appreciable increase in cost.

The number of movements possible in a total production sequence must be limited to the number of limit switches, stops and programmable switches contained by the robot. Such robots are not "taught" to

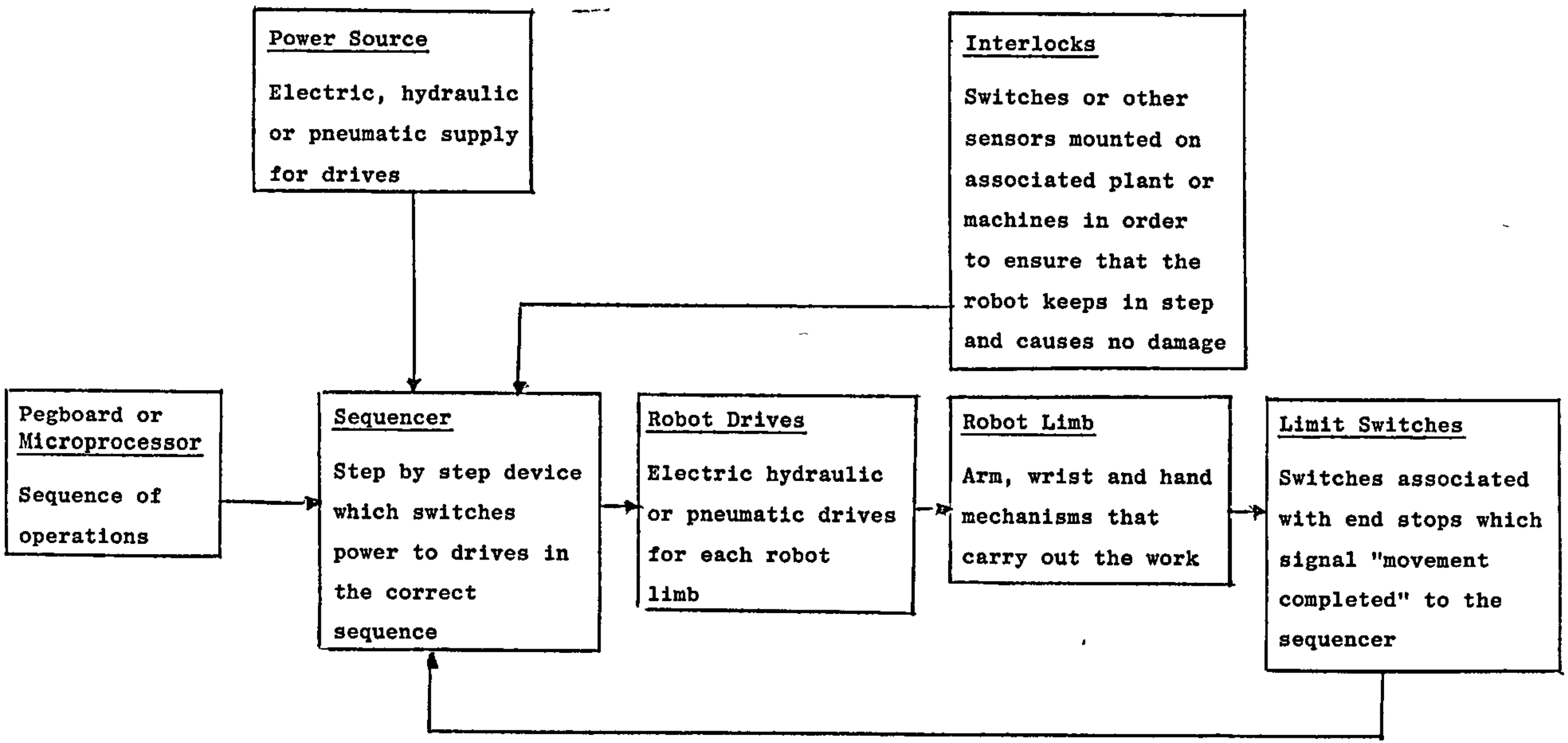


Figure 2.7 Schematic arrangement of a typical limited sequence robot

perform their job, but have to be set up in the same way as an automatic machine would be adjusted. There is no memory as such, other than that embodied in the settings of the plug board and all the mechanical stops.

Unlike robots in the other categories, the simple limited sequence control system cannot exercise any real control over the limbs while they are actually in motion. It is possible to provide more than one stopping point along each path, but the primitive nature of the memory system restricts the number of these for practical purposes.

The sequence of events which occur when a typical limited sequence device performs an operating sequence or task can be described as follows.

When an axis movement is required it is necessary for the controller to switch power to the relevant drive element. If the drives are electric, then the controller will probably close a relay to switch the current through. Where the drives are hydraulic or pneumatic, then appropriate solenoid valves are operated. The motion generated by the drive element normally continues until the moving limb is physically restrained by an end stop, the physical shock usually being "cushioned" by some form of shock absorbing device. Thus there are only two positions at which the moving part can come to rest, one at the beginning and the other at the end of a programmed move. Obviously, the system is arranged so that a limit switch cuts off the motive power as soon as the end stop is reached. When the initial movement has been finished, the limit switch not only cuts off the power, but it also signals to the controller that the particular movement has been finished, so that the next movement can start.

How does the controller ensure that the robot does not put its arm into the closing jaw of a press, or try to load a workpiece into a spinning chuck? The robot cannot see the machine it is trying to operate, there are no robot senses equivalent to those of a human operator. The method utilised to make the robot aware of the real world around it is by providing additional limit switches or other electrical sensing devices on the machine to be operated. These are connected to the controller to provide additional signals to the sequencer, complementary to those obtained from the switches mounted on the robot itself. Robot limb movements are therefore carefully interlocked with the machine being operated. This prevents the robot from trying to commit 'suicide', avoids collision damage to associated plant, and enables the robot to carry out its operations not only in the correct sequence, but also at the appropriate moment in time. However, such interlocks can only act as a safeguard relating to events which are predictable and unforseen events cannot be allowed for.

A characteristic of limited sequence robots is that they are generally difficult to reprogram. This is particularly true if hardwired control equipment is employed where the nature of the control system and memory, (which are all embodied in a complex and interdependent set of limit switches, interlocks, and stops and electrical connections) offers little flexibility. Not only does this kind of electromechanical arrangement prove tedious to change, but it also limits the number of different sequence steps that can be accommodated within a particular handling task.

2.3.3.2 Playback Robots - with Point-to-Point Control

Another method for achieving positional control of each limb relies on the use of some form of servo mechanism. Figure 2.8 illustrates a schematic representation of such a closed loop control scheme. Each movable robot limb is fitted with a device which produces an electrical signal, the value of which is usually proportional to the limb position. The system is arranged so that the direction of drive travel is such as to reduce the positional error,[8] and as the limb moves closer to the desired position this error signal automatically reduces until it becomes zero, and the limb stops in the correct position. This is analog control and in practice calls for a high degree of engineering skill in design to achieve satisfactory positional accuracy and freedom from oscillation.

If a time varying input is provided via a control panel to vary the command signal for a particular limb, then the limb will move as the knob is moved. Thus a form of remote control is achieved, and as many time varying inputs as there are limbs can be provided via the

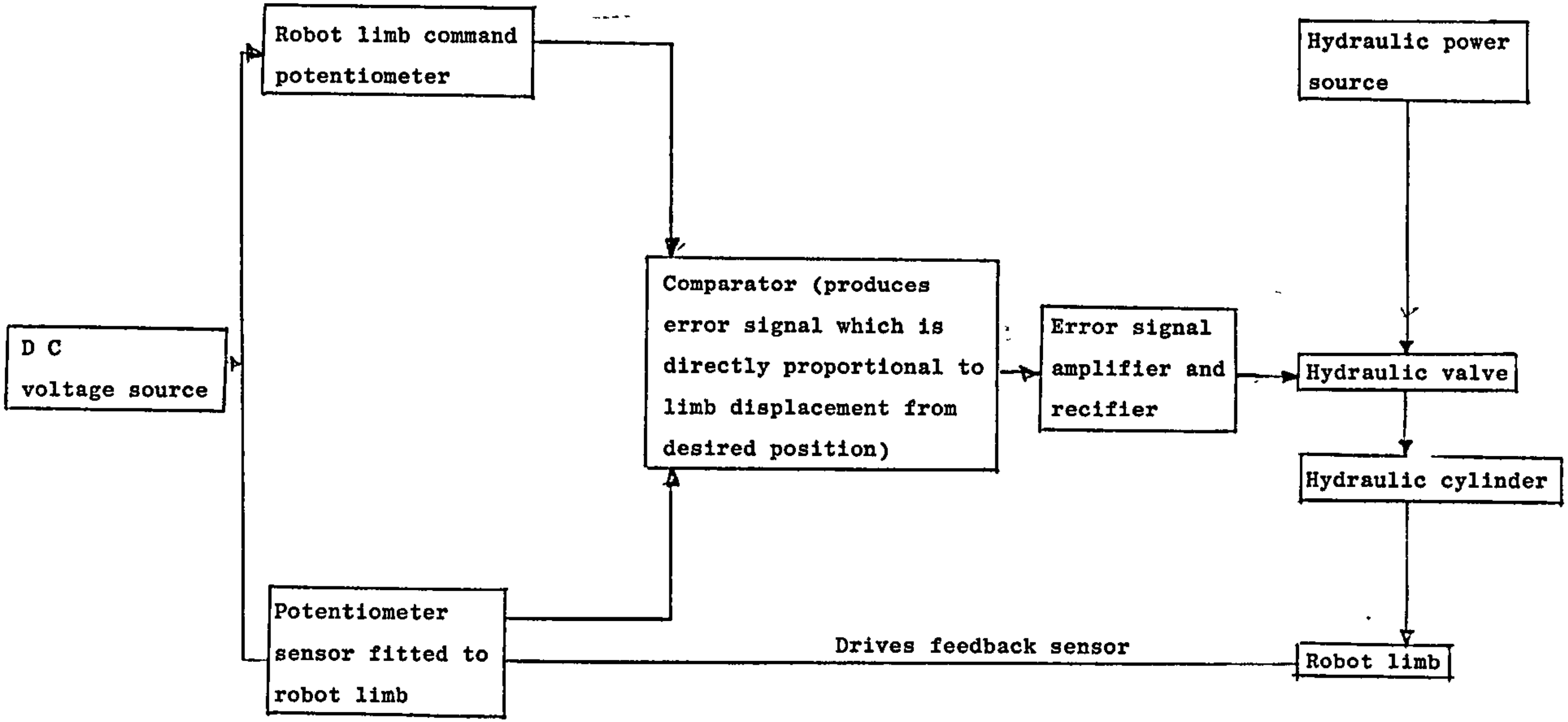


Figure 2.8 Analog servo system

control panel. Such a device so far is a manipulator and when a memory unit is added it becomes a flexible robot. The position of the limbs at each operational step and the total operational sequence can be recorded in the memory unit. The stored locations can then be recalled and used to stimulate all the servo systems. The procedure for setting up such a robot is far easier than for a limited sequence robot which can be achieved as follows:- either by inputting the required digital values into memory (which have been obtained by moving the robot to these positions) or to teach the robot i e to drive the robot limbs to the required positions for each operational step, and then record the exact condition of the robot in memory by the simple act of pushing a button before proceeding to drive the robot to the next step in the sequence or by pre-programming the positions in memory. However, it is evident that the control equipment for a "playback" robot will been to be more sophisticated than that for a "limited sequence" robot.

When the robot is commanded to move from one position to another, this could involve independent operation of two or more of its articulations. The only information that the robot knows is the attitude of all the limbs at the start and end of the move and will generally perform the moves as quickly as possible, moving all limbs simultaneously to fulfill the given command. In such an arrangement there is no definition of the paths which the robot limbs will trace between programmed points, hence the name point-to-point". Point-to-point robots are capable of doing any job performed by a "limited sequence" robot and presuming that their memory capacity is sufficient, they are also capable of performing demanding tasks such a pallitizing, stacking, spot welding etc.

2.3.3.3 Playback Robots - with Continuous Path Control

There are applications in manufacturing industry where it is necessary to control not only the start and finish points of each robotized step but also the path traced by the robot hand as it travels between these two extremes. An example of this requirement is provided by seam welding, where a robot is asked to control a welding gun, and move it along some complex contour at the correct speed to produce a strong and neat weld. One way of looking at this problem is to regard continuous path control as a logical extension of point-to-point control. It is feasible to provide a robot with a memory that is sufficiently large to allow path control that is, to all intents and purposes, continuous. Alternatively, the continuous path robot may be taught in real time. The operator leads the robot through the motions that it is required to perform at the correct speed. During this teaching process, the robot has to record the movement and hand attitudes continuously or approximately continuously, in its memory. This can be achieved by giving the robot an internal timing system, which for example, could be synchronized with the main supply frequency ($50M_3$). Using this time reference, the robot's movements can be sampled at the rate of 50 times each second, with the result being committed to memory. Even at this sampling speed, a large amount of data has to be accumulated in the memory, consequently magnetic tape units are often used. To increase the operational usefulness of continuous path robots provision is usually made for the playback speed of operation to be different from the teaching speed.

It is clear from the above description that a computer is required as the central element of any control system used for point-to-point or continuous path robots. The equation solving and storage capabilities of the computer allow it to be used to monitor and modify axis motions. Furthermore, providing that time constraints permit (see section 3.1.)[28], modern control algorithms can be incorporated, to allow position and velocity loops to be closed within the computer, thereby optimising the performance of the robot in terms of positioning accuracy and dynamic characterisitcs. The availability of low cost LSI devices will have particular impact here although it must be stressed that the wide range of possible axis configurations and servo-drive elements result in the need for computer controls with a corresponding large variety of interface hardware and controlling software.

2.4 SECOND GENERATION ROBOTS

High-precision assembly tasks by industrial robots require sensory feedback and an increased autonomous intelligence, necessary to cope with uncertainties caused by inaccuracies of the robot and by the changing environment.

An active adaptable compliant writ (AACW) has been designed by Van Brussel et al[11] enabling precision assembly with general purpose industrial robots. It uses force feedback as sensory information. A probabilistic learning algorithm, with minimal memory requirements has been developed and used in automatic assembly of closely fitting parts. The algorithm optimizes, by means of an appropriate rewarding rule and a properly chosen evaluation criterion, the probability relationship between the possible wrist actions. Visual and tactile-force information and free programmability are two key elements of the second generation of robots which allow manipulators to service a broader field of application including the more complex and high level tasks[12].

A main problem in the near future will be the development of control algorithms that translate the input and sensor signals into the right control commands. Conventional preprogramming of all possibilities in a real world environment soon becomes very tedious if not impossible, while for higher level tasks the interpretation of the measured process feedback signals can be of unsurmountable complexity. For handling these and related problems, the future generations of robots will need a degree for autonomous intelligence which makes their behaviour human-like. Despite the recent evolutions, there is still an enormous gap between artificial intelligence models and the feasibility and usefulness of practical realisations.

2.4.1 Proximity Sensor Technology for Manipulator End Effectors

A proximity sensor denotes a small device, suitable for mounting on a manipulator hand, which can detect the presence or approximate position of a nearby object without actual contact. Sensors are typically of the order of 1 cm in linear dimension but a separate electronic module may be required. Their basic function is to measure the effector-object position for use as an aid to manipulator control during grasping. Proximity sensors can be used to measure the position of either an effector as a whole, or the finger components individually with respect to the object to be grasped, alternatively they could be used as an obstacle detector to avoid hitting objects.

2.4.2 Learning Systems in Manipulator Control

Learning systems have a hierarchical feedback loop structure.

The lowest level in such a learning system is a simple feedback configuration with a fixed relation between input and output. The mathematical description of the process under control has to be completely known in order to able to design such a feedback controller (figure 2.9).

At the second level, the so-called adaptive loop, a system identification is performed and the basic feedback controller structure is adapted in accordance to the actual state of the process. Although it is no longer necessary to know exactly the dynamic characteristics of the process, it is still necessary to know how to influence the basic control algorithm as a function of the measured signals. The third level, the learning loop, teaches the adaptive loop how to change the basic controller in order to achieve optimal control. This learning loop is clearly distinguished from the two lower levels by a supplementary "teacher-input" which is used to evaluate the quality of the actual performance of the system with respect to a certain goal. It is this information of the "teacher" that the learning system accumulates as experience from the past and gives it its ability to gradually improve its behaviour in time.

The ability of learning systems to cope with problems with only a limited amount of prior information makes this kind of approach interesting for automatic manipulator control. The exact position of a robot arm, that takes into account all the dynamic parameters and non-linearities of a joint articulated manipulator configuration soon becomes too complex to be of any practical use. Similar problems arise in describing a real world working environment due to the vast amount of mostly unknown parameters. A more serious case of lack of prior information is found in the interpretation of the measured feedback signals in so called "higher level" tasks, as for example, the insertion of a peg into a hole or the grasping of the fragile object. The human reasoning in those situations is not fully understood.

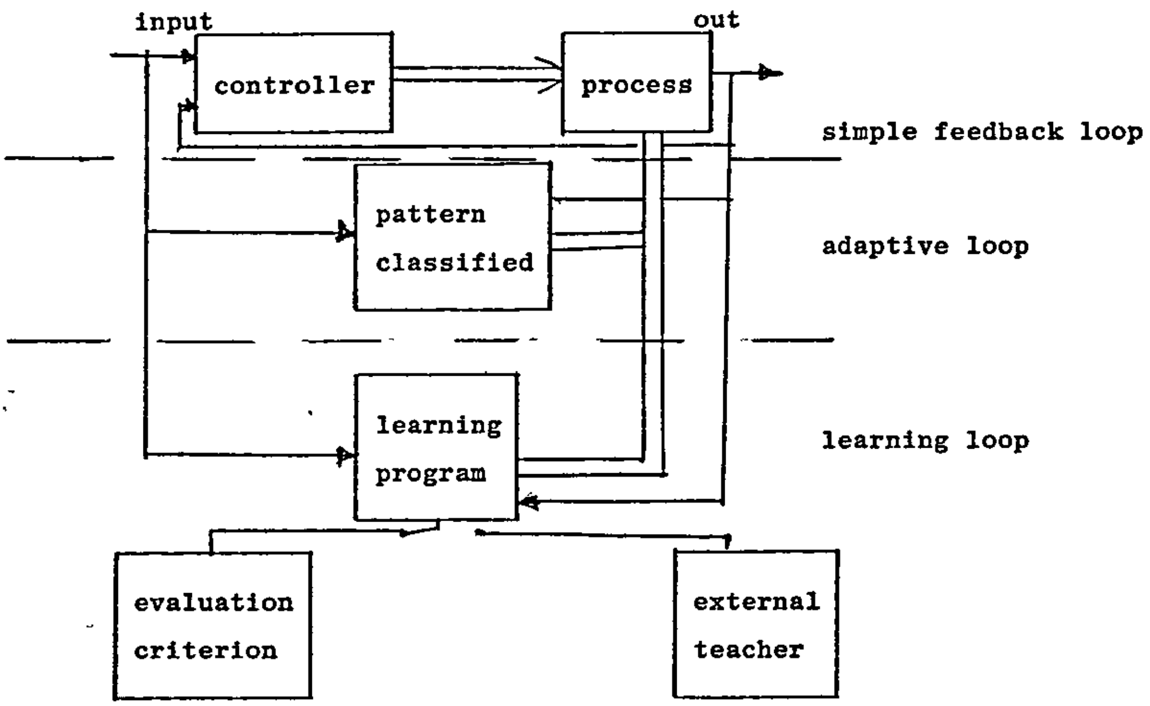


Figure 2.9 Hierarchy Feedback Loop Structure

Common artificial intelligence methods have little practical use in robot control applications because of the non-availability of adequate mathematical formulated optimization functionals. Pattern recognition with trainable thresholds or decision surfaces seems to hold more potential and base for their work. There is a high cost as large and fast computers needing special array processors are required.

2.4.3 Classifications of Robot Vision Systems

About ten years ago the first robotic vision systems appeared in research laboratories[21-23].

The first generation vision systems are capable of recognising components and determining their orientation. The components must be in strong contrast to their surroundings, must not touch or overlap other components, must have a limited number of stable states. Recognition of a part takes less than one second and the vision system can be taught a number of different components at a given time. The main targets for second generation robots should be to overcome the following constraints:-

(i) that each object should be in strong contrast to its background so that a silhouette image of the component can be easily obtained.

(ii) that each component be separated from its neighbours.

The first constraint is a function of computation time available for recognition in the industrial environment. Complex edge detection algorithms based on grey level images are currently capable of separating a component from realistic backgrounds but the computational time (at least by serial computers) is too long. The second constraint of non-touching components is a fundamental requirement of the majority of recognition algorithms. The algorithms are based around the centre of area which defines a unique position on the component that is independent of the angle of orientation. For this point to be found accurately it is essential that the component be separate from its neighbours. Any system that can cope with touching components must therefore dispense with the centre of area and instead find some new constant on which to base second generation algorithms.

At present there is no robot manufacturer which makes its own vision system and similarly no vision system manufacturer makes its own robot. The linking of a vision system with a robot therefore involves an electronic interface between the two and a considerable amount of cooperation and collaboration between the manufacturers.

Two stages of interface can exist between a vision system and a robot. With the first, both robot and vision system function autonomously with only very simple 'Yes', 'No' information being passed between the two. An example of this might be a pick and place task where a vision system constantly checks to see if a component has arrived at the pick up position. If the component has arrived correctly the vision system sends a 'Yes' to the waiting robot, which will then move in and pick up the part. If no component is in position, or if the component is damaged, or not in the correct orientation then the robot will be told 'No' and will take no further action until signalled to continue by the vision system.

The second stage of interface involves the communication of position and orientation information to the robot which then uses this information to control its movements. An example of this type of system is a vision system looking at parts beneath it on a conveyor belt. A number of different parts may be present on the belt at a given time and the position and orientation of each part is unknown. The robot tells the vision system which component (eg no 10) it wishes to pick up next. The vision system will then look at the conveyor belt until it recognises part number 10. When it does so it will compute the position of the part and orientation and then communicate this information to the robot. The robot then uses this information to adjust arm position and gripper rotation, allow an offset for the movement of the conveyor belt and then move in and grasp the part. The robot must also know the coordinates of the plane to which the vision system relates, and be able to work in world coordinates relative to this plane (ie transferred plane mode). At the present time only one commercially available combination of robot/vision system exists that is capable of this level of communication and that is the Unimation PUMA and the MIC vision system.

2.4.4 Existing First Generation Vision Systems

At present there are five vision systems known to the author which are available as commercial units. Their manufacturers are as follows:-

Machine Intelligence Corporation (MIC)	USA
Automatix	USA
Brown Beveri and Cie (BBC)	W Germany
ITTB	W Germany
Autoplace	USA

With the exception of the Autoplace Opto-Sense vision system, all are very standard in their capabilities. The only significant variations are the ease with which the system can be used, the speed of computation and price. Only Autoplace and MIC systems have been designed in close collaboration with a robot manufacturer. (In appendix 4 the features of these vision systems are summarised)

2.4.5 Current Second Generation Systems

Up to the end of 1980 only two systems could be used to partially satisfy the criteria for second generation vision systems with industrially acceptable computation times. These have been developed at the Lausanne Polytechnique in Switzerland and the General Motor Research Laboratories in the United States. The essential elements are common to both systems even though the software techniques differ considerably. The GM 'Model Based Vision System' appeared to be faster although the speed of computation for both systems is largely scene dependent, showing considerable variation between different scenes containing the same components. Both systems use the shape of the components' outline as the basis for recognition. The complexity of the system can be shown by the following example. General Motors use an IBM 370/168 computer to analyse a 256x256, 32 grey level image. The time taken to analyse different overlapping parts was 31.6 seconds[23]. The size of computer used and the computation speed are clearly not acceptable for widespread application.

2.4.6 Tactile Sensing

Tactile sensing is by no means perfected[19] and many reseachers are endevouring to produce cheap and efficient tactile sensors.

An interesting example of a compliant device for inseting a peg in a hole is examined in the next section.

2.4.6.1 A Compliant Device for Inserting a Peg in a Hole

The insertion of a peg in a hole is the final phase in the assembly of a peg and a block with a hole[11,20]. McCallion et al analyses the physical interaction between these two components during insertion, describes a simple fine-motion device which utilizes this interaction to insert pegs into closely-fitting holes, and discusses possible variations to the construction of the device.

The problem of placing a peg into a hole is a common problem in the assembly of mechanical components. It occurs when pistons are fitted into cylinders, bolts passed through unthreaded holes, bearings fitting into housings and so on.

In general, a peg-hole assembly involves four phases;

(i) pick-up phase: the two components to be assembled are picked up from bins, magazines, pallets, etc by some assembly machine;

(ii) transport phase: they are taken to an assembly station and brought into contact with each other.

(iii) fine-positioning phase: the initial misalignment between the components is reduced and they are driven inside the 'insertion funnel', a spational region defined by the geometry of the components.

(iv) insertion phase: the final misalignment is corrected and the components placed into their designed positions.

Current industrial robots can readily implement the first two phases. Where the robots are sufficiently accurate to transport the components directly into the insertion funnel, the separate fine positioning phase is eliminated. However, the final phase, due to the high degree of interaction between closely-fitting components during insertion, remains difficult and outside the scope of most available machines.

2.4.7 Man-Robot Voice Communication

It may be attractive to allow the human operator to use English in instructing a robot as to its ongoing work. Moreover, the robot which is likely to be highly sophisticated could, with justification, respond to the human voice with synthesized speech to explain its view of the work situation. Its speech might be used to explain internal ailments which need service attention. The technologies involved in speech recognition and in speech synthesis are growing in sophistication and decreasing in cost.

2.4.8 Total Self-Diagnostic Fault Tracing

Whatever the level of robot sophistication, it is crucial that the machine exhibit an on-the-job reliability competitive to that of human worker. Thus, the robot user must have a long Mean Time Between Failure and a short Mean Time To Repair. If the machine is, indeed, an elegant one, then repair will be intellectually demanding. What is needed and what will be provided is a self-diagnostic software package that pinpoints a deficiency under any failure condition and directs the human service staff in efficient methods for recuperating performance.

2.4.9 Inherent Safety (Asimov's Law of Robotics)

Asimov's Laws become more important as robots become more competent and as robots are utilized in more intimate relationships with other human workers. Safety must be inherent if robots and humans work shoulder-to-shoulder with the robots doing the drudgery and with the humans contributing the judgement. The development task is not easy, but fortunately it is also not impossible.

CHAPTER 3

SOFTWARE SURVEY

A computer controlled industrial robot provides significant advantages over conventional hard wired robot controls[27,62,67]. Having software control provides the means for tailoring a robot to meet the specifications of a particular application. Software features for aiding in program generation and modification include three coordinate systems for positioning the robot while teaching, on-line editing, and copying a data point (or an entire sequence of points).

Although programmable systems have emerged as an important class of machines, the development of complete software system for controlling and programming such machines has just started.

Computer control provides an industrial robot with a decision making link to the outside world and gives the robot the capability of logically deciding what to do based on external signals and of reacting immediately to interrupts activated by emergency conditions. Using its computer the robot can issue status notification through output signals and can even communicate over a serial line with another computer to retrieve data. In addition, an on-line computer provides the computational capability to solve coordinate transformations in real time (ie software interpolation) permitting a computed path control system. A computer control also simplifies the teaching task for an industrial robot.

Flexibility, generality, ease in reprogramming, documentability, are the most important advantages produced by the introduction of a software system. The certain shortcoming is that it is hard to express by a formal language the human expertise of performing tasks.

Many researchers have been primarily directed to general and complex problems, while relatively little attention has been paid to questions about computational cost and programming difficulty, questions of great importance for industrial applications.

The decreasing cost of computer components and the widespread introduction of microcomputers in industrial equipment makes possible a new era in programmable industrial manipulation.

In industrial applications, robots are commonly programmed by guiding the mechanical device through a sequence of operations required to perform the assembly process. A joystick or a button box is used to insert in the control memory the positions that must be remembered. The position sequence may be played back to cause the arm to accomplish the task ie "teach" mode or "tape recorder" mode. According to R Taylor (Stanford University) this method is called "non-textural" to make it clear that programming a robot by the teach mode does not require a program. Any user, without specific training, may program the robot; this method does not require the user to associate abstract symbols with manipulator movements.

Nevertheless, many disadvantages cannot be avoided. The execution of the task is obtained playing a fixed sequence of movements, and the impossibility of expressing conditional actions makes it impossible to use force sensors or to introduce some adaptation, while the lack of text produces the impossibility of maintaining, documenting and modifying the program.

The direction of improving that method of robot programming was investigated at Stanford Research Institute. More flexible systems were developed, and joystick, teletype, or voice translaters were employed for giving commands to the robot. This augmented teach mode demonstrates that interaction with the robot by means of symbolic commands allows more flexibility, although certain programming expertise is required.

The major advantages of a textural language are that the text can be read by people, can be saved in an understandable form, and can fit different situations.

Control structures allow branching and conditional activity. Interface with people allows editing, modifying and documenting program, and supplies facilities in programming.

The textual approach to robot programming introduces in robotics the philosophy and the experience of software systems design. New languages for robot programming are necessary, because general purpose languages are generally not adequate.

Software design can be considered in two main categories. The first is explicit-programming which makes the user responsible for everything and requires explicit instructions for every action the robot must take. The second philosophy, called world-modeling, tries to make the robot responsible for taking some decision according to its knowledge.

3.1 HEIRARCHY OF SOFTWARE STRUCTURE

To enable the software to be more easily understood, a possible hierarchy which can be utilised is one which consists of five levels (figure 3.1). The lowest three levels are directly concerned with the robot, while the last two are concerned with the robot's immediate and global environment.

The lowest level in the hierarchy is where servo control functions are computed[28]. The input commands are joint positions which are compared to the feedback from the joint position sensors. If these values are different, a drive signal is generated to move each joint until the position error is nulled. The commands to the second level of the control system are calls to primitive function subroutines. 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 current sequence of joint position outputs to the next lower level (servo level) to accomplish this operation. Programming at this second level is enhanced over the first level since coordinate transformations are now possible with a computer. Thus, the arm can be commanded in terms of X,Y,Z coordinate space through the use of a joystick. The coordinate transformation routine calculates all of the joint motions required to cause the robot's hand to move along the

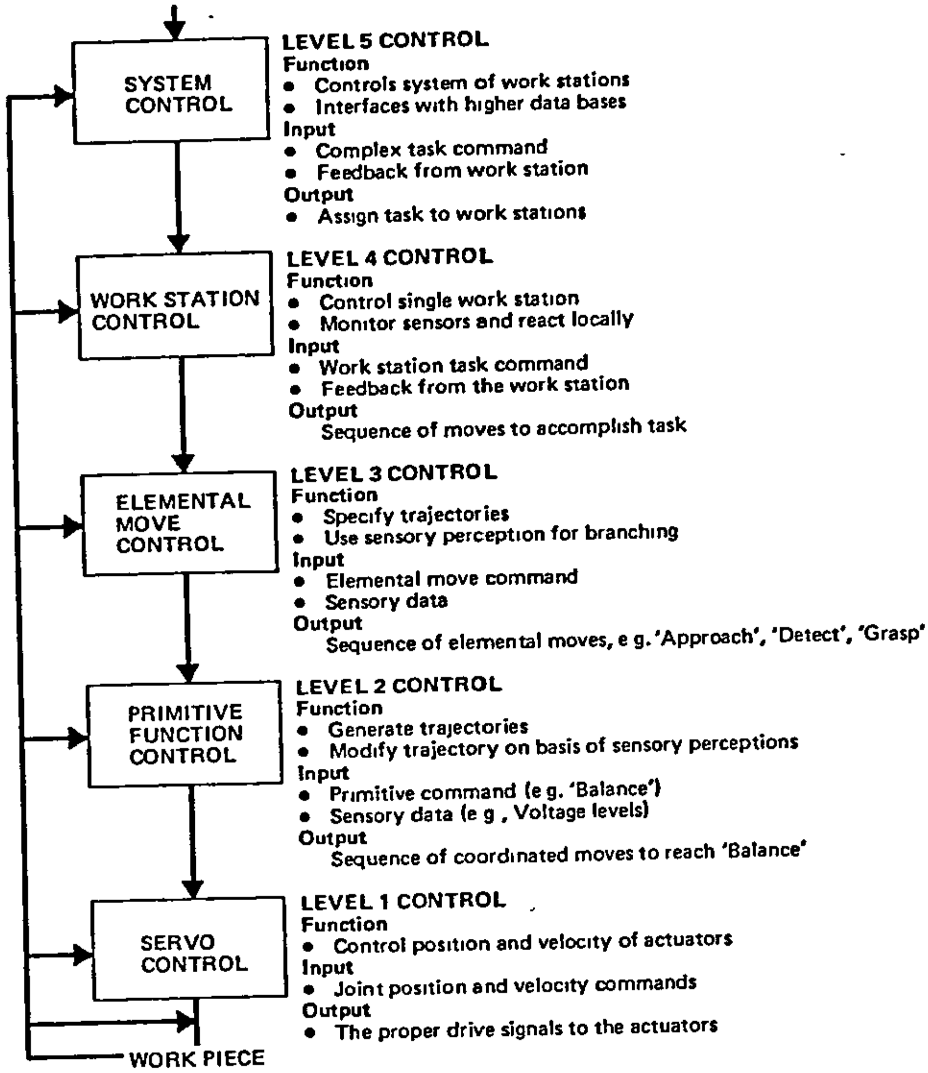


Figure 3.1 Hierarchical control system for robot installation

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 illustrates the power of an hierarchy as, when higher levels are added, the input commands become simpler and more procedure oriented. The sequences of detailed operations required to accomplish the tasks are generated by the lower levels in response to these commands.

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.

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 operation. An example of an elemental move command would be "GO TO 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 operations. 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 a specified location point still have to be recorded in a table of points. However, these points can be entered under joystick control or as X,Y,Z coordinates of the locations. Once a location is stored under 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 are entered.

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. This results in a separation of the description of the task as much as possible from the particular robot that may carry out its operation.

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

The fifth level of control is the "system control" and 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 the 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.

One of the advantages of hierarchical control is that complexity at any level in the hierarchy can be held within manageable limits irrespective of the complexity of the entire structure. However, such a hierarchical decomposition extends far beyond programming convenience. The real-time use of sensory measurement information for coping with uncertainty and recovering from errors requires that sensory data be able to interact with the control system at many different levels with many different constraints on speed and timing. For example, joint position, velocity and sometimes force measurements are required at the lowest level in the hierarchy for servo feedback. This data requires very little processing, but must be supplied without time delays of more than a few milliseconds. Visual depth (proximity) and information related to edges and surfaces are needed at the primitive function level of the hierarchy to compute offsets for gripping points. This data must be supplied within a few tenths of a second. Recognition of part position and orientation requires more processing and is needed at the elemental move level where the time constraints are of the order of seconds. Recognition of parts and/or relationships between parts which may take several seconds is required for conditional branching at the single work station level. Attempting to deal with this full range of sensory feedback in all of its possible combinations at a single level would lead to extremely complex programs. A sensory hierarchical can also be utilised as illustrated in figure 3.2.

The sensory processing hierarchy receives the raw sensory data at its lowest level. Each ascending level processes this feedback further, relaying the appropriate processed data to the corresponding level in the parallel control hierarchy. The sensory processing hierarchy also receives input at various levels from the control hierarchy. This input defines the type of sensory processing to be performed and the expected results. There is, therefore, a two way exchange of information between these two hierarchies at all levels.

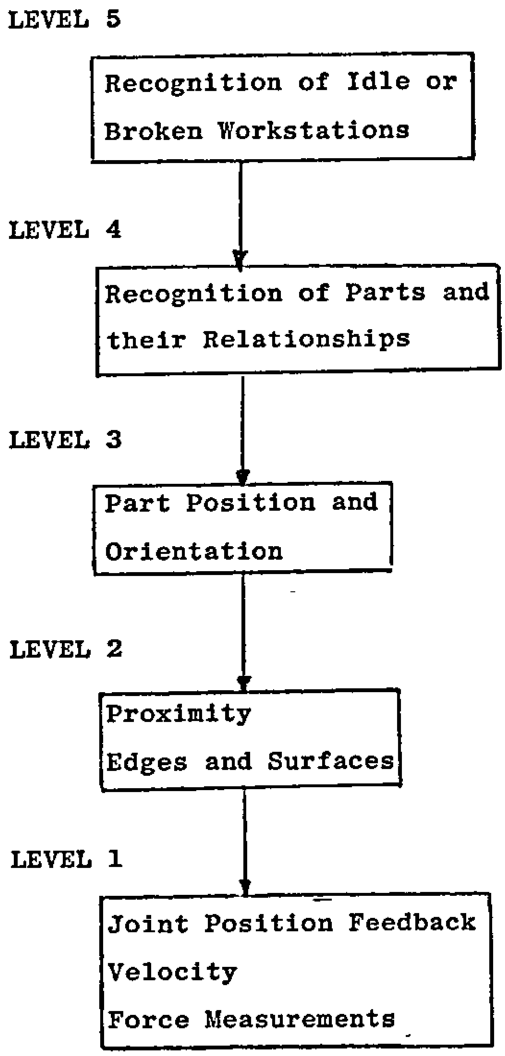


Figure 3.2 Hierarchical control system for sensory information

3.2 SPECIAL ROBOT LANGUAGES

Robot languages[32] have been developed by various organisations for the following reasons:- to facilitate the ease of programming for complex tasks; to minimise the "teaching" time especially for small batches and so increase flexibility; to link robots to CAD and CAM.

The robot languages include:-

1 Multipurpose Assembly Language (MAL)[33] which was designed and implemented at Milan Polytechnic for the Supersigma robot.

2 WAVE which was developed at Stanford Artificial Intelligence Laboratory (SAIL) and was the first flexible system for developing complex manipulation algorithms[34].

3 AL which is a world-modeling robot language is being developed by the Computer Integrated Assembly Systems project at SAIL[35].

4 Cincinnati Milacron Inc utilise an explicit program to control their robots[43,44].

5 VAL which was designed for use with Unimation Inc industrial robots

6 LAMA is a world-modelling mechanical assembly language which is being developed at MIT Artificial Intelligence Laboratory[37,17].

7 AUTOPASS is a world-modelling programming language used by IBM[37,38].

8 EMILY also used by IBM is an explicit language[39].

9 SIGLA is an explicit programming language used by the Olivetti corporation of Italy for controlling the SIGMA robot[40].

10 In the Department of Artificial Intelligence at the University of Edinburgh a mixture of explicit programming and world-modelling philosophies have been used to control their robot (which is called FREDDY)[41].

11 The National Bureau of Standards (NBS) uses a Cerbellar Model Articulation Controller (CAMAC)[42].

12 Perceptronics Inc uses an Adaptive Control System to control a silent anthropormorphic arm powered by 1000 psi.

13 System for Aiding Man Machine Interaction Evaluation (SAMMIE) which has been developed by the Production Engineering and Management Department at Nottingham University and is used as a basic modelling system for the simulation of industrial robots[45,46].

14 Graphical Representation Assessment and Simulation Package (GRASP) is currently being developed at Nottingham University.

15 DEA of Torino in Italy has applied to assemble part-programming a High-level Expansible Language for Programming (HELP)[47] which is used for the control of the PRAGMA 3000 robots.

16 RAPT has been partially developed at Edinburgh University and it is a geometrical expert.

17 PARAPIC which is also being developed at Edinburgh. It is a high-level language for parallel picture processing.

In the following sections some of the more important robot languages are considered in further detail with the emphasis on MAL, WAVE, AL and Cincinnati Languages.

3.2.1 MAL

MAL is an interactive system, which allows the user to describe the sequence of steps necessary to realize assembly tasks. It allows the independent programming of different tasks and provides semaphors for synchronization. A MAL system is completely implemented in Fortran IV except for a small interface to the robot, which is written in assembler, due to the demand for portability. Moreover MAL is implemented in such a way that the change of the controlled robot would not require a complete rewriting of the system. For example, the conversion from a cartesian robot to a polar one should require changes only to a given module.

The MAL system is made up by two different parts, one devoted to the compilation of the input language into an internal form, the other one devoted to the execution.

The compilation part gives the user facilities to create, update and maintain the source program. The execution part has the responsibility of executing the sequence of operations described in the user program. The debugging of the program is easy and the user can modify his program and immediately check it again.

To develop a program the user has to express his assembly task as a sequence of elementary operations. If the task requires some parallel activities, the programmer writes the different parts as they are independent and then synchronizes them. Then the MAL compiler translates the program into an easily interpretable object code.

3.2.2 WAVE

WAVE was developed at SAIL to show the feasibility of doing different tasks with the same robot system and it has been used for assembling a hinge, a water pump, a pencil sharpener, and other objects by using two arms and simple power tools. The facility of scene analysis programs permits verification of successful completion of individual actions. Interactive debugging facilities permit quick development of programs to do new tasks, although execution times are two to four times longer than a person would need. Although WAVE is an explicit-programming system, it maintains a world model of the arm for planning purposes. To write an arm-control program one first defines macros that expand into sequences of arm-control instructions to perform simple actions like screwing on a nut or picking up a screwdriver. Planning dictates the use of macros rather than subroutines because each call generates much information that depends upon the arm position. Nesting and parameters are allowed, and individual macros may be expanded, tested and revised with essentially a very small elapse of time. Once the macros perform as expected, the task program is written in the form of another macro, that is a sequence of calls to the previously-defined macros. WAVE lacks certain debugging facilities such as single-step execution, breakpoints and hot editing. So, although WAVE is convenient to program in and is interactive, it behaves like a system with a compiled user program.

3.2.3 AL

AL is a high level language programming system for specification of manipulatory tasks such as assembly of an object from parts. AL includes an ALGOL-like source language, a translator for converting programs into runable code, and a runtime system for controlling manipulators and other devices. The system includes advanced features for describing the motions of manipulators, for using sensory information, and for describing assembly algorithms in terms of common domain-specific primitives. The principal aim of the work carried out at SAIL is not to provide a factory floor programming system but rather to design a language which will be a tool for investigating the difficulty, necessary programming time and feasibility of writing programs to control assembly operations.

The supervisory software is the top level of AL. It runs on the timesharing computer and provides an interface between the user and the other parts of the system

i) listening to the user's console and interpreting simple command language input
ii) controlling the compiler, starting it and relaying its error messages back to the user
iii) signalling the loader when it is necessary to place compiled code into the mini.
iv) handling the runtime interface to the mini.

AL is important for several reasons which are :-

i) It shows what sort of considerations are necessary for the flexible control of mechanical manipulation.
ii) It demonstrates the feasibility of programmable assembly.
iii) It provides a research tool for investigation of new modes of

software servoing, assembly primitives, arm-control primitives and interactive real-time world systems.

AL is currently limited by the lack of certain features which would make it more useful. These features will now be described. Fine control of the arm could be enhanced by more sensitive force-sensing elements on the hand. Visual feedback should be implemented to provide better positioning capability, error detection, and error recovery. Moving assembly lines imply that AL should be able to understand motions which it does not cause directly through manipulation; objects should have a dynamic capability. Collision detection and avoidance remain difficult issues. AL would be more error-free if the trajectory calculator could ensure that the arms never interfere with each other or with objects in the current world.

3.2.4 Cincinnati Developments of Software

Cincinnati have developed software features to increase the flexibility of their robots which exploit the use of an on-line computer. These features are:-

i) Teach Coordinates

Using either a hand held Teach Pendant (figure 3.3) or a CRT and keyboard unit (figure 3.4). When in the teach mode the computer does not care how the tool centre point (TCP) is manipulated in getting from point to point in space. The only information to be retained is; the location, hand orientation and function data pertaining to each data point.

ii) Alignment Method in Software

The conveyor alignment (figure 3.5) method enables the user to adjust the X,Y,Z coordinate system so that it is parallel to X,Y,Z coordinate system of its working environment. This is especially useful when programming a tracking application since the tracking axis (the axis of conveyor motion) must be aligned to one of the major axes of the robot X,Y or Z. This method eliminates the precise positioning of the robot during installation by allowing the computer to adjust the robot to the conveyor. Another advantage of the alignment method is in

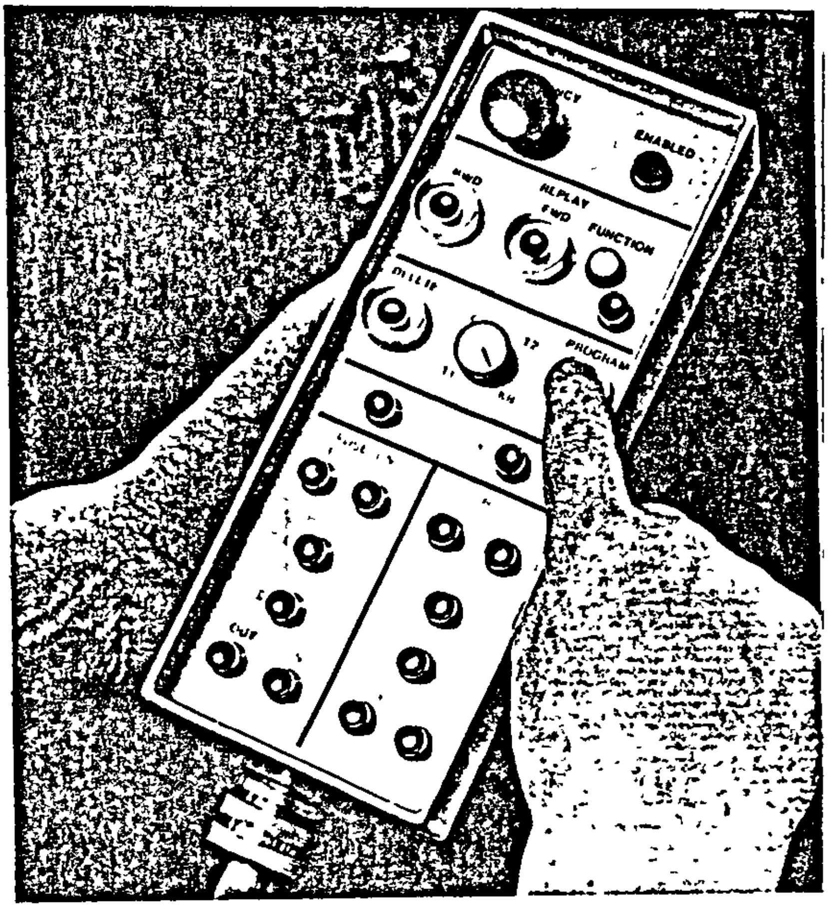


FIGURE 3.3 HAND HELD TEACH PENDANT

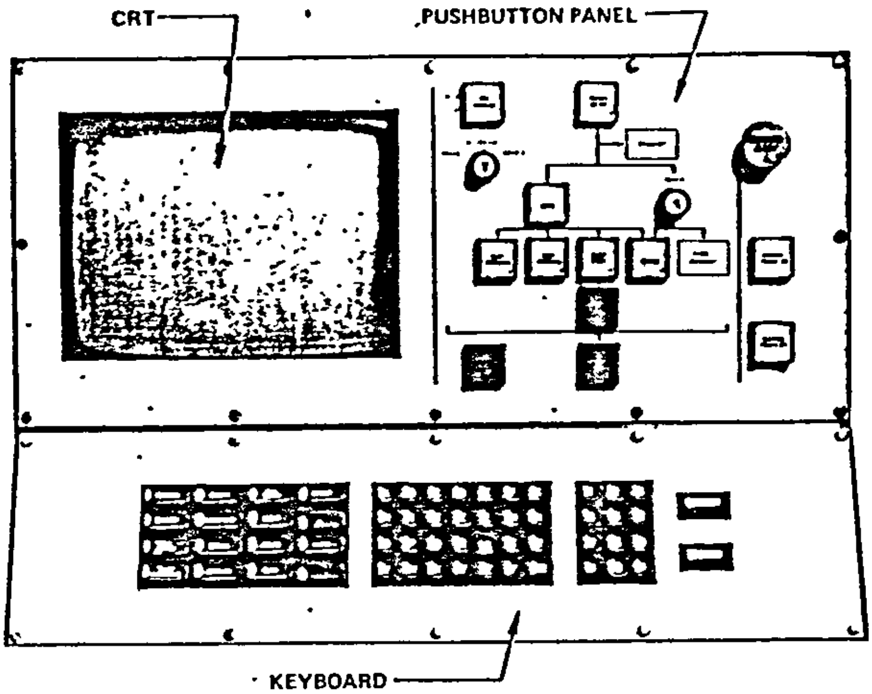


FIGURE 3.4 CRT AND KEYBOARD

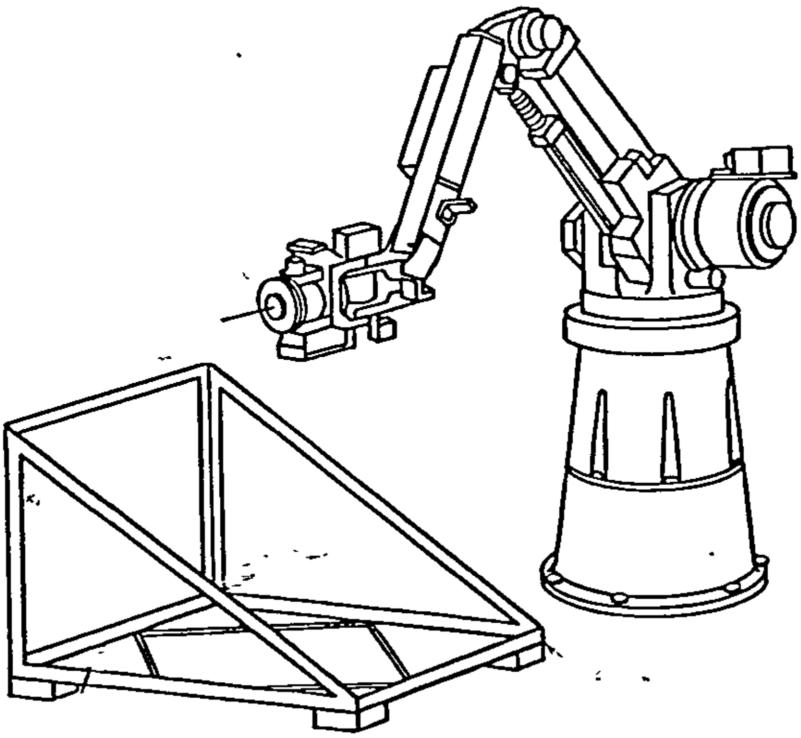


FIGURE 3.5 ALIGNMENT FIXTURE

multiple robot systems because it allows one robot to be used to create programs for all other robots. Whatever differences in alignment which might occur between robots is eliminated by this alignment method.

iii) Programmable System Generation

The programmable system is generated in the following way:-

a) The Programmable System Tape is loaded into the robot controller.
b) The user responds to messages on the CRT using the keyboard to define the parameters.
c) When all the parameters have been defined, the robot control will produce a final System Load Tape.

iv) Index Function

This is utilised, for example, to pick components off a pallet one at a time.

3.2.5 VAL

VAL runs on a LSI-11 and is used to control Unimation's Puma series of robots. On-line program generation and modification can be performed as the real time control and operator communication modules reside in the same computer. The language uses clear, concise and easily understood word and number sequences. It includes facilities such as subroutines, program branching and integer calculations, together with interrogation of and signalling to external devices via an input/output module. There are three coordinate systems which are available with the VAL operating system, which are available with the VAL operating system, which are joint, world and tool modes (figure 3.6). The difficult modes are selected by the operator as the robot is taught a task. VAL automatically takes these modes into account so that the task can be accomplished 'as taught'.

3.2.6 LAMA

LAMA will allow the user to describe an assembly procedure in the kind of English statements that may be used as captions under illustrations in a shop assembly manual. The system requires the user to decide the sequence in which the parts are brought together

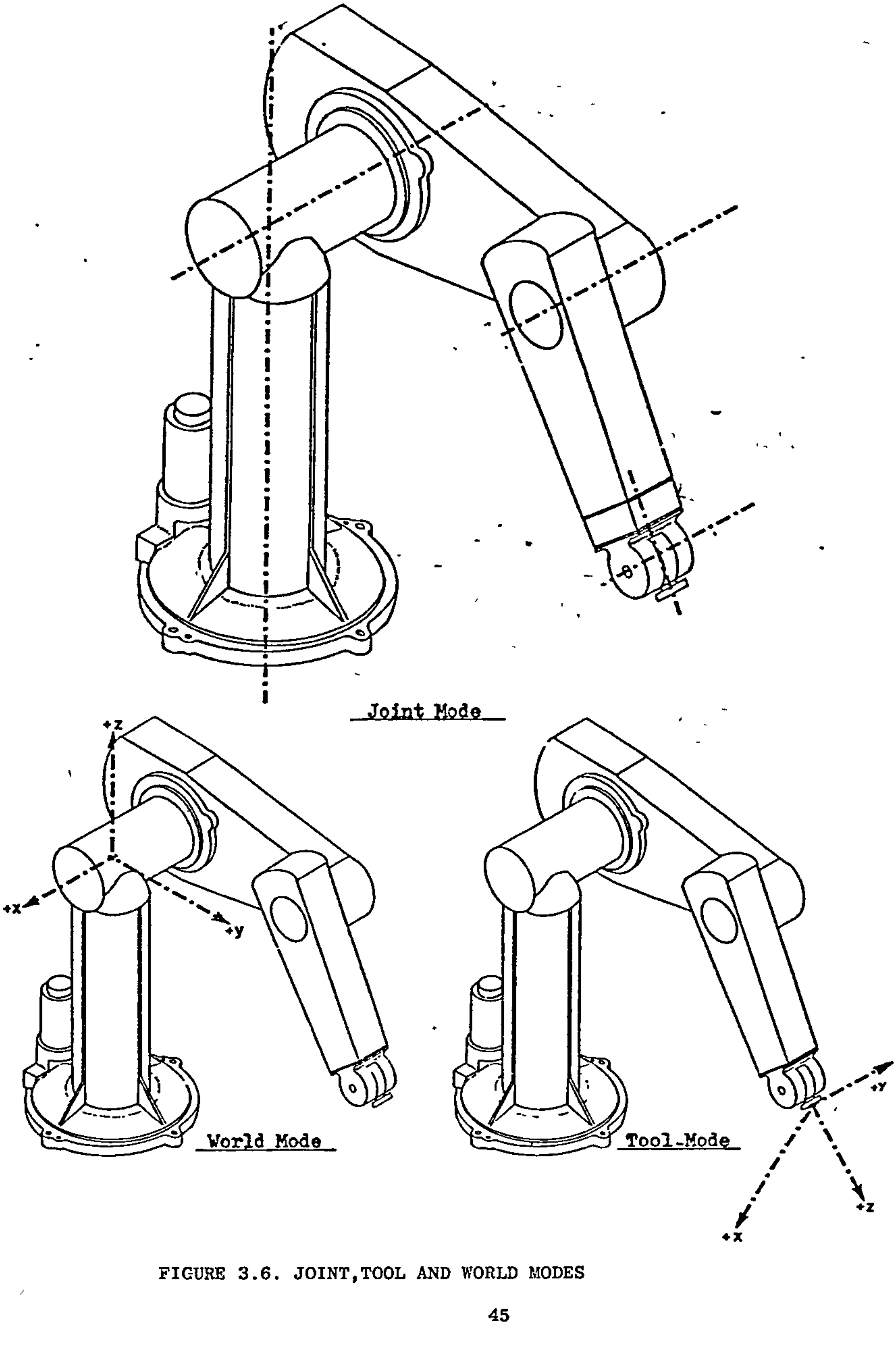


FIGURE 3.6. JOINT,TOOL AND WORLD MODES

then LAMA generates an appropriate sequence of pick-and-place motions and chooses grasp points on objects by itself. If also uses uncertainty and tolerance information in its world model to generate appropriate test procedures for every fabrication step. LAMA keeps geometric information in its world model and simulates the effect of candidate strategies by making temporary modifications to it.

3.2.7 AUTOPASS

The AUTOPASS user plans the part-attachment sequence, tool usage, and general object positions. AUTOPASS is designed for finding user errors at compile time rather than during execution. Graphic simulation substitutes for interactive debugging in trial runs. The AUTOPASS world model represents assemblies as a graph structure of object part, subcomponent, attachment, and constraint relationships. Basic entities on the graph are points, lines, and surfaces, and the user sets up the world model with a separate geometric design program. The statements deal with parts, tools, fasteners, and instructions for placement and attachment, and they are translated one at a time in a single pass. The compiler chooses optimal grasps for the user and plans hand trajectories to avoid collisions and to obey constraints on part motion. It originates some kinds of simple actions by itself, if necessary, to achieve preconditions needed to carry out a user statement. AUTOPASS statements translate into MAPLE-language statements containing the explicit planning decisions made by the AUTOPASS compiler. AUTOPASS and MAPLE are PL/I-like languages with extensions for manipulator-related language constructs.

3.2.8 EMILY

EMILY has control structures similar to FORTRAN, which can call upon more powerful user-written routines in the IBM 370/145. A program called Manipulator Operating System (MOS) moves the arm (or can operate two arms simultaneously) by interpreting the contents of tables, which have been produced by EMILY. Entries in one table describe the assembly algorithm in terms of pointers to MOS library routines and to other table entries to be passed as arguments to those routines. Debugging involves rewriting the individual library routine calls by using a metal language called ML and by changing the values of entries in the data tables.

3.2.9 SIGLA

SIGLA (SIGma LAnguage) controls SIGMA robots by symbolic commands, which are later interpreted as calls on library routines. The user moves the hands with a rate joystick during training. Routines for computation and sensing wrist displacement permit a variety of programmed search, patterning and accommodation behaviour.

3.2.10 FREDDY

FREDDY is a sophisticated vision-controlled robot. Using overhead and oblique cameras, Freddy looks at three-dimensional wooden parts poured onto his worktable in a heap. It breaks up the heap to see individual parts, then sorts out the good parts, discarding any it does not recognise. FREDDY uses a world model containing information about part images and object locations to recognise the objects placed under his camera and to find space in which to work. The recognition and sorting-out phase is programmed by showing the robot examples of every part in each of its stable postures on the table and by leading it through a pick-and-place sequence. During execution. Freddy generalises these motions to deal with parts in the same posture but with different positions and orientations. It assembles the parts under the control of an explicit, compiled POP-2 program written and debugged interactively for each particular assembly task. Freddy's library contains routines for constrained moves and insertions. The system software has a main loop that repeatedly classifies the current situation into one of eight distinct categories and takes the appropriate action for that category. For example, if it finds a complete kit of parts, it assembles them, if the kit is incomplete, it looks for more parts.

3.2.11 CAMAC

CAMAC is an algorithm and data structure that can learn to compute extremely complicated functions of many variables which run on PDP 11. It has been tried as a fast servo computation mechanism to determine, for example, the required joint motor drive current from the desired joint velocity and the actual position and velocity of all joints.

3.2.12 Adaptive Control System (ACS)

Perceptronics' Adaptive Control System (ACS) learns a manipulation task by monitoring a person operating a teleoperator master arm. It takes control of the slave arm when it has enough confidence in its ability to predict what the man will do next. ACS uses statistical decision theory to adapt a set of Bayesian arm-action probabilities. Perceptronics have also developed several kinds of three dimensional graphic displays and have used them for simulation.

3.2.13 System for Aiding Man Machine Interaction Evaluation

SAMMIE which is written in Fortran, simulates various industrial robots and provides an aid to robot selection, and a quick evaluation of robot/machine/robot interactions. The simulation is in two parts: firstly it is a planning phase where the workplace layout and robot manipulator articulations are examined - this can be considered as static analysis. The second phase is an execution phase, or a dynamical analysis: here robot times are produced and compared. The planning or programming phase is usually first, unless a programmed sequence already exists, and then information satisfying a static analysis is then used for a dynamic analysis. In the event of certain criteria not being satisfied in the dynamic analysis then perhaps it may be necessary to reprogramme the sequence and modify the layout. The ability to communicate with the computer with a light pen and a simple instruction set displayed together with computer generated pictures on the graphics terminal allows a number of iterations to be carried out quickly to obtain optimum results.

3.2.14 GRASP (Graphical Robot Assessment and Simulation Package)

This is currently in the early stages of research and development. The system will ease the introduction and application of present day robot technology into the manufacturing environment. The completed system will enable the engineer to produce three-dimensional models of robots and workplace similar to that used by SAMMIE. The actions of the robot are then specified by a means analogous to those programming real robots, although a high-level task specification language is planned as an additional aid. The simulation, used in conjunction with models of the workplace, equipment and machines, will enable assessments to be made of the suitability of particular industrial robots for the proposed task. Finally it is intended that the knowledge gained from the simulation will be used to program the robot itself in the same way as tapes are produced for Numerically Controlled machines.

3.2.15 HELP

The HELP language has been implemented on computers of the DEC 11 family. HELP language has two phases, one in which the translator acts as a pure compiler which alternates with the other in which the translated program is executed. Such phase alternate inside each single program element, it means that each single element is translated and then executed which results in a high interactivity level. Due to the compiler structure of HELP, the execution of the program can be postponed or the translated program can be stored for as long as desired for further recalling. The translation structure allows dialogue with the system during program set-up; it also yields reasonable execution efficiency as the programs which are available in memory are close-to-the-machine internal langauge. Externally the language is much like the ones of the Algol family, its elements are sentences or sentence blocks. The language has some macro-definition and macro-calling devices that make programming more intuitive and easier for the end user.

3.2.16 RAPT

RAPT at present uses a textural input with an APT like syntax. There are various descriptions utilised which are; objects - named features, for example plane faces; situations - described in terms of spatial relationships between features of objects; actions - descriptions in terms of movements of bodies relative to features, action statements are used like situation statements to form equations; ties - between bodies; subassemblies of bodies - partially assembled objects, residual degrees of freedom and mechanisms, for example vices.

All the languages which have been described are only applicable to either one robot and/or one computer system.

Robot software can be divided into two basic categories, on-line and off-line. The on-line software system controls the robot at run time (real-time control) whereas the off-line software generates the instructions required by the on-line software (operator communicator). The linking of the off-line and on-line systems varies for each language, at present there is no standard interface between the on-line and off-line systems.

Most of the languages are designed for assembly work and the only ones in use commercially are at the manipulation level. The high world model level languages are still under development in academic establishments. With the manipulation languages,.the on-line and off-line components of the system are run on the same computer, which is the robot controller. The higher world model languages, the off-line processing is carried out on a larger more powerful computer, which generates an intermediate language which can be input to the on-line system of the robot controller.

There is an immediate requirement for standards to be defined especially for this intermediate language so that one off-line system can interface with a wide variety of robot controllers and vice versa.

	Interactive	Language	Operations	Explicit	World	Uses Sensory Information	Calculates Trajectories
MAL	yes	FORTRAN IV interface in assembler	assembly	yes			
VAL	yes	ALGOL based	manipulation	yes	maintains world model of arm for planning purposes	yes	yes
WAVE	yes	ALGOL textural	assembly	yes	maintains world model of arm for planning purposes		
AL	yes	ALGOL textural	assembly			yes	yes
LAMA	yes	English statements	pick & place		yes		
EMILY	yes	Fortran	assembly	yes			
SIGLA	yes	symbolic commands		yes		yes	yes
ACS			manipulation by learning from a teleoperator arm				
AUTOPASS	yes	PL/I	finds user errors at compile time translates into MAPLE		yes	yes	yes
CINCINATI	yes	ALGOL	welding assembly	yes		yes	yes
HELP	yes	ALGOL	assembly	yes			
RAPT	yes	APT	assembly		yes	yes	yes

Table 3.1 Summary of Robot Languages

CHAPTER 4

THE COMPUTER, MICROCOMPUTER, TMS 9900 MICROPROCESSOR AND SOFTWARE/HARDWARE DEVELOPMENT SYSTEMS

In this chapter the evolution of the computer and the microcomputer is outlined. The hardware/software development aids are then considered which is followed by a description of the processing element of the controller, the TMS 9900 microprocessor.

4.1.1 Historical Development of the Computer

The abacus, a frame with wires along which beads can be slid, is well known in the West as a child's toy and an educational aid. In the East it is still extensively used for arithmetical calculations. Since each digit is separately represented (in this case by a bead) it is a digital machine[48].

An early analogue machine, and, until about 1970 the machine most widely used for multiplication and division, was the slide-rule. In analogue machines, numbers are expressed by the measure of some physical quantity (in a slide-rule the physical quantity is length which is made proportional to the logarithm of the number). As microprocessors and microcomputers operate digitally, analogue machines will not be considered further[49,50].

From the middle of the nineteenth century onwards various manually operated adding machines, including cash registers, became commercially available. These were followed in the first half of the twentieth century by desk-top machines that could divide and multiply. At first these were manually operated but as electricity became generally available some were powered, as were adding machines, by electric motors or actuators. Although such machines were able to store their results and often to print them out, each new entry had to be entered digit-by-digit by pressing the appropriate key or by adjusting the appropriate pointer.

The advent of punched cards at the turn of the century made it possible for the same data to be entered automatically for different calculations, for entries to be sorted into any required order and for the results of one calculation to be stored on new cards ready for later calculations.

Sensing of the holes in the punched cards was at first mechanical and calculation was undertaken by a system of mechanical linkages, cranks, rotating wheels and sliding bars. Later, electrical sensing of the cards was introduced and many of the mechanical links were replaced with electromechanical elements. In the main, operations were limited to addition, subtraction, sorting and tabulation. Multiplication and division were either impracticable or much slower than the normal operating speed of the rest of the equipment which usually handled two cards per second. In the 1930's however, multipliers consisting of banks of 'telephone type' relays were developed that enabled multiplication and division to be carried out within the cycle time of the rest of the equipment[51].

Probably the first digital electronic computer was built at the British Post Office Research Station during the Second World War. It was a special purpose machine dedicated to speeding up the deciphering of intercepted German signals (for security reasons, the existence of this machine was not announced for over 30 years and very few details have been published).

The war also produced ENIAC[51], probably the first true electronic digital calculating machine, built at the University of Pennsylvania, which was designed specifically for ballistic calculations which can be described in the following manner.

(i) it occupied a room approximately 12m x 6m
(ii) it contained nearly 18,000 thermionic valves
(iii) its power consumption was 150kW
(iv) it operated on numbers with ten decimal digits
(v) addition could be carried out at the rate of 5,000 calcaulations per second, multiplication at 350 per second and division at 166 per second.
(vi) it was able to store up to 20 different numbers and recall them immediately when required.

ENIAC was shortly followed by EDVAC, the first electronic machine to use binary arithmetic. It operated on binary numbers of 43 digits (equivalent to about 13 decimal digits) and could store over 1000 numbers for immediate recall. It was also the first machine to use an external store (using magnetic recording) to which it had automatic, but comparativelt slow access.

The success and publicity attached to these two US machines led to worldwide activity, at first in universities and military establishments where cost was not usually the prime consideration, and later in commerce and industry where the machines were expected to pay their way but probably seldom did. The machines of the mid 1950's cost about £100,000 for the computer and probably about half as much again for the air-conditioned room that was necessary to dissipate the heat from the electronic valves.

4.1.1 The Impact of Transistors

Transistors were invented in 1948 and 10 years later began to replace valves. Simultaneous developments in the design of immediate access memory stores enabled general purpose computers to be produced at a price which gave a reasonable chance of a satisfactory return on investment. By 1960 they were also of a reasonable physical size, 2 or $3m^3$ for the heart of the machine, the central processing unit (CPU) and the immediate access memory store, with a power consumption of 1 or 2kW (thereby much reducing heat dissipation problems). Despite inflation, prices had halved in 5 years for machines of similar computing power and were to do so twice more in the next decade.

To understand what happened in the 1960's which led directly to the advent of microprocessors, it is necessary to look briefly at the technology of the transistor. The actual diameter and height of the body are each about 5mm, anything smaller would be difficult to handle in an electronics assembly factory. The same size would protect a silicon chip probably smaller than 0.5mm square and 0.15mm thick (much smaller than a pinhead) so that the package is 2500 times as large as the contents. In turn the chip is much larger than is technically necessary because an assembly worker cannot handle chips much smaller than 0.5mm square. The active part of the transistor occupies less than 10% of this area so it is quite feasible to form two or more transistors on the same chip. In fact the chips are actually manufactured by forming many hundreds of them on a slice of silica (nowadays 60mm or more in diameter) and then cutting the slice into chips of the desired size. When it was realised that other circuit elements, (resistors, capacitors and interconnecting 'wiring') could also be built on the chip, the door was open for manufacturing more and more complex circuits on the chips.

Integrated circuits (ICs), were produced and the period from 1961 to 1972 saw the development of small scale integration (SSI) through medium scale integration (MSI) to large scale integration (LSI). These terms are not precisely defined but in general an SSI chip has tens of transistors with their associated circuit components, an MSI chip has hundreds and an LSI chip thousands. Vary large scale integration (VLSI) techniques which have tens of thousands of transistors have been developed which has significantly decreased the cost of the hardware so enabling a wide range of machines and processes to be controlled due to the low cost of the hardware.

However, at this time only limited memory and Input/Output (I/O) facilities can be configured on a single chip and "controller chips" are therefore used mainly in high volume applications which are generally of low complexity. For the majority of applications today the CPU will be a VLSI or LSI chip with powerful computing facilities and support VLSI (or LSI) memory, I/O, buffer, decode etc chips will be used to constructure a controller with memory and I/O facilities which meet the required specification.

4.2 HISTORICAL VIEW OF THE MICROCOMPUTER

The term micro in this connection first appeared in 1972 when the Intel Corporation produced the 4004 microcomputer. The heart of this system was an LSI package that included, on a single chip, all the features normally encountered in the central processing unit (CPU) of a mainframe or minicomputer. This IC was therefore given the name of a microprocessor or microprocessing unit (MPU)[69]. The principal elements of any digital computing system are the CPU and the immediate access memory. In terms of operating speed and other performance citeria, the 4004 fell far short of available minicomputers. There are however, applications for which large numbers and high speeds are unnecessary and for which low cost makes the Intel 4004 and other MPUs with restricted 'computing power' eminently suitable. The race for larger capacity and increased speed was on and during the next three years, half-a-dozen manufacturers developed single chip MPUs with capacities and speeds approaching those of the CPUs of some minicomputers. If present trends continue, the distinction between microcomputers and minicomputers, which is already becoming blurred, may eventually disappear.

The chip for a modern MPU is approximately 5mm square and contains many thousands of transistors and their associated wiring. It can perform all the functions of the CPU of a typical machine of the early 1960's often at comparable or faster speeds, with a power consumption of less than 150mW, (one-ten-thousandth that of the 1960 machines). The 5mm square chip has to be packaged in such a way that it can be handled and connected to the other system components. Since 40 separate lead-outs are required it is necessary to have a package several times larger than the chip itself. This dual-in-line package,.a reference of the two rows of connecting pins that can be inserted and soldered into a printed circuit board, is of the general configuration used in most ICs. It is also referred to as a 'DIL'.

In its package an MPU occupies about one-five-thousandth of the space occupied by the comparable CPU of a 1960 machine. Price, too, has come down by a factor which, allowing for inflation, is of the same order as the reduction in size and power requirements.

4.3 PROPERTIES OF HARDWARE AND SOFTWARE FOR THE IMPLEMENTATION OF DATA PROCESSING ALGORITHMS

In table 4.1 there is a comparison between the use of hardware and software for the implementation of algorithms

HARDWARE	SOFTWARE
Designed by an engineer, who must know the physical limitations (fan-out, propagation delay, heat dissipation etc) of the components in use	Designed by a programmer, who must understand his machine as an abstract mathematical object with formal properties, but needs no detailed knowledge of the hardware
Capable of fast operation if necessary	Speed limited by (a) algorithm (b) design of computer
Design methods uses finite state machines and logic diagrams	Design methods use flow charts, mnemonic codes etc

Table 4.1 Comparison between the use of hardware and software for implementing algorithms

In comparing the two methods of implementing algorithms, software is relatively slow in performing operations when compared with hardware, but it is very much faster to design and write, and it can handle more complex algorithms. This seems to make it preferable in most circumstances.

4.4 HARDWARE/SOFTWARE TOOLS FOR MICROPROCESSORS

4.4.1 Hardware Aids

Development of microcomputer-based products usually requires a support computer system. Products are seldom developed in the computer environment that will surround the eventual software. More often, a separate computer system is used to support software development efforts and augment hardware testing. This is in sharp contrast with most minicomputer applications in which the mini is used for both development and production. Similarly, digital design engineers will find that checking out a microcomputer prototype requires more support than traditional TTL or CMOS design efforts.

4.4.2 Small Support Environments

The most primitive kind of support system is based on the actual microcomputer being developed, or on another small system. Often, this small configuration is actually nothing but an evaluation kit from the semiconductor maker. The kit provides a micro, a small amount of data storage space and a debugging monitor in ROM or EPROM. The devices supported for I/O may be on-board or outboard. The on-board peripherals usually include a keyboard and some seven-segment LED displays used for hexadecimal data and address bus contents. Outboard devices are usually assumed to be teletypewriter terminals.

These small systems are inexpensive but the features they provide are very limited. The limited program and data storage sizes of many evaluation kits prevent the use of an assembler. Programming must therefore be done in machine code, although usually hexadecimal or octal code representations are used as determined by the features of a debugging monitor which acts as a limited function operating system. The interactive ability of most debug monitors may consist of little more than loading, executing, modifying and dumping a small program from memory. The small read-write memory sizes included are typically less than 1024 bytes which only permits the smallest of programs to be entered and tested without appreciably increasing memory size. Memory expansion may be complicated by the lack of space on the evaluation kit's printed circuit card and the lack of external bus buffering necessary to communicate with an outboard memory card. These products were designed for small evaluation exercises, not wholesale program development[48].

4.4.3 Simple Software Support

Even the evaluation kits have some software for aiding the debugging process. However, they nearly all require recourse to some computer terminal unless the kit's own hexadecimal keyboard and displays are used. If the internal "terminal" is used, operations are usually very limited and error-prone. If the computer's debug monitor permits the reading of programs from cassette tape (usually from an audio recorder), the lack of an external terminal may be mollified.

The debug monitor consists of some simple input/output (I/O) software, plus a command interpreter to allow loading, modifying, executing and dumping memory contents. The more sophisticated monitors include the ability to perform hex-to-decimal conversions, insert software breakpoints which, when reached, cause control to return to the monitor, and the ability to display CPU register contents of programs that are being debugged.

4.4.4 Software Development Systems

Traditionally, microcomputer software design has been supported on large-scale computers, either through time-sharing or under the aegis of an operating system on an available computer, or on a microcomputer development system.

The simplest development systems are made up of a microprocessor, some limited input/output capability to a terminal, and a large amount of memory. Microcomputer development systems typically provide from 16K to 64K bytes of read-write storage, plus a small monitor. A high-speed printer and floppy disc can be added.

4.4.5 In-Circuit Emulation

In-circuit emulation is a concept pioneered by Intel, and is a circuit, that plugs into a socket replacing the CPU chip.

An in-circuit emulator allows the host computer, with all of its additional memory and monitor software and peripherals, to become a resource to support the operation of the system under test. In the emulation mode, the operator can remove control from the executing program by using monitor commands. The suspended program's register and memory contents can be examined, modified and execution resumed. As there are no changes in the system under test except microprocessor replacement, all of this testing capability is transparent to the design. This means that the system under test can be a nearly completely finished system. The in-circuit emulator can be utilised to exorcise the last residual design "blunders".

Until 1977, virtually all microcomputer development systems were dedicated to supporting one particular vendor's microprocessors. However, Tektronix have designed a development system which supports the Z-80, 6800 80808 8085. In Table 4.2 there is a survey of the available microcomputer development systems.

4.4.6 Logic Analyzers

The basic principle behind Logic Analyzers is illustrated in figure 4.1. The probes bring in signals from the equipment under development. The signals are matched with switches on the front panel so that some combinations of them can be used to enable the memory to record all the inputs.. The front-panel controls are usually toggle switches that can be set to recognize each input at 1 or 0, or to ignore that particular input. When the input conditions match the conditions specified in the switches, the memory is enabled. The inputs are collected in memory, one for each clock pulse. As each new word is written, the oldest work is purged from memory and produces the information on an internal (or, sometimes, external) oscilloscope.

The input probes are usually connected to bits of the address bus, and some of the control bus signals. (It is often useful to have come record of the data bus contents too). The number of input signals that can be sensed and recorded in parallel which range from 8-40 bit memories is an important parameter of these devices.

4.4.7 Software

Programming languages can be divided into three categories, high level, low level and machine code.

High level languages (HLLs) offer two important advantages over machine and assembly codes. In the first place, the programmer is freed from having to remember the precise arbitrary details of the target machine which is being utilised, and so can concentrate on the problem which is to be solved. This makes programming easier and up to 10 times faster. HLLs are therefore called 'Problem-Orientated Languages'.

Systems and support

System	CPUs supported	Support hardware	High-level languages
Advanced Micro Computer AmSYS 8/8	Z80, 8080 8085, Z8000	Z8000 prototyping board with in-circuit emulation capability	Pascal (all CPUs) and Basic, Fortran Cobol for 8 bit processors
American Microsystems Inc. MDC 1000	S2000, 6800	DEV 2000 emulator 68000 prototyping boards, EPROM programmer, MDC-140 logic analyzer	Basic, Pascal
Fairchild Microflame development system	9400 Microflame I, 9445 Microflame II 16 bit CPUs	PROM/FPLA programmer	Basic, Fortran Pascal
GenRad/Futuredata Advanced development system	808x, Z80 680x, 1802, 387X, 8048	Emulators logic analyzer, PROM programmer	Basic (compiler and interpreter versions) Pascal
Hughes Semiconductor Products HMDS 2D	1802, Z80 8080, 8085	In-circuit emulation for all CPUs, and PROM programmer, light pen	Basic compiler for 1802
Intel Corp Intellec Series II Model 240	808x, 802x, 804x, 8035	In-circuit emulation for all CPUs except 8088, multi ICE (two CPUs emulated simultaneously) and PROM programmer	Basic, Fortran, Cobol (for all 8-bit processors)
Millennium Systems Inc Microsystems Designer Series 1000	Z80 8088 8086, Z8000	Microprocessor personality modules	Monitor software for program development, debug and system control
Mostek Corp AID 80F	Z80 (but can be set for 3870 CPU)	PROM programmer, in-circuit emulation	Fortran, Basic, PL/1S, Cobol
Motorola Semiconductor Products, Inc. EXORcisor II	6800 family and 68000 (all CPUs of company supported)	PROM programmer, 68000 prototyping board with emulation capability, ICE for all CPUs including 6809, system analyzer (for added hardware-debug capability)	Cobol, Basic Fortran, MPL for all CPUs except 6809 and 68000 which have Pascal
National Semiconductor Starplex	8080, 8085, 8048, 8049, Z80	PROM programmer, In-circuit emulation for all CPUs	Basic, Fortran
Rockwell International System 65	6500, 6500/1 single-chip computer	In-circuit emulation for 6500 and personality option for 6500/1, PROM programmer	PL/65
Solid State Scientific μMOS development system	1802	In-circuit emulation EPROM programmer	μ Forth
Tektronix 8002A Microprocessor Development Lab	8080/85, 6800, Z80, TMS 9900, 3870/72, 1802, 8048 family	In-circuit emulation, real-time trace, and PROM programmer	Basic and Fortran for 8080, 8085, and Z80
Texas Instruments FS 990 AMPL Microprocessor Prototyping Laboratory	All 9900 family CPUs	In-circuit emulation for all CPUs, logic- state trace, PROM programmer	AMPL (language) Pascal, Fortran
Zilog PDS 8000	Z8, Z80, Z8000	Z8000 development module, PROM/EPROM programmer, in-circuit emulation and logic analyzer capability	PLZ/ASM, Z8000 translator

Table 4.2

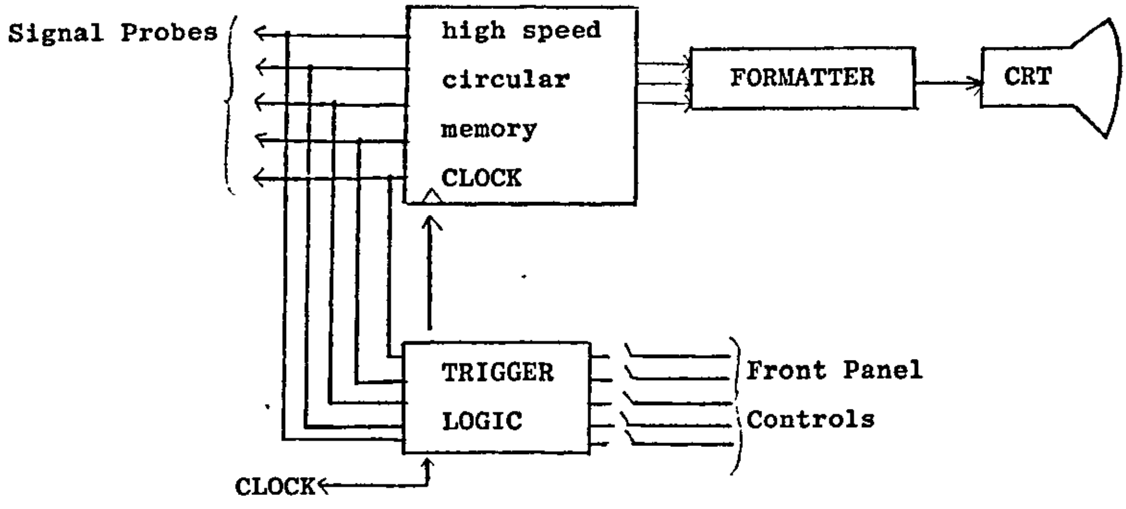


Figure 4.1 Organization of a Logic Analyser

Secondly, the algorithm is not related to any particular design of machine and can therefore run on any computer or microcomputer for which a suitable compiler exists. Such programs are called 'portable'. To offset these advantages, HLLs do have certain drawbacks. In general, the translation process is not perfectly efficient, and a HLL source usually runs slower and needs more storage space than a low level language (LLL). Once the source code, either HLL or LLL, has been produced, it is then assembled into machine code. This is the pattern of ones and zeros that actually drives the microprocessor. It is possible to write programs in machine code by hand assembling, but if there are more than a few lines, it is too time consuming.

4.5 THE TEXAS 9900 FAMILY

The 9900 family is a compatible set of LSI components including microprocessors, microcomputers, microcomputer modules and minicomputers. It is supported with peripheral devices development systems and software. The family features true software compatibility, I/O bus compatibility and price/performance ratios which encompass a wide range of applications[51-54].

4.5.1 The Hardware Family

Figure 4.2 is a diagram of the 9900 family members. The spectrum of microprocessors and microcomputer products available in a variety of formats as in figures 4.3 and 4.4. In the first part of figure 4.2 the microprocessors or microcomputers are combined with microcomputer support components (figure 4.4) to form systems. These systems also include I/O interface, read only and random access memory and additional support components such as timing circuits and expanded memory decode.

The family also includes microcomputer board modules containing the 9900 microprocessor and peripheral components (figure 4.5). As shown in the second part of figure 4.2, these modules can be used for product evaluation, combined for system development or applied directly as end equipment components.

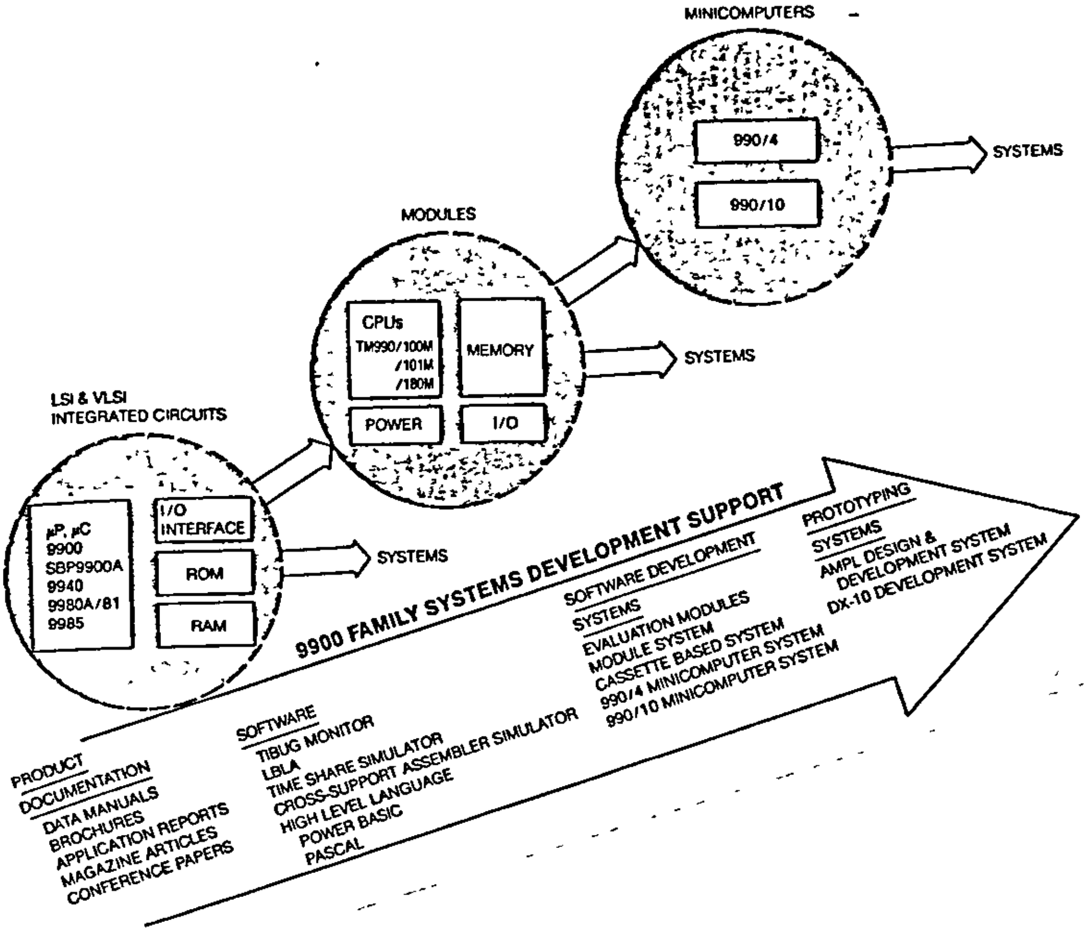


Figure 4.2 The 9900 Family

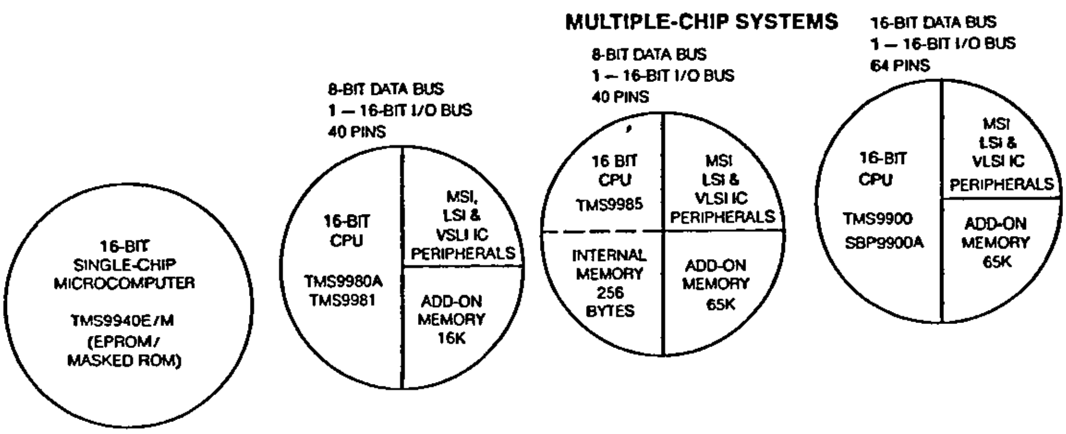


Figure 4.3 9900 Family CPUs

CPU's	
TMS9900	NMOS 16-Bit Microprocessor, 64 Pins
TMS9900-40	Higher Frequency Version 9900
SBP9900A	I²L Extended Temperature Range 9900
TMS9980A/ 9981	40-Pin, NMOS 16-Bit Microprocessor with 8-Bit Data Bus 9981 has XTAL Oscillator
TMS9985	40-Pin, NMOS 16-Bit Microprocessor with Single 5V Supply and 256-Bits of RAM
TMS9940E	40-Pin, NMOS Single Chip Microcomputer, EPROM Version
TMS9940M	40-Pin, NMOS Single Chip Microcomputer, Mask Version

PERIPHERAL DEVICES			
TMS9901	Programmable Systems Interface	TMS9914	GPIB Adapter
TMS9901-40	Higher Frequency Version of 9901	TMS9915	Dynamic RAM Controller Chip Set
TMS9902	Asynchronous Communications Controller	TMS9916	92K Magnetic Bubble Memory Controller
TMS9902-40	Higher Frequency Version of 9902	TMS9922	250K Magnetic Bubble Controller
TMS9903	Synchronous Communications Controller	TMS9923	250K Magnetic Bubble Controller
TMS9904	4-Phase Clock Driver	TMS9927	Video Timer/Controller
TMS9905	8 to 1 Multiplexer	TMS9932	Combination ROM/RAM Memory
TMS9906	8-Bit Latch	SBP9960	I/O Expander
TMS9907	8 to 3 Priority Encoder	SBP9961	Interrupt-Controller/Timer
TMS9908	8 to 3 Priority Encoder w/Tri-State Outputs	SBP9964	SBP9900A Timing Generator
TMS9909	Floppy Disk Controller	SBP9965	Peripheral Interface Adapter
TMS9911	Direct Memory Access Controller		

ADD-ON MEMORY

ROMS	EPROMS		DYNAMIC RAMS
TMS4700—1024 X 8	TMS2508	—1024 X 8	TMS4027—4096 X 1
*TMS4710—1024 X 8	TMS2708	—1024 X 8	TMS4050—4096 X 1
TMS4732—4096 X 8	TMS27L08	—1024 X 8	TMS4051—4096 X 1
SBP8316—2048 X 8	TMS2516	—2048 X 8	TMS4060—4096 X 1
SBP9818—2048 X 8	TMS2716	—2048 X 8	TMS4116—16,384 X 1
	TMS2532	—4096 X 8	TMS4164—65,536 X 1

*Character Generator—ASCII

**PROMS	STATIC RAMS			
SN74S287— 256 X 4	TMS4008	—1024 X 8	TMS4043-2	— 256 X 4
SN74S471— 256 X 8	TMS4016	—2048 X 8	TMS4044	—4096 X 1
SN74S472— 512 X 8	TMS4033	—1024 X 1	TMS40L44	—4096 X 1
SN74S474— 512 X 8	TMS4034	—1024 X 1	TMS4045	—1024 X 4
SN74S476—1024 X 4	TMS4035	—1024 X 1	TMS40L45	—1024 X 4
SN74S478—1024 X 8Δ	TMS4036-2	— 64 X 8	TMS4046	—4096 X 1
	TMS4039-2	— 256 X 4	TMS40L46	—4096 X 1
	TMS4042-2	— 256 X 4	TMS4047	—1024 X 4
			TMS40L47	—1024 X 4

ΔEquivalent to SN74S2708

**Also available in 54 series

Figure 4.4 Microcomputer Support Components

MICROCOMPUTER MODULES	
TM990/100M	Microcomputer, 1-4K EPROM
TM990/101M	Microcomputer, 1-4K ROM, 1K-2K RAM
TM990/101M-10	Microcomputer, 1-4K ROM, 1K-2K RAM, Evaluation POWER BASIC®
TM990/180	Microcomputer, (8-Bit Data Bus), 1-2K ROM, 256-1K RAM
TM990/189	Microcomputer, University Microcomputer Module
TM990/201	Memory Expansion Module, 4K-16K ROM, 2K-8K RAM
TM990/206	Memory Expansion Module, 4K-8K RAM
TM990/301	Microterminal
TM990/302	Software Development Module
TM990/310	I/O Expansion Module
TM990/401*	TIBUG® Monitor in EPROM
TM990/402*	Line-by-Line Assembler in EPROM
TM990/450*	Evaluation POWER BASIC® —8K Bytes in EPROM
TM990/451*	Development POWER BASIC—12K Bytes in EPROM
TM990/452*	Development POWER BASIC Software Enhancement—4K Bytes in EPROM
TM990/501-521	Chassis, Cable and Power Supply Accessories

Figure 4.5 TM990 Board Module and Software Support

CS990/4	• A 990/4 Minicomputer with 4K words of RAM • Expanded memory controller with 4K words of RAM • 733 ASR ROM Loader • 733 ASR Data Terminal • Necessary chassis, power supply, and packaging

FS990/4	• Model 990/4 Minicomputer with 48K bytes of parity memory in a 13-slot chassis with programmer panel and floppy disk loader/self-test ROM • Model 911 Video Display Terminal (1920 character) with dual port controller • Dual FD800 floppy disk drives • Attractive, office-style single-bay desk enclosure • Licensed TX990/TXDS Terminal Executive Development System Software with one-year software subscription service

FS990/10	• Model 990/10 Minicomputer with 64K bytes of error-correcting memory and mapping in a 13-slot chassis with programmer panel and floppy disk loader/self-test ROM • Model 911 Video Display Terminal (1920 character) with dual port controller • Dual FD800 floppy disk drives • Attractive, office-style single-bay desk enclosure • Licensed TX990/TXDS Terminal Executive Development System Software with one-year software subscription service

DS990/10	• Model 990/10 Minicomputer with mapping, 128K bytes of error-correcting memory in a 13-slot chassis with programmer panel and disk loader ROM • Model 911 Video Display Terminal (1920 character) with dual-port controller • Licensed copy of DX10 Operating System on compatible disk media, with one-year software subscription service • DS10 disk drive featuring 9 4M bytes of formatted mass storage, partitioned into one 4 7M-byte fixed disc and a 5440-type removable 4 7M-byte top-loading disk cartridge Options One additional DS10 disk drive with 9 4M bytes of formatted mass storage, in deskmount, rackmount, or quietized pedestal version

Figure 4.6 990 Minicomputers

When applications require minicomputers, completely assembled units can be utilised. An overview of minicomputers is given in figure 4.6. The software is fully compatible with any associated micorprocessor and microcomputer system.

These three levels of hardware - the TMS 9900 family parts, the TM 990 microcomputer modules and the 990 minicomputers - constitute the hardware family.

4.6 THE SOFTWARE AND DEVELOPMENT SYSTEMS SUPPORT

New products cannot be made without design, development, test and debug. Development support for all levels is shown in figure 4.2 including

A Products documentation
B Software
C Software development systems
D Prototyping systems

Figure 4.7 outlines the above.

4.7 THE MICROCOMPUTER

The microcomputer is a computer with a microprocessor as the central processing unit and various peripheral devices that complete the various requirements. (See Appendix 5 for details of various microprocessors). Microcomputers are often classified by the number of chips required to make up the computer, namely single chip, twin chip, single board, etc. The microcomputer can be placed in three broad categories :-

(i) Central Processing Unit (CPU)
(ii) Input/Output Facilities (I/O)
(iii) Data Storage/Memory

The general system configuration is shown in figure 4.8. The intelligence of the machine is provided by the software capacity. It is this software that provides the required outputs in response to given inputs.

PRODUCT DOCUMENTATION

- 9900 Family Systems Design and Data Book
- 9900 Software Design Handbook
- TM990 System Design Handbook
- 990 Computer Family Systems Handbook
- Product Data Manuals
- Product User's Guides
- Product Brochures
- Application Notes
- Application Sheets

SOFTWARE AND FIRMWARE

TM990/401	TIBUG Monitor in EPROM
TM990/402	Line-by-Line Assembler in EPROM
TMSW101MT	ANSI-Fortran Cross-Support Assembler, Simulator and ROM Utility
TM990/450	Evaluation POWER BASIC —8K Bytes in EPROM
TM990/451	Development POWER BASIC — 12K Bytes in EPROM
TM990/452	Development POWER BASIC Software Enhancement Package — 4K Bytes in EPROM
TMSW201F/D	Configurable POWER BASIC in FS990 Diskette
TMSW301F/D	TIPMX — TI PASCAL Executive Components Library

SOFTWARE DEVELOPMENT		SUPPORT SOFTWARE
TM990/302	Software Development Module	Edit, Assembler, Load, Debug, PROM Programming
TM990/40DS	Software Development system for TMS9940 Microcomputer	Assembler, Debug Monitor, Trial-in-System Emulator, PROM Programmer
CS990/4	Single User Software Development System (Cassette Based), uses PX990 software	Text Editor, Assembler, Linking Loader, Debug Monitor, PROM Programmer
FS990/4	Software Development system (Floppy Disk)	Source Editor, Assembler, Link Editor, PROM Programmer
FS990/10	Software Development System (Floppy Disk)	Same as 990/4, expandable to DS System
DS990/10	Disk Based 990/10 with Macro Assembler	Source Editor, Link Editor, Debug, Librarian, and High-Level Language such as FORTRAN, BASIC, PASCAL, and COBOL

MICROPROCESSOR PROTOTYPING LAB FOR DESIGN AND DEVELOPMENT

AMPL FS990 with video display and dual floppy diskettes includes TX990/TXDS system software — Text Editor, Assembler, and Link Utility — and has an in-circuit Emulator Module and a Logic-State Trace Module for proposed system emulation and analysis

TIMESHARE SYSTEMS

GE, NCSS, Tymeshare	Assembler, Simulator, ROM Utilities

Figure 4.7 The 9900 Family Software and Development Systems

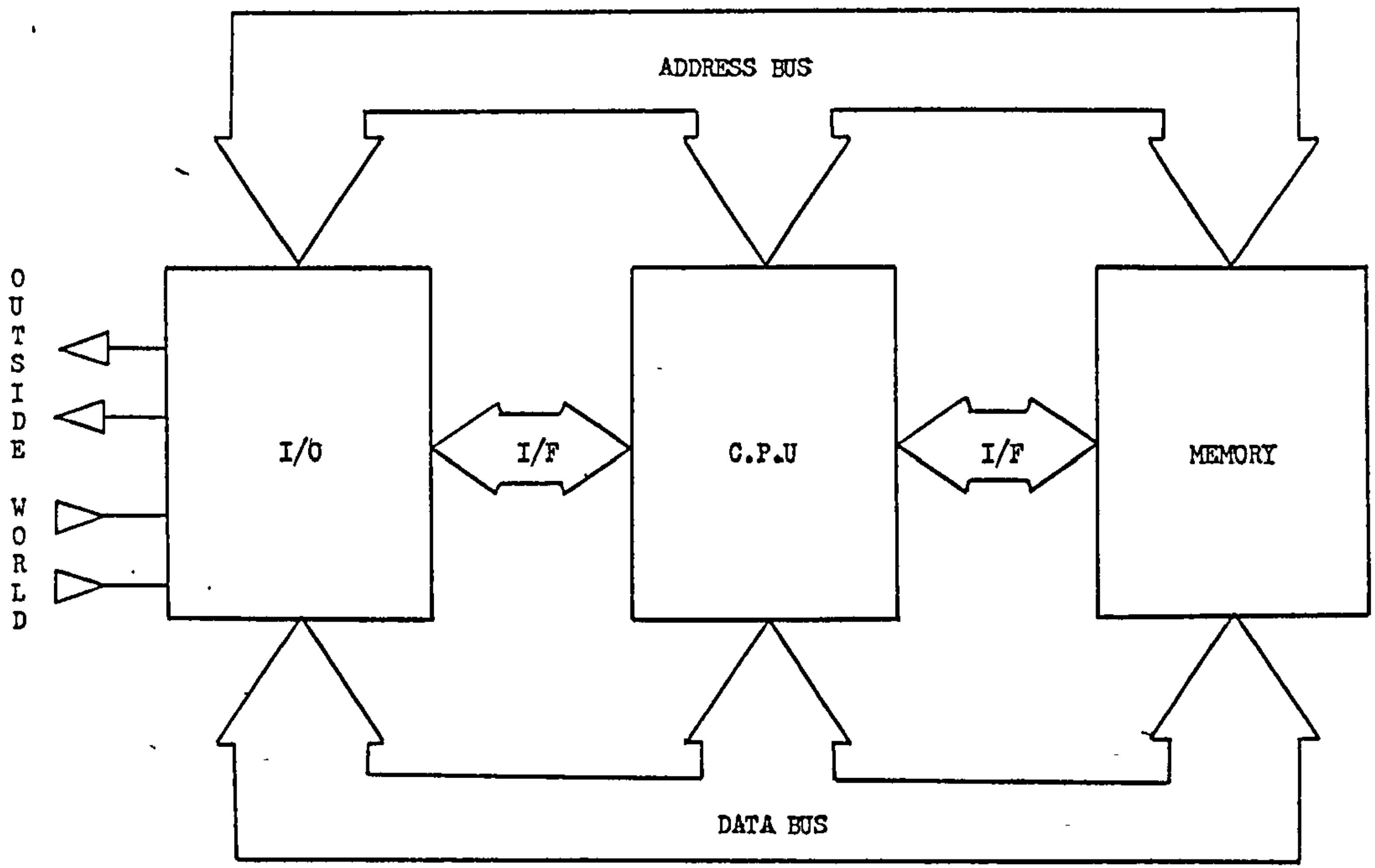


Figure 4.8 Microcomputer General Arrangement

The hardware of microcomputers is complex and varies according to the manufacturer, as does the software required to drive the various hardware.

4.8 HARDWARE

The hardware of a microcomputer essentially consists of three parts, as described earlier; the CPU, I/O facilities and data storage. It is now not uncommon for all these facilities to be available on a single chip microcomputer, but we shall concentrate on the single chip processing unit. The single chip processors can be put together in a variety of ways with other standard support chips, the configuration dependant on the application requirements and availability. The processing power of some microcomputers can approach that of the minicomputer.

4.8.1 The Central Processing Unit

The CPU has the capability of arithmetic and logic functions to process the information and data with which it is supplied. Dependent upon the design of the microprocessor, there is a set of instructions which the devices will recognise and respond to. It is the sequencing of these instructions that give the microprocessor the ability to carry out given tasks. The sequencing of these instructions is the 'software' of the processor. It is the software that makes the microprocessor flexible when compared to hardwired electronic devices. The software can be changed, modified and improved quite easily, the hardwired device being a more permanent and less flexible solution in many cases.

The software or program is stored in memory, with each instruction achieving a predetermined address. The exclusive addresses allow the CPU to communicate with any instruction held in memory by examining the address.

The interrogation of these instructions is performed within the CPU, and to achieve this it requires the following constituents shown in figure 4.9.

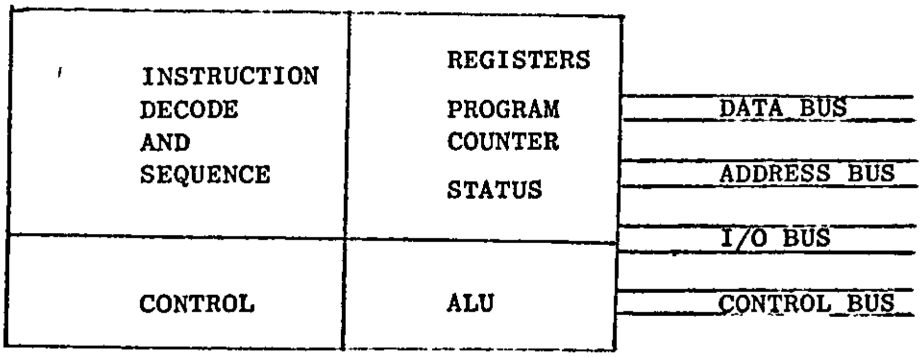


Figure 4.9 CPU Block Diagram

The Arithmetic Logic Unit (ALU) has two sets of data inputs and one set of outputs. It performs the logic and arithmetic functions on the input words and presents the results at the output. The control input determines what function the ALU performs at any given instance. The ALU is shown in figure 4.10.

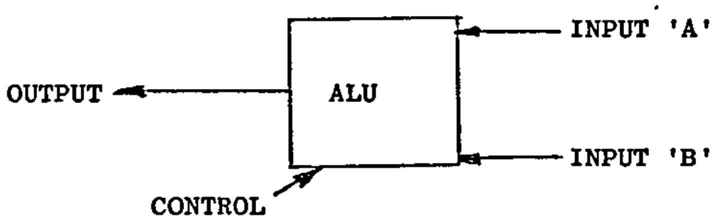


Figure 4.10

The functions of the ALU are listed as follows:-

AND	ADD
NAND	SUBTRACT
OR	INVERT A OR B
NOR	SHIFT L
EXCLUSIVE OR	SHIFT R

For any instruction supplied to the microprocessor there must be a facility to decode it and supply the relevant information to the ALU. This is done with the Instruction Decode and Sequencer.

The CPU has working storage registers. These are small amounts of dedicated memory with the CPU for immediate data storage. The status register primarily is set according to the results of the prior operation of the ALU. The program counter records the current location of the instruction sequence in memory.

The processor must communicate with memory, I/O devices etc. This is achieved via the data bus, address bus, control bus and I/O bus. These are multiwire 'highways' to the environment external to the CPU.

4.8.2 Memory Devices

There are two basic types of memory chips that are normally used in microprocessor systems, namely:-

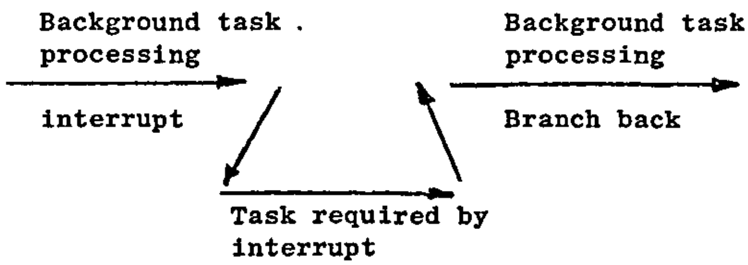
RAM - Random Access Memory
ROM - Read Only Memory

RAM devices allow data to be entered (WRITE) altered and retrieved (READ) at any time. They are volatile and hence if power is removed, the contents of the memory are lost. The RAM memory is normally assigned to user memory area.

ROM memory devices are non-volatile. Once the contents of the memory have been burnt into place, the contents as such are fixed. These devices can only be read from as the name implies. The inflexibility of these devices has led to the development of PROM (Programmable Read Only Memory) and EPROM (Erasable Programmable Read Only Memory) devices. The devices are used to store the operating system programs such as MONITORS, ASSEMBLERS, etc. In EPROM devices, the contents can be erased by exposure to ultra violet light and then reprogrammed as required, making the devices more versatile but relatively more expensive.

4.8.3 Input/Output Devices

The input/output section provides the communication between the microprocessor and the outside world. This can be achieved either by parallel data transfer, where more than one 'bit' of information if passed via the I/O bus in parallel, or by serial data transfer where one 'bit' at a time is passed. Each microprocessor family usually contains LSI devices designed to handle parallel and serial data transfer and to provide interrupt and timing controls. Interrupt controllers are used to signal the microprocessor at the instance of an external event which requires the microprocessor to perform a set of different instructions after completing the instruction which is currently being executed, this is diagramatically represented below in figure 4.11.



Timers and Event Controllers are devices which count clock pulses usually by decrementing a register. They can produce set delays or measure actual events and activate an interrupt to the microprocessor. Other LSI I/O devices include memory controllers, keyboard decoders analogue devices, display controllers etc.

CHAPTER 5

THE VERSATRAN ROBOT AND CONTROL SYSTEMS

The Versatran Robot was designed and made by Hawker Sidely Dynamics Limited. It derives its name from the Versatile Transfer operations which it performs.[58] The robot is capable of lifting, rotating and setting down components weighing up to 220 kg (1001 lb) anywhere within its sector of operation.

The mechanical unit consists of a rotatable column through which an arm passes. The arm has a wrist/gripper mechanism at its end. The arm is moved hydraulically, either by motor or ram in three major axes:-

(i) Horizontal (H)
(ii) Vertical (V)
(iii) Rotary/Swing (S)

In addition three other degrees of freedom are available: the wrist can be rotated and swept about the end of the arm, whilst the gripper can be opened or closed. There are various types of gripper that can be used, dependant upon the nature of the work being undertaken.

Each major axis of the arm forms part of an electro-hydraulic closed loop servo-system for which the command signals can be supplied by a microprocessor.

Each hydraulic circuit within these servo loops consists essentially of a reservoir, radiator, pump and accumulator, together with the arm swing servo-control valve, and all hydraulic components are positioned within the base of the unit except the hydraulic servo valves controlling the horizontal and vertical axes which are located at the top of the column.

The dimensional details of the Versatran Robot are shown if figure 5.1 and the locations of various components are shown in figure 5.2. For more comprehensive data on the robot, see Appendix 9.

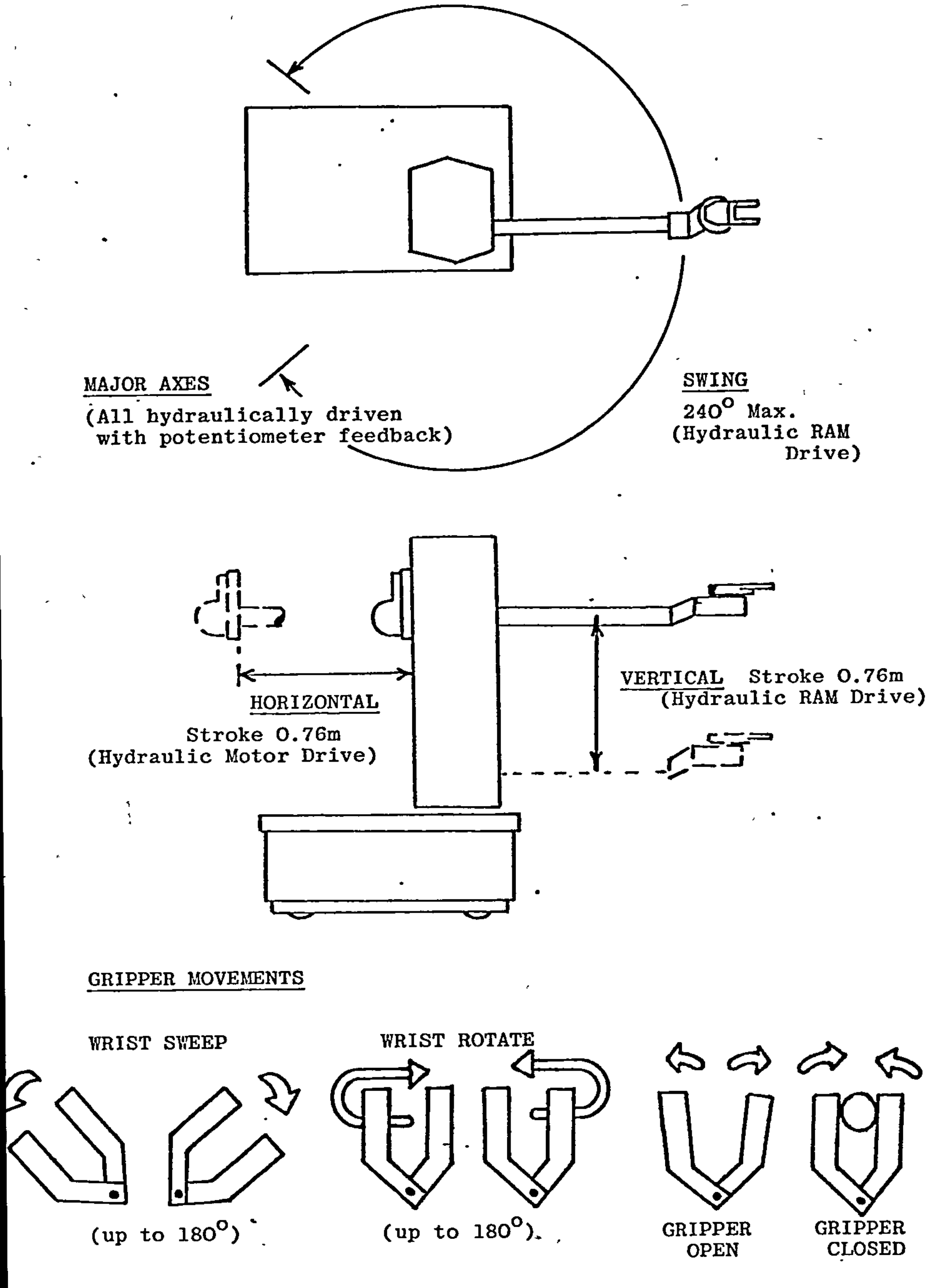


FIGURE 5.1. DIMENSIONS OF VERSATRAN ROBOT

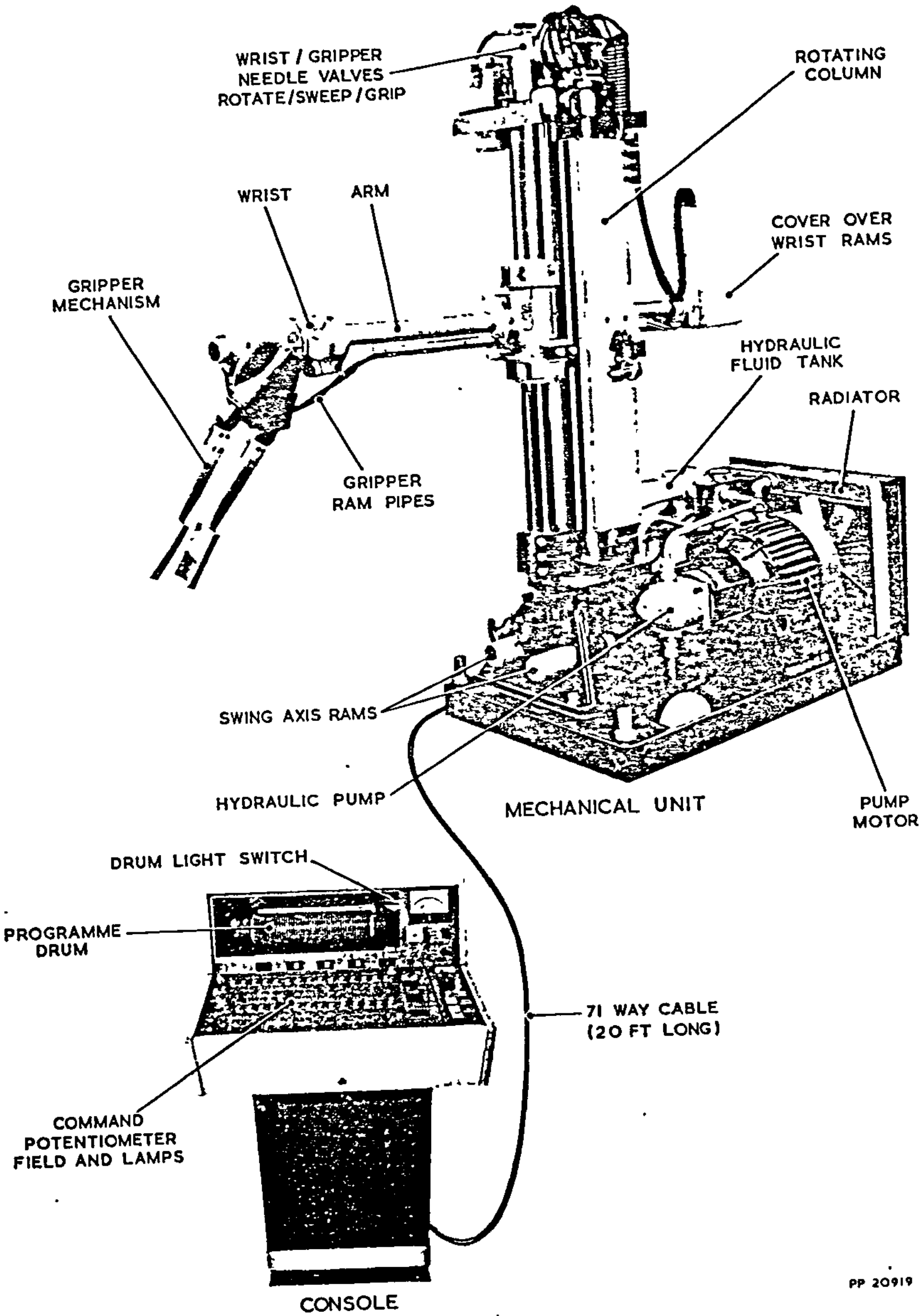


FIGURE 5.2 MECHANICAL UNIT AND CONSOLE

5.1 CONTROL OF THE VERSATRAN

At the beginning of this research project the Versatran was presented to the Department of Engineering Production and was controlled by a dedicated console. This console is drum operated and provides programme selection and control equipment including all electronic components associated with the axis servo systems.

A field of ninety command potentiometers, arranged in groups of three, allowed up to thirty discreet arm positions to be set up for an operational cycle. It was possible to select each position more than once, the total number of movements in any one cycle being one hundred. The group of potentiometers in use at any one time was selected by a 100 step, rotary program drum, this also operated the wrist and gripper mechanisms. The drum stepped from one command to the next when the arm reached its command position.

This method of programming worked successfully, but was difficult to carry out, (programming a new sequence of tasks could take many days), and placed many limitations on the robot.

Thus a decision was made to design a microprocessor base control system for the Versatran to act as a controller to replace the existing console, and so achieve 'state of the art' performance from the robot. This required the configuration of computer hardware and interface circuitry. Some of this hardware was purchased as off the shelf printed circuit boards although various interface circuitry was designed and constructed "in haste". The complete hardware was configured within a standard 19 inch racking system and backwired with associated power supply equipment.

A schematic representation of this hardware is shown in figure 5.3.

5.2 HYDRAULIC SYSTEM

The hydraulic power supply for the Versatran produces a pressurised fluid which, via servo valves, is directed to the various rams, jacks and motors that power the robot. The hydraulic fluid is pressurised to approximately 2840 kg/cm^2 (2001 lb/in^2) during operation. Included in the circuitry are three master solenoid

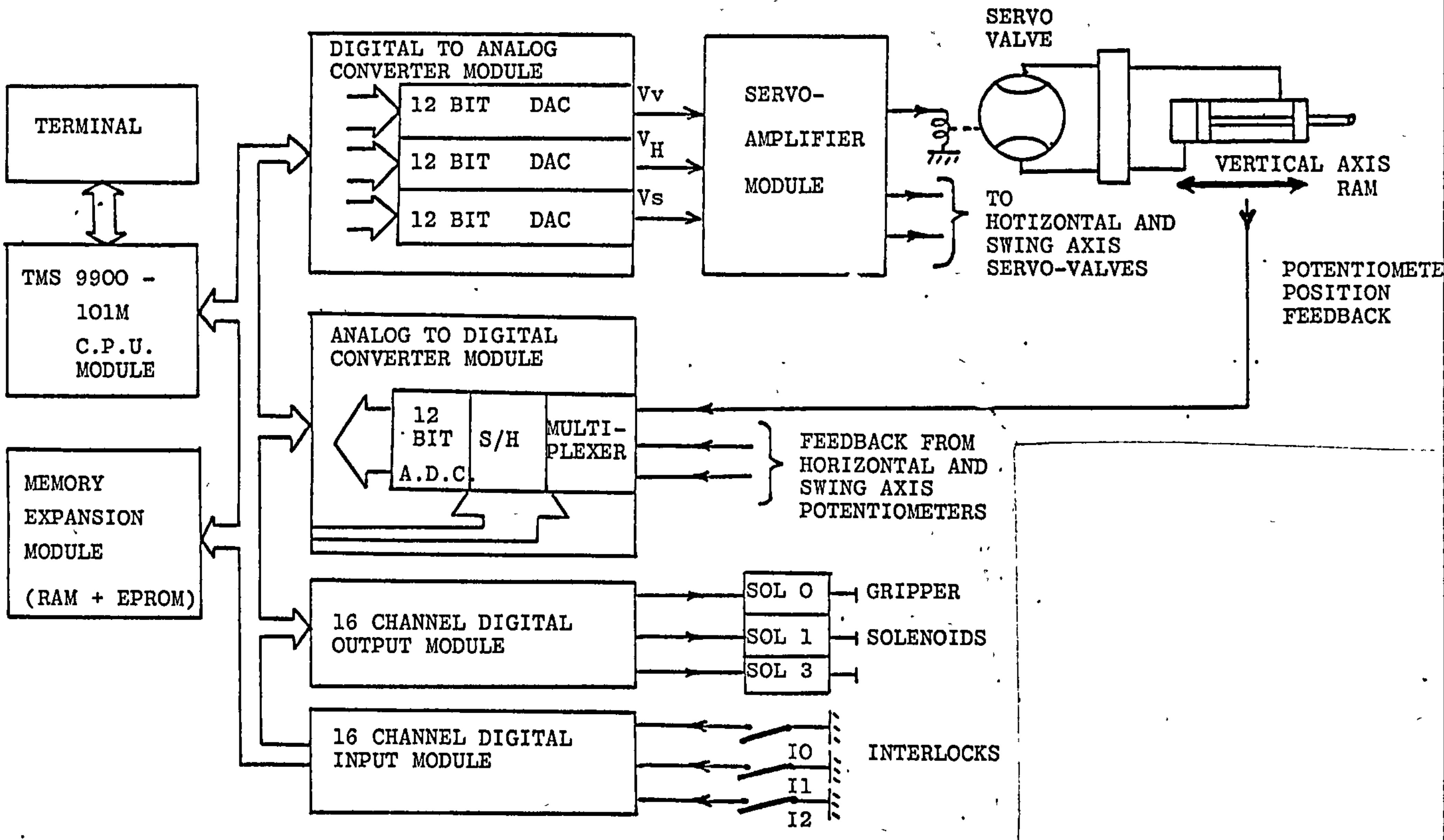


FIGURE 5.3. VERSATRAN INTERFACING SCHEME

operated lock valves that prevent any movement when they are closed. The hydraulic system will not operate successfully until it has reached a stabilised working temperature, which takes approximately fifteen minutes to attain once the robot is powered up.

5.3 SERVO-SYSTEM

Solenoid actuated servo valves control the three major axes and thus provide controlled fluid power to the robot via hydraulic rams on the swing and vertical axes and via a hydraulic motor on the horizontal axis. A schematic of the servo-valve system is shown in figure 5.4. The solenoids act against the reference springs to provide a valve displacement proportional to the input signal. A displacement of the valve from the null position causes fluid to flow to the piston, the amount by which the valves open is proportional to the current flowing through them. The rate at which the arm moves can be varied by altering the input signal in any one sense (positive or negative). A reverse voltage signal will create a movement in the opposite direction in the same way.

The Versatran feedback signal is obtained from three potentiomenters; one mounted on each axis of motion. Through these potentiometers the coordinates of any desired location can be described. With the console controller, the position to which each axis of the robot arm was driven, was proportional to the voltage of a pre-set command potentiometer. With the microprocessor control unit, the feedback signal is the three voltages which are dependent upon the arm position.

5.4 DESIGN OF THE HARDWARE FOR THE MICROPROCESSOR BASED CONTROLLER

The controller is based on the Texas Instruments TM 990/101 M self-contained microcomputer which is contained on a single printed-circuit board. A description of the board is given in Appendix 8. Interface printed circuit boards are also required, which include digital to analogue and analogue to digital converters, amplifiers to drive the servo valves and solenoid drivers using relays to control the wrist and grippers.

5.4.1.1 Introduction to the TMS9900 Microprocessor

The TMS9900 microprocessor is a single chip 16 bit central processing

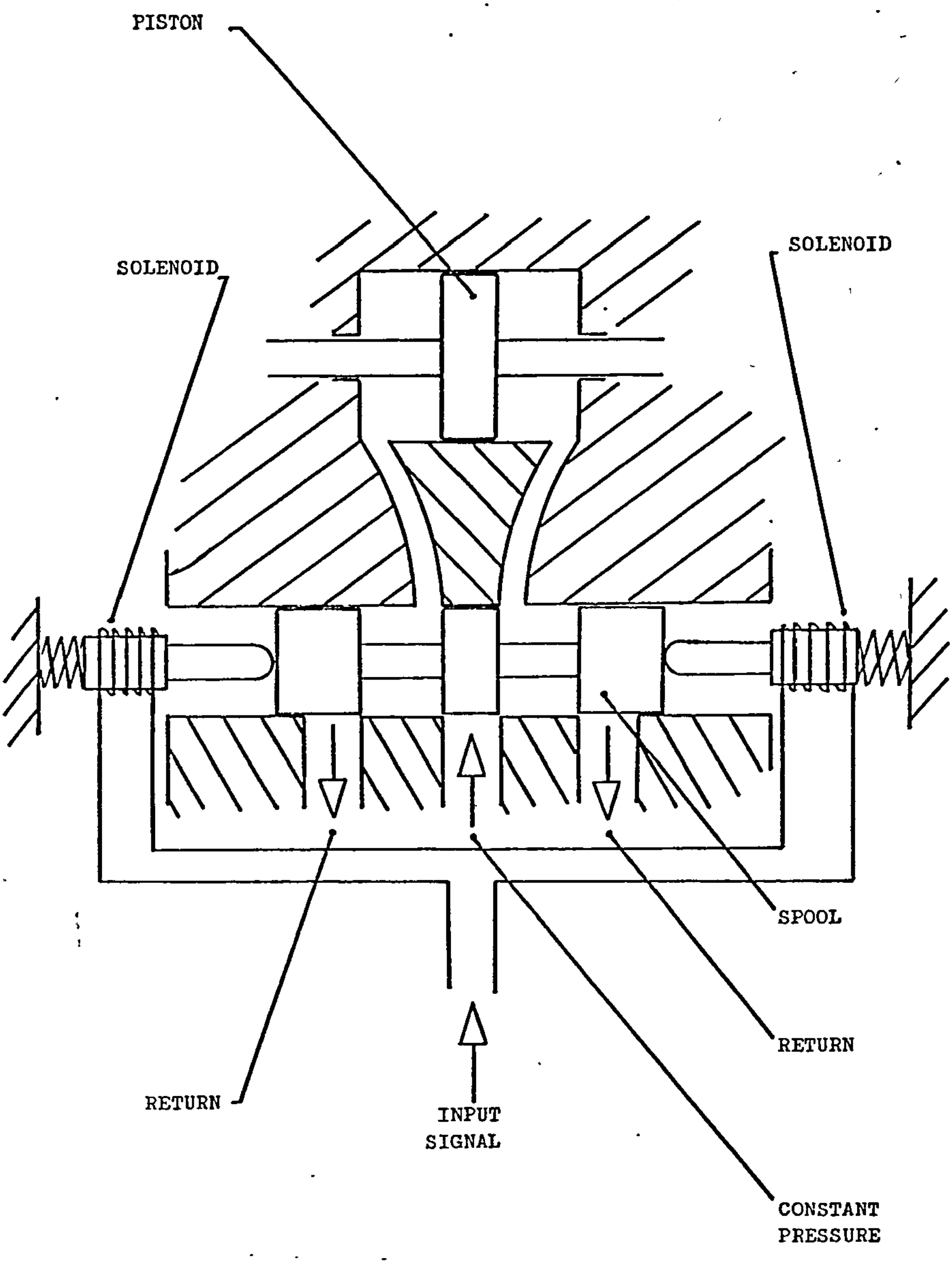


FIGURE 5.4. SERVO-VALVE CONFIGURATION

unit utilising n-channel silicon gate MOS (metal oxide semi-conductor) technology. The CPU communicates with memory devices, namely ROM, RAM, PROM, EPROM, etc, via a 16 bit bi-directional data highway. It also uses this data bus to communicate with external peripherals that are treated as memory locations.

Addressing is through the 15 bit address bus which gives the capacity to address 32K (32,768) words each being 16 bits wide, or a total of 64K (65,536) memory locations, each location being 16 bits wide. This allows either word or byte arithmetic and logic operations to be performed dependant upon the instruction used.

The TMS9900 Microprocessor does not have any on chip register file for handling working data storage. It utilises memory locations to store this data. Specific blocks of words are designated for this task. There are three user accessible registers in the CPU, namely the workspace register, program counter, and status register.

This context switch architecture of the TMS9900 is not common in microprocessors in that it uses an on chip workspace pointer, pointing to a set of workspace registers in memory rather than use on chip registers and a stack pointer. It utilises a system where the workspace pointer can be saved in any new workspace memory when a sub-routine is called. A current workspace is simply the 16 consecutive memory locations beginning at the address contained in the workspace pointer. The unique memory to memory architecture allows faster response to interrupts and increased flexibility in programming.

5.4.1.2 Programmable Systems Interface

The TMS9901 Programmable Systems Interface is a multifunctional chip designed to provide low cost interrupts and I/O ports in a 9900/9980 microprocessor system. It is fabricated with n-channel silicon gate technology and is completely TTL compatible on all inputs and outputs including the power supply (+5v) and single phase clock. The programmable systems interface provides a 9900/9980 system with interrupt control, I/O ports and a real time clock.

The TMS9901 interfaces with the CPU through the Communications Register Unit (CRU). It can perform interrupts and I/O, interface functions via 6 dedicated interrupt input lines, 7 dedicated I/O ports and 9 ports programmable as either interrupts or I/O ports.

The programmable real time clock consists of a 14 bit counter that decrements at a rate of F(Ø)/64 (at 3 Mhz this results in a maximum interval of 349ms with a resolution of 21.3us) and can be used either as an interval or as an event timer.

5.4.1.3 User Accessible Registers on the CPU

There are three user accessible registers in the TMS9900 CPU, the workspace pointer, the program counter, and the status register. These are 16 bit registers with word organisation (ie each being 2 bytes).

Workspace Pointer - The workspace pointer indicates the block of memory to be used as the workspace registers. There are 16 workspace registers, designated RO to RF. All these registers may be used for general operations except RC, RD, RE, and RF. They will be used in the following way:-

(i) RC - used for the CRU base address
(ii) RD, RE and RF - used to store the workspace pointer, program counter and status register respectively during a software context switch or interrupt.

Program Counter - The program counter points to the instruction to be executed next by the CPU. The program counter will automatically be incremented to point at the next instruction prior to the execution of that instruction. The program counter can be set at the beginning of the program using an absolute origin instruction (AORG). For example, if a program begins with AORG 100, followed by an LWPI 100 instruction. The program counter will start at)120 (namely immediately following 32 bytes of memory designated for workspace). Subsequent instructions will be based on)120 as the start point. Thus it is obviously essential that the program counter is set at the correct memory location before execution of a program. Both the workspace pointer and the program counter can be altered if they are not assigned in the program software.

Status Register - the status register contains the interrupt mask level and information pertaining to the prior operations. The bits of the status register are used as follows:-

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
L	A	=	C	O	P	X	..not used......					interrupt mask			

BIT 0	L)	- logical greater than
BIT 1	A)	- arithmetic greater than
BIT 2	EQ	- equal
BIT 3	C	- carry
BIT 4	O	- overflow
BIT 5	P	- odd parity
BIT 6	X	- extended operation

Bits 0 to 6 are set to either 1 or 0 dependant upon the result of the prior instruction.

eg C R1, R2

If the contents of R1 are the same as the contents of R2, resulting from the composition instruction, bit 2 of the status register is set to 1. Similarly other bits are set dependant upon the function of the instructions performed.

Bit 12 through 15 set the level of the interrupt that the microprocessor will accept. The interrupt will then be processed if of sufficient high priority after the completion of the current instruction.

5.4.1.4 Input/Output

The 9900 has a special I/O device called the communications register unit (CRU). This is a one bit wide data bus for I/O having its own control signals which use the address on the main address bus as a bit address of an input/output line. From 1 to 16 bits can be read of written with a single instruction.

Interrupts are one method of controlling I/O. Interrupt level 0 is the highest priority, and level 15 is the lowest. An interrupt mask in the status register loaded with a LIMI instruction determines which level of interrupt can be accepted. Only interrupt of equal or higher priority than the level set in the mask will be accepted by the CPU.

When the microprocessor is in the interrupt service routine the old workspace pointer, program counter and status register are stored in RD, RE and RF respectively, of the new workspace. Status register is also decremented by one, so that only interrupts of a higher priority can interrupt the processor until the service routine has been executed. On reaching an RTWP (return to workspace pointer) instruction, the CPU returns to the old workspace pointer, program counter and status register.

There are five dedicated software instructions associated with I/O from the microprocessor via the I/O bus. For output, they are LDCR (load communications register), SBO (set bit one) and SBZ (set bit zero). For input, the STCR (store in communications register) and TB (test bit) instructions are used. These instructions allow both single bit and multi-bit (up to 16 bits maximum) to be handled.

5.4.2 Interface Printed Circuit Boards

Analogue to digital converters are required to convert the potentiometer readings of the robot into digital signals which can be processed by the microprocessor. Digital to analogue converters are required so a digital velocity word can be converted to an analogue voltage output. Standard Texas Instruments printed circuit boards, the RTI-1241 and RTI-1242, are utilised, the details of which are given in Appendix 6.

This analogue voltage is then amplified through an operational amplifier and additional circuitry, which is illustrated in figure 5.5. This amplified voltage is then used to drive the servo-valves. Relays are used to operate the solenoids for the wrist and gripper movements.

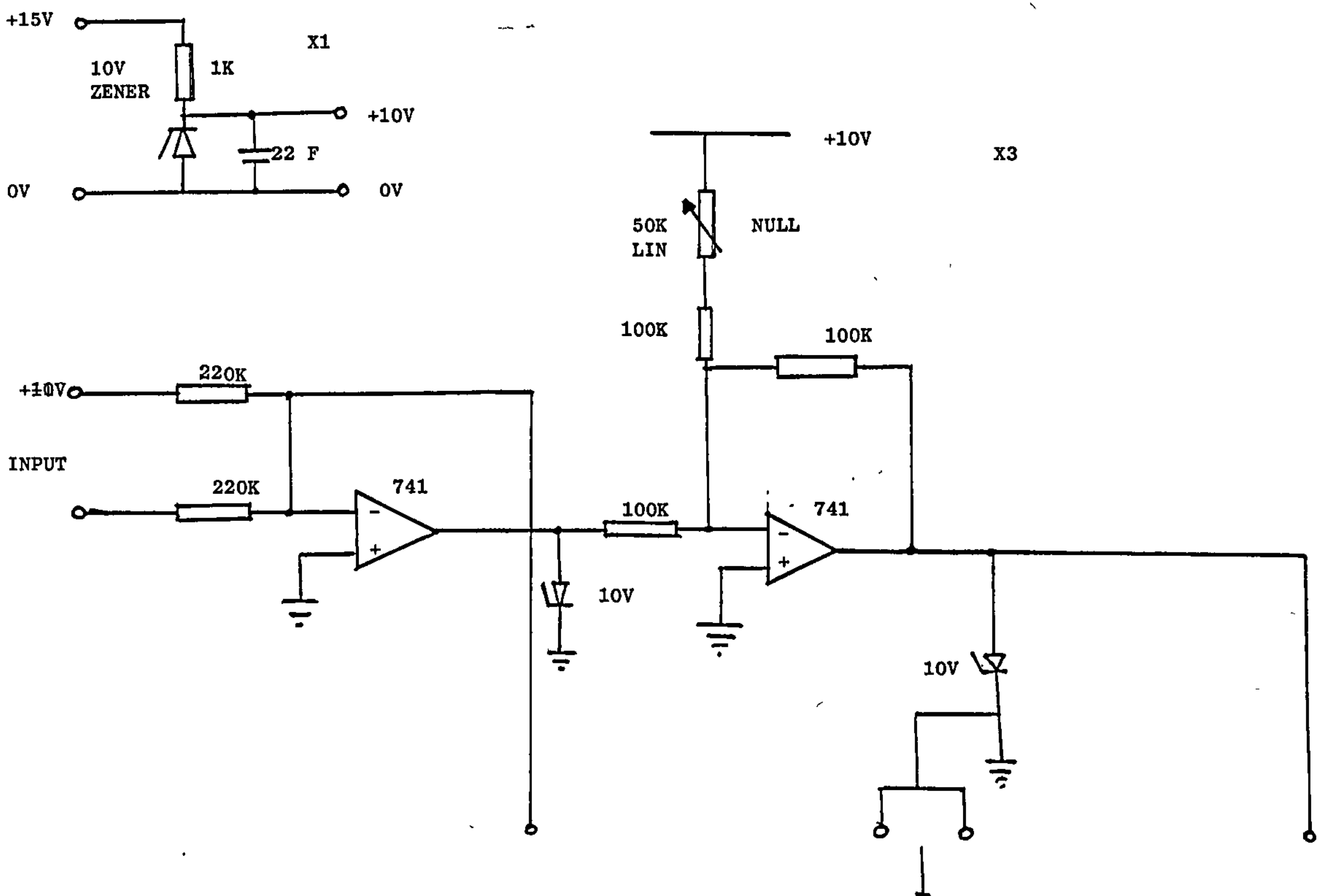


FIGURE 5.5. SERVO-AMPLIFIER MODULE

5.5 SOFTWARE AND HARDWARE DEVELOPMENT FACILITIES FOR USE WITH THE TEXAS INSTRUMENTS 16 BIT MICROPROCESSOR

The Tibug Interactive Debug Monitor provides an interface between the user and the TM 990/101 M microcomputer.through a teletype (TTY) or any RS 232 compatible terminal[52]. It provides commands for loading, debugging and executing a program and also seven software routines which can be called up in user programs by the XOP machine instructions to perform special tasks such as writing characters to a terminal. Loading a program manually into the Tibug debug monitor requires the tedious task of first writing the machine language instructions and keeping track of binary machine addresses within the program. "The Terminal Executive Development System" (TXDS) provides an extensive software capability to assist in developing, improving, changing or maintaining the user's customised operating system and the user's applications programs, or any other type of user produced programs. It gives users the chance to write programs in assembly language and then edit, assemble and debug them. It does this by means of the following nine utility programs:-

(i) TXDS Text Editor
(ii) TXDS Assembler
(iii) TXDS Copy/Concatenate
(iv). TXDS Linker
(v) TXDS Cross Reference
(vi) TXDS Standalone Debug monitor
(vii) TXDS PROM Programmer
(viii) TXDS BNDF/High Low Dump
(ix) TXDS LUNO

The TXDS Terminal Executive Development System programmer's guide[75] gives a detailed description of the utility programs.

An in-circuit emulator and a logic-state trace data module are included in hardware configuration in the AMPL Microprocessor Prototyping Laboratory. This laboratory is structured around the FS 990 system, which includes a video display terminal, a dual floppy disc unit and the TX 990/TXDS system. It provides a dedicated design centre where both the software and hardware of any

9900 - based system can be designed and debugged. Additional information can be found in various manuals [74,52] and reports [72,73].

CHAPTER 6

THE DEVELOPMENT OF REAL-TIME CONTROL SOFTWARE

6.1 INTRODUCTION

As discussed earlier, to achieve the required positioning accuracy on the three major axes, the closed-loop control concept is utilsed. In a simple closed-loop system the controller is no longer actuated by the input (as is the case for an open-loop system), but by the 'error'.[59,60] The error is defined as the difference between the system input and output. Such a system contains the same basic elements as the open-loop system (see figure 6.1), plus two extra features - and 'error detector' and a feedback loop. The error detector is a device which produces a signal proportional to the difference between input and output (figure 6.2). Thus there are three basic components which are required for a closed-loop system.

i) The Error Detector

This is a device which receives the low-power input signal and the output signal which may be of different physical natures, converts them to a common physical quantity for the purposes of subtraction, performs the subtraction, and gives out a low-power error signal of the correct physical nature to actuate the controller.

ii) The Controller

This is an amplifier which receives the low-power error signal, together with power from an external source. A controlled amount of power (of the correct physical nature) is then supplied to the output element.

iii) The Output Element

This provides the load with power of the correct physical nature in accordance with the signal received from the controller.

6.2 CLOSED-LOOP CONTROL FOR THE ROBOT

The control is achieved by outputting a velocity error signal, the control loop is closed within the microprocessor to permit software

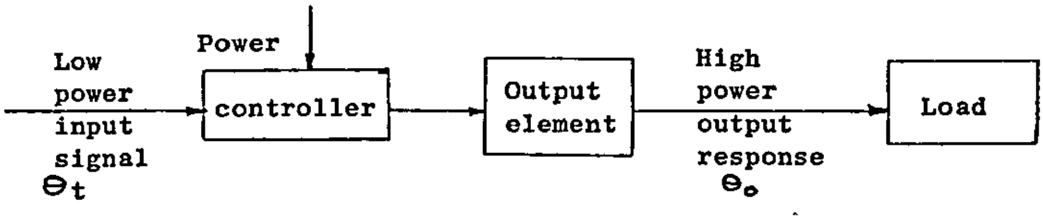


Figure 6.1 Open-Loop System

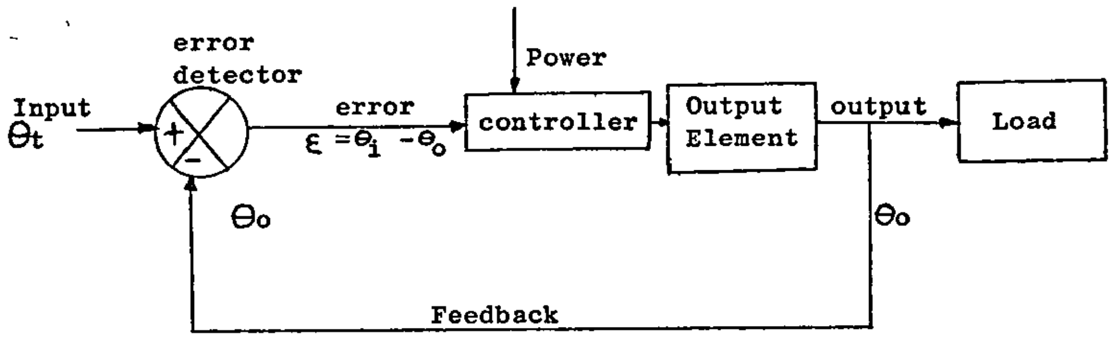


Figure 6.2 Closed-Loop System

control of the velocity error signal. Figure 6.3 shows a schematic diagram of the closed-loop scheme utilised.

The actual position (ACT) of the robot is converted to a digital word via an ADC and fed back into the software "comparator". The actual position is produced by the potentiometers which are on each major axis of the robot. The voltage range is from 0 to 10V which is dependant upon the position, see figure 6.4 which illustrates a trace on the output voltage. This voltage is then converted to a digital word by an analog to digital convertor and the associated input channel of the muliplexer. The ACT position is then compared to the commanded position (CMD) which are both stored in the memory. The numerical value of this error and its sign determines the numerical value and sign of the output velocity word. The velocity word is then converted to an analogue voltage by the digital to analogue converter associated with the axis in question. This output analogue voltage is then amplified through an operational amplifier and additional circuitry which is shown in figure 6.5. This amplified voltage is used to drive the appropriate servo-valve, which in turn controls the actuator. The actuator is a motor or hydraulic ram which is dependent upon this axis which is being controlled.

In the early stages of this project it was considered necessary to limit the maximum velocity of each major axis of the robot, thereby providing a measure of protection during the design and evaluation of the controls. The limiting of velocity is achieved within the microprocessor by limiting the maximum value of the velocity output work for both the negative and positive values. The initial digital value corresponding to this limit was set at the hexadecimal number (>)180, which converts to 1.86 volts before amplification. This amplified voltage produces a controlled motion for all the axes on the robot. If the difference between the ACT and CMD positions is less than > 180, then the velocity output word is proportional to this error value. Using this method the velocity of the robot is reduced as it approaches the commanded position so to eliminate overshoot, see figure 6.6.

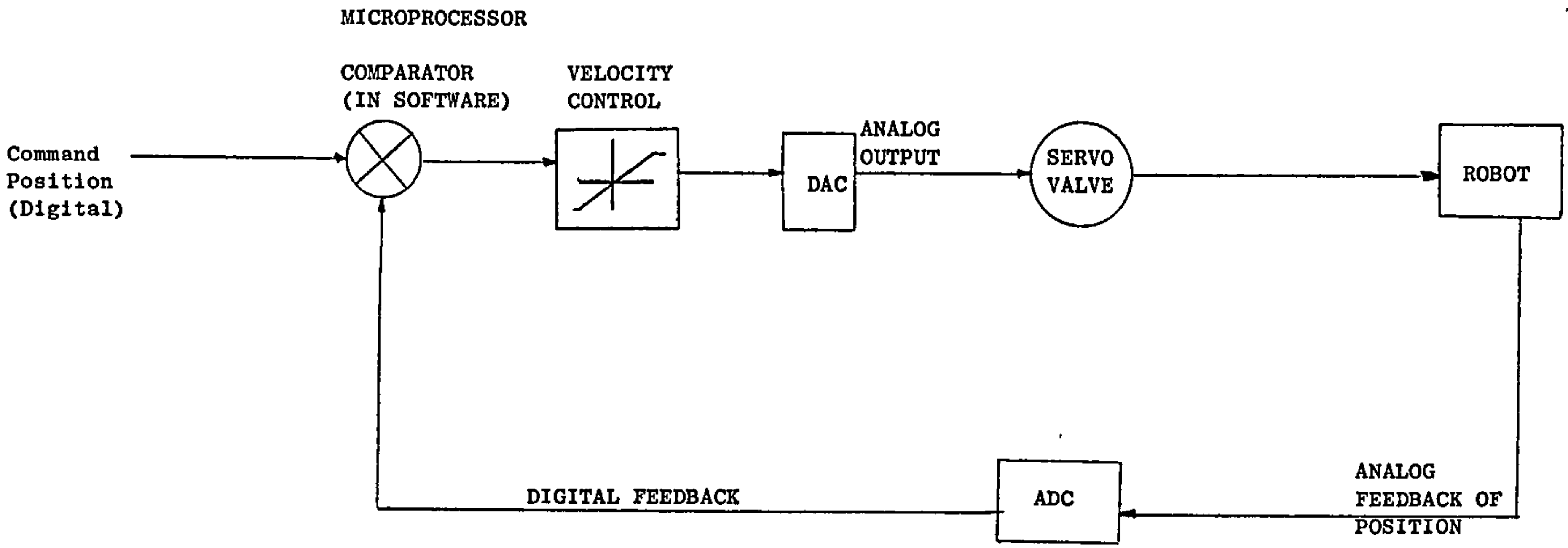
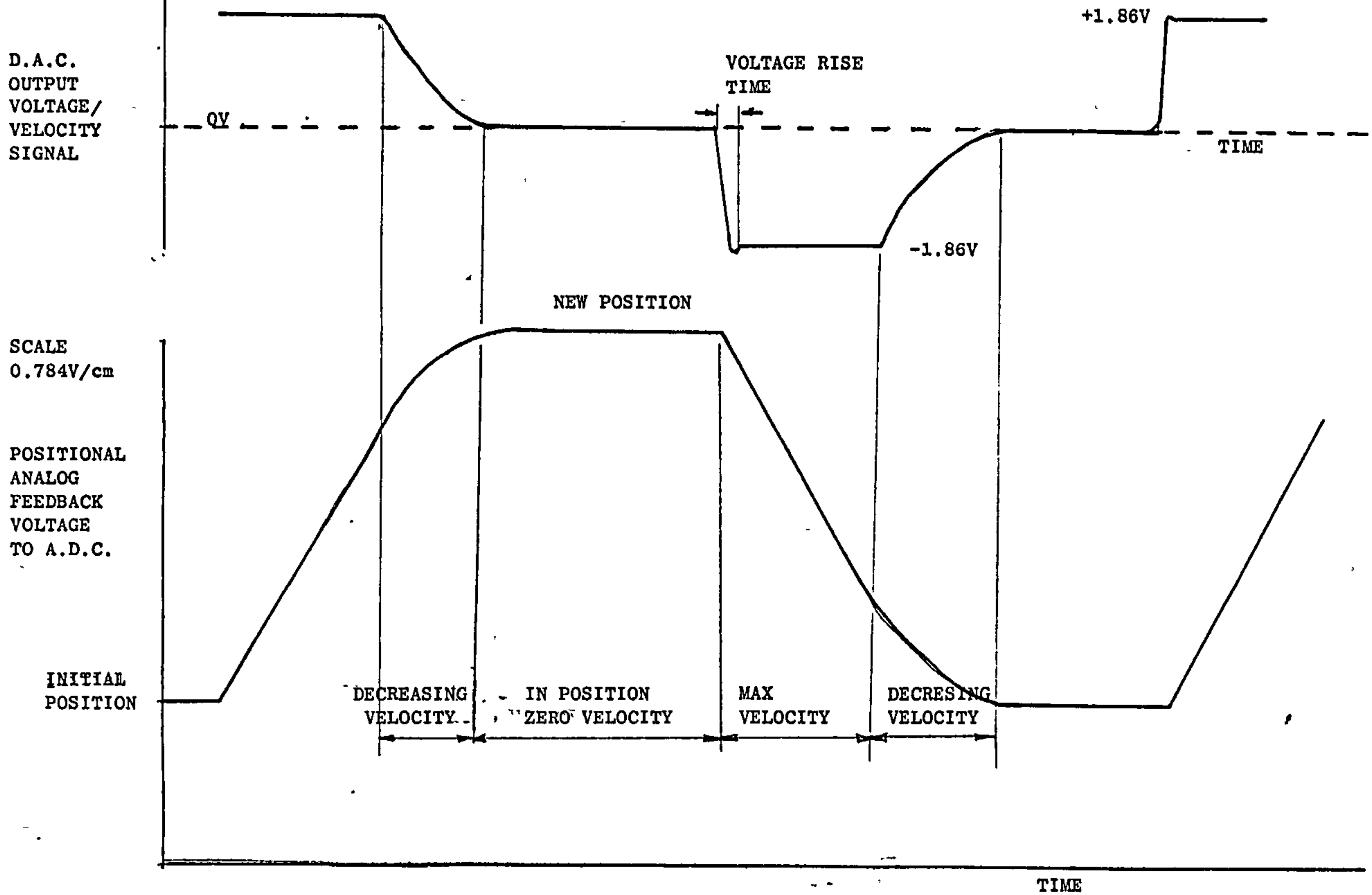


Figure 6.3 Schematic Diagram of Closed Loop Control

FIGURE 6.4. DESCRIPTION OF TEST TRACES



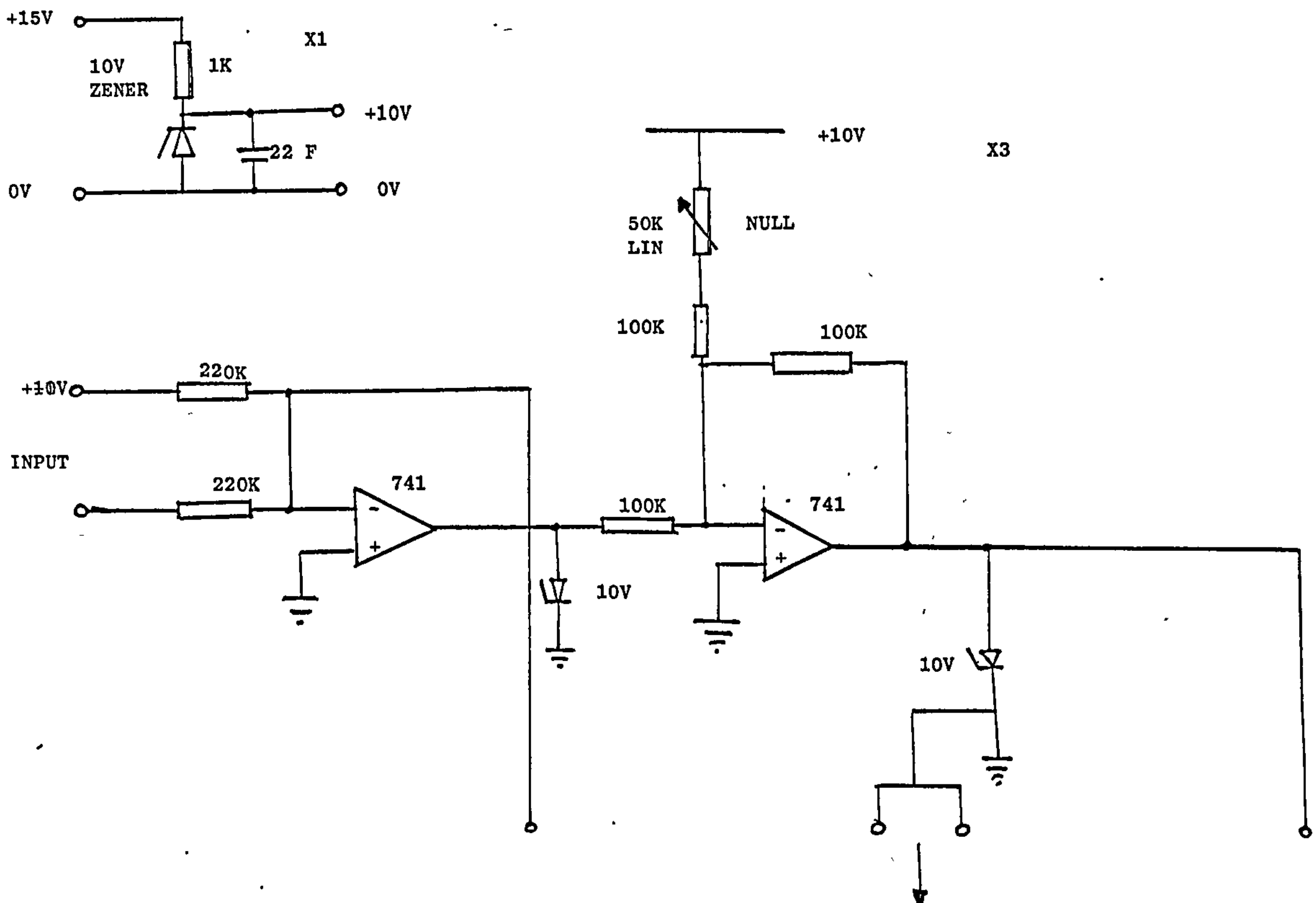


FIGURE 6.5. SERVO-AMPLIFIER MODULE

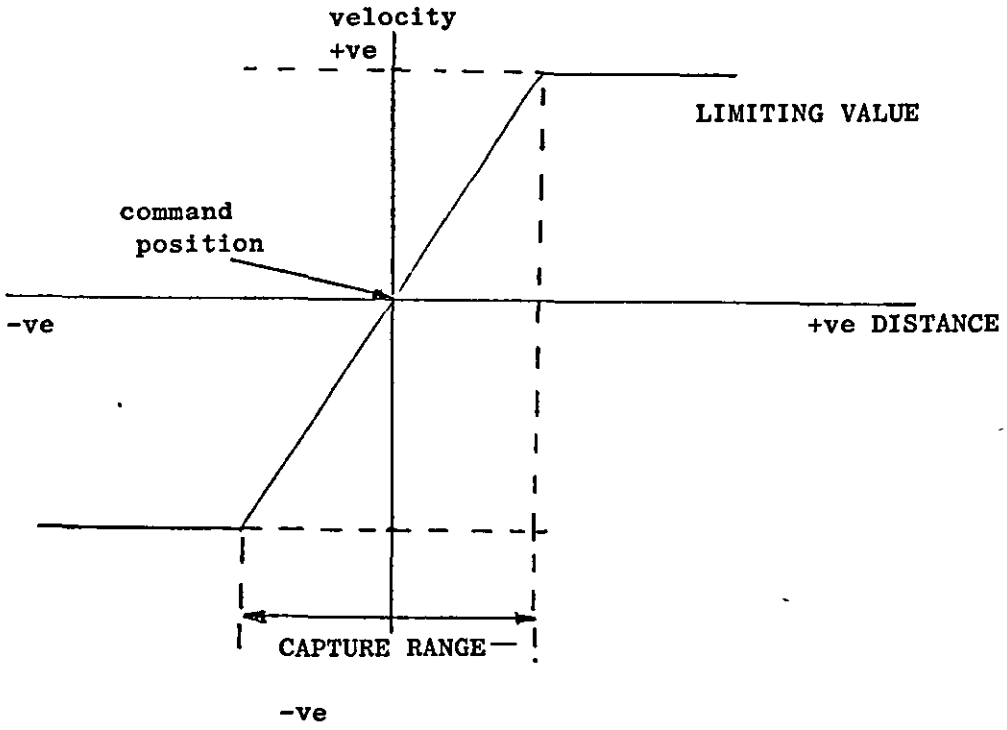


Figure 6.6 Velocity Control

6.3.1 Control of One Axis

Initially it was decided that software should be developed to control just one axis. Before this software was tried out on the robot a potentiometer and bench power supply were used to generate an analogue signal to check that the connections were correct for the servo valves (that is to ensure that a reducing error signal should be obtained). When this had been done for all three connections (ie all the axes) the microprocessor was connected to the robot and the program executed. In early development, a decision was also taken to run all programs using interactive programming that is, each command position required was input into memory via a VDU or teletype. This gave greater flexibility when testing for accuracy etc. The various approaches to the software will now be described.

6.3.2 Program TRY1

The details of the program structure can be seen in figure 6.7 and 6.8. (A full program listing is shown in Appendix 7).

Figure 6.7 Overall Requirement of Software

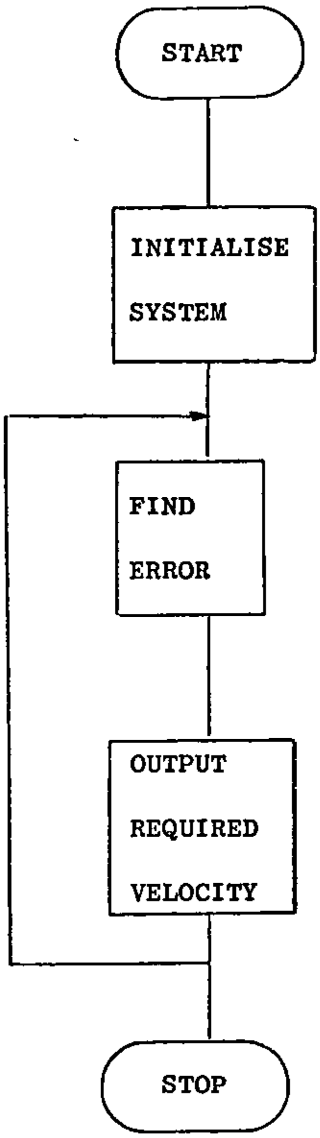


Figure 6.8 Flow Chart for Program TRY 1

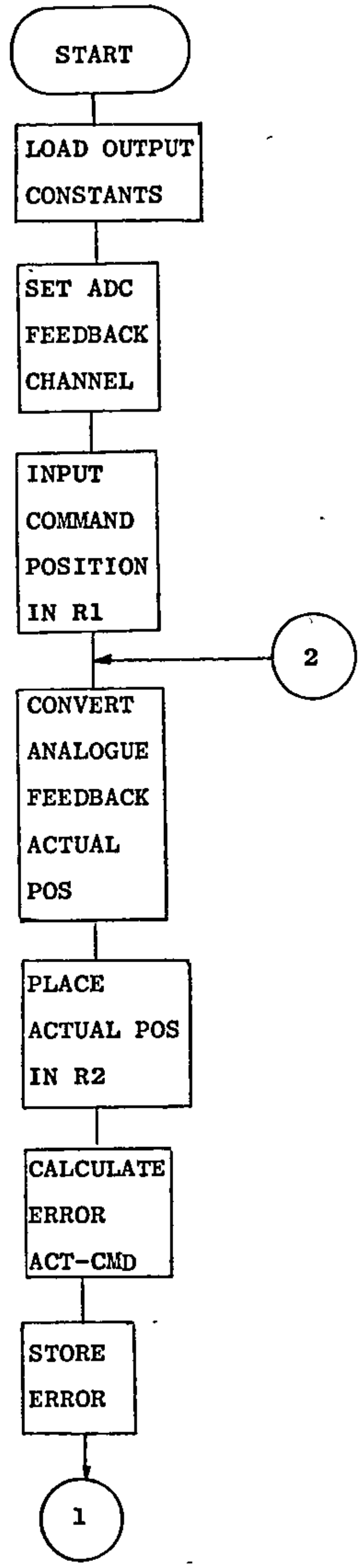
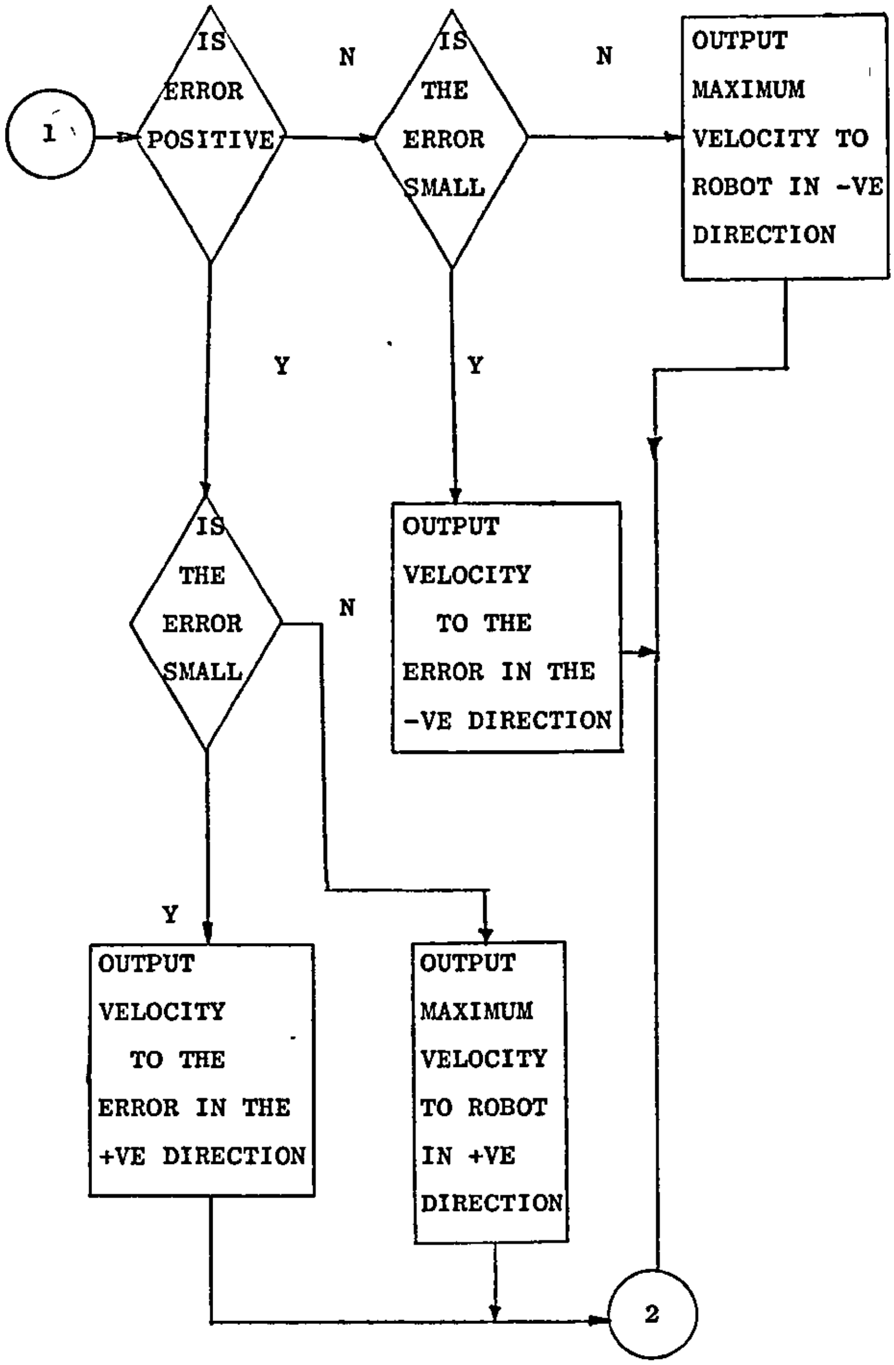


Figure 6.8 (continued)



The directive instruction EQU is used in the program to equate various memory locations to symbols to allow software updates to be achieved easily and to simplify interpretation of the software.[61]

For example

```
ADCDAT          EQU          >EFFE
  ↑                            ↖
ADC DATA                        MEMORY LOCATION
SYMBOL

CONV            EQU          >EFFA
```

Particular use of these directive instructions has been made in connection with the ADC and DAC control software as the ADC/DAC is a block of words in memory, now each of these words has been designated with an appropriate label.

The output constants are then loaded into registers.

LI R5,>FE8Ø
FE8Ø is equivalent to ->180 and so this is the maximum negative value which is permitted. Similarly a maximum positive value is in register 4.

The channel on the ADC is set using CLR@MUXADR which selects channel Ø. An ADC gain of unity is selected as any of the other possible gains, which are 2,4 or 8 will result in the reduction of the size of the output voltage before amplification. A larger amplification could be used to increase the output voltage however, additional circuitry would have to be designed.

CLR @GAIN
In this program only one axis position is specified and the value in this case is >3FF (this can be any value between Ø and >7FF) which is stored in register 1.
LI R1,>3FF
The conversion of the analogue feedback signal is intiated using the following instructions:-

```
SAM SETO @CONV
CHK INV @STATUS
    JLT CHK
```

This method is termed 'Polled Status Control'(56). Using this method a specific command is required to start the conversion process. The command can be achieved using either an external signal or by a signal from the CPU, in this case using the SETO (set to one) command, this is the quickest method in terms of instruction speed. The loop following the start of the conversion process continues until the EOC (end of conversion bit) in the 'status' is a 'one'. The EOC is the leftmost bit in the word at STATUS. The most efficient method of checking for a one or a nought, in terms of execution time is shown above. The INV (logical complement) does not change the STATUS word (read only at STATUS), but it sets the appropriate bits in the status register accoding to the result of the INV operation. The JLT instruction tests these bits in the status register and only branches if the operand of the preceeding INV operation had a Ø in its MSB (the EOC bit). Once the conversion is complete the CPU is then allowed to read the value at ADCDAT, which is the result of the conversion just described. This value can be read into a register using only a single instruction

```
MOV @ADCDAT,R2
```

In a similar way it is possible to carry out an arithmetic operation using @ADCDAT as the source operand. For example, if it is necessary to add the feedback value to the contents of another register
eg A @ADCDAT,R4
The next operation after the actual position is in R2 to calculate the error between the commanded position and the actual position.

S R1,R2 calculates ACT-CMD
The answer for this calculation is in R2, that is the error is in R2. This value is then subjected to a series of compare immediate instructions and the conditional jumps which follow determine the value output to the DAC. The positional error is compared to 180, if it is less than >180 the JLT LAB1 comes into operation otherwise the maximum positive velocity is output ot DAC 2 by the instruction

MOV R4, @DAC2

followed by an unconditional jump (JMP SAM) to sample once more the actual position of the robot arm. Conversley if the error is less than >180 then it is compared to > FE8Ø (which is ->18Ø) using the instruction

LAB1 CI R2,>FE8Ø

If the error is greater than FE8Ø, that is it lies between+>180 and ->180 then the statement JGT LAB2 comes into operation and the output velocity word is directly proportional to the positional error value calculated. This is achieved using the instruction

LAB2 MOV R2,@DAC2

again the unconditional jump operation JMP SAM is executed. If the error is not greater than >FE80, that is a higher negative number then the maximum negative velocity word is output to DAC 2 by

MOV R5,@DAC2

again the unconditional jump operation JMP SAM is executed.

This program showed how the robot could be programmed but it has severe limitations which include,

a) Only one position is used for one axis
b) The microprocessor is sampling at a faster rate than is necessary.

These limitations are overcome in the following programs.

6.3.3 Program TRY2

This program will allow a single axis to move to more than one position (a full program listing is shown in Appendix 7).

The subroutine illustrated by the flow chart in figure 6.9 is added to TRY1 after the ADC channel has been selected. Furthermore the error signal is compared with zero after it has been stored (figure 6.10).

The subroutine beginning at 'NEXT' asks the user, via a VDU prompt, 'What is the command position?' using the following instructions.

Figure 6.9 Flow Chart for Program TRY 2

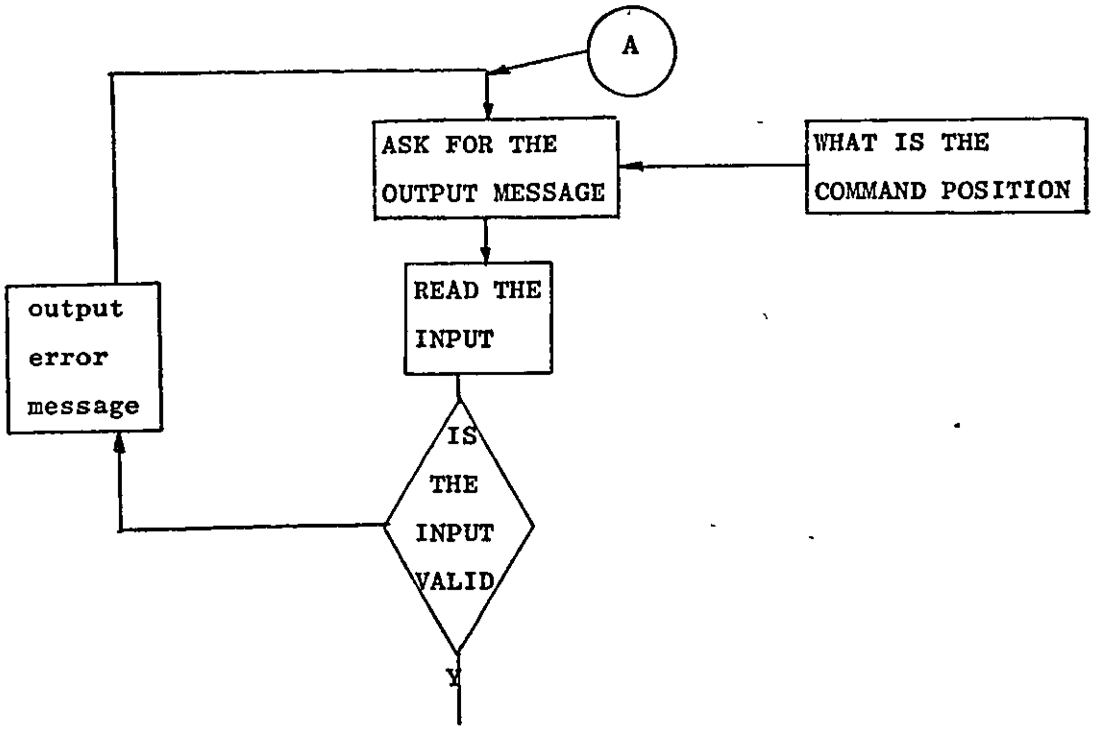
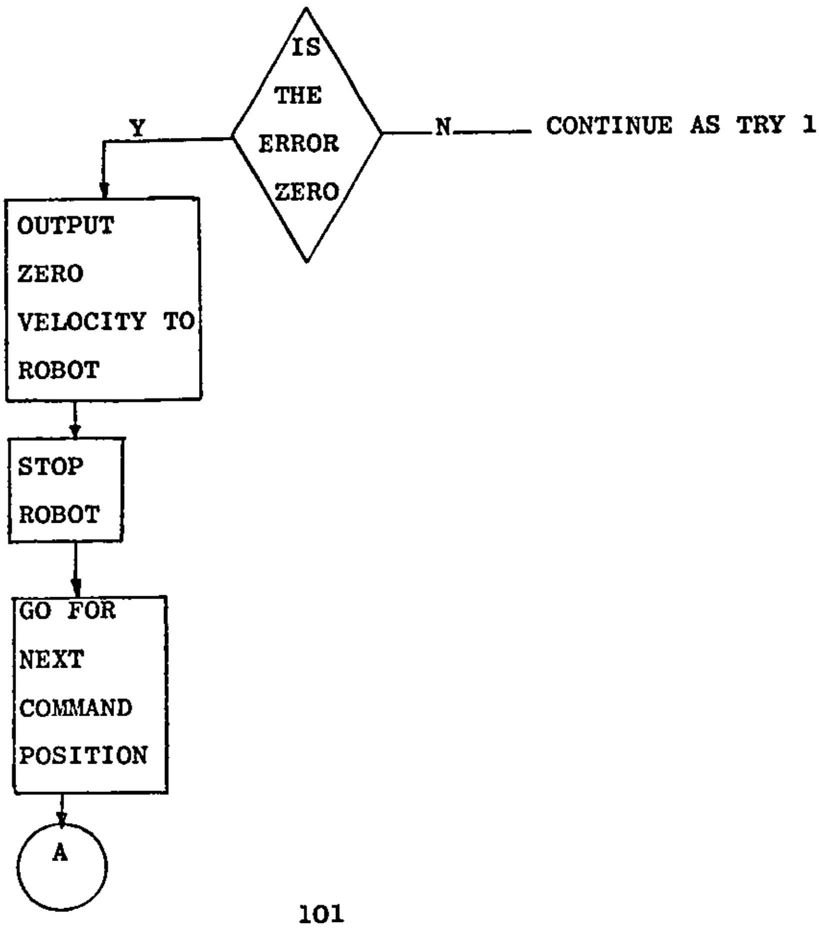


Figure 6.10



XOP @ MESS,14

This command writes the message out to the terminal asking for the position.

NULL XOP R1,9

This reads the value from the terminal into R1 and the following code - DATA NULL is included to ensure that only a hexadecminal number is read in. The value in R1 is then checked to see if it lies between 0 and >7FF by two "compare immediate" statements - if it is between these limits execution will commence, otherwise an error message is printed out using an XOP and then the value is asked for again.

After the error has been stored if it equals zero the program jumps to LAB 3 and outputs a zero velocity word at DAC 2 which stops the robot and then proceeds to ask for the next command position via XOP instructions as previously described. If the error is not equal to zero execution continues as in TRY1 until it does equal zero.

6.3.4 Program TRY3

This program will input a table of data before execution commences for one axis and is illustrated by the flow charts in figures 6.11,6.12. The first message asks how many positions there are going to be, this value is stored in R9 using the XOP to read 4 hexadecimal characters from the terminal, the value is then copied in R11 by the statement

MOV R9, R11.

Then R8 is loaded with a memory location for the start of the table of positions.

LI R8,>FA5Ø.

Another XOP writes out the message asking for the position. The hexadecimal value is read into the memory location which is in R8 and then R8 is increased by two, so when the next value is read in it goes into the next memory location, this is achieved by one statement

XOP *R8+,9

The number of positions left is decrement, that is

DEC R9

Figure 6.11 FLOW CHART FOR TRY3

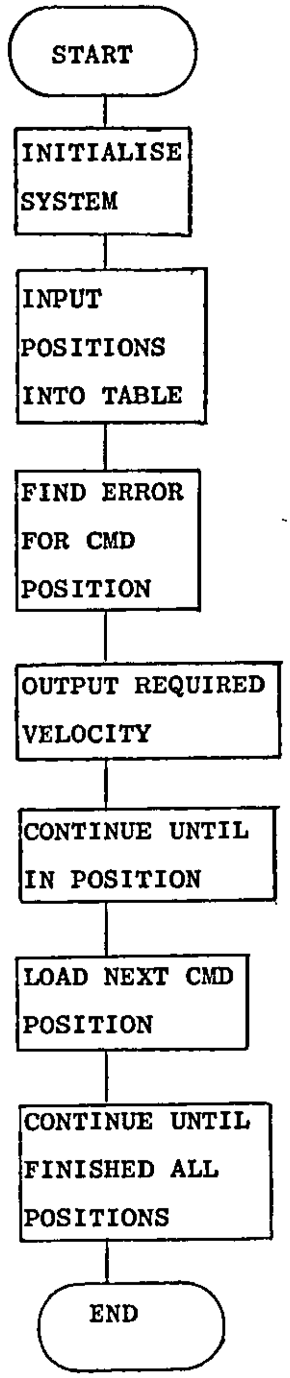
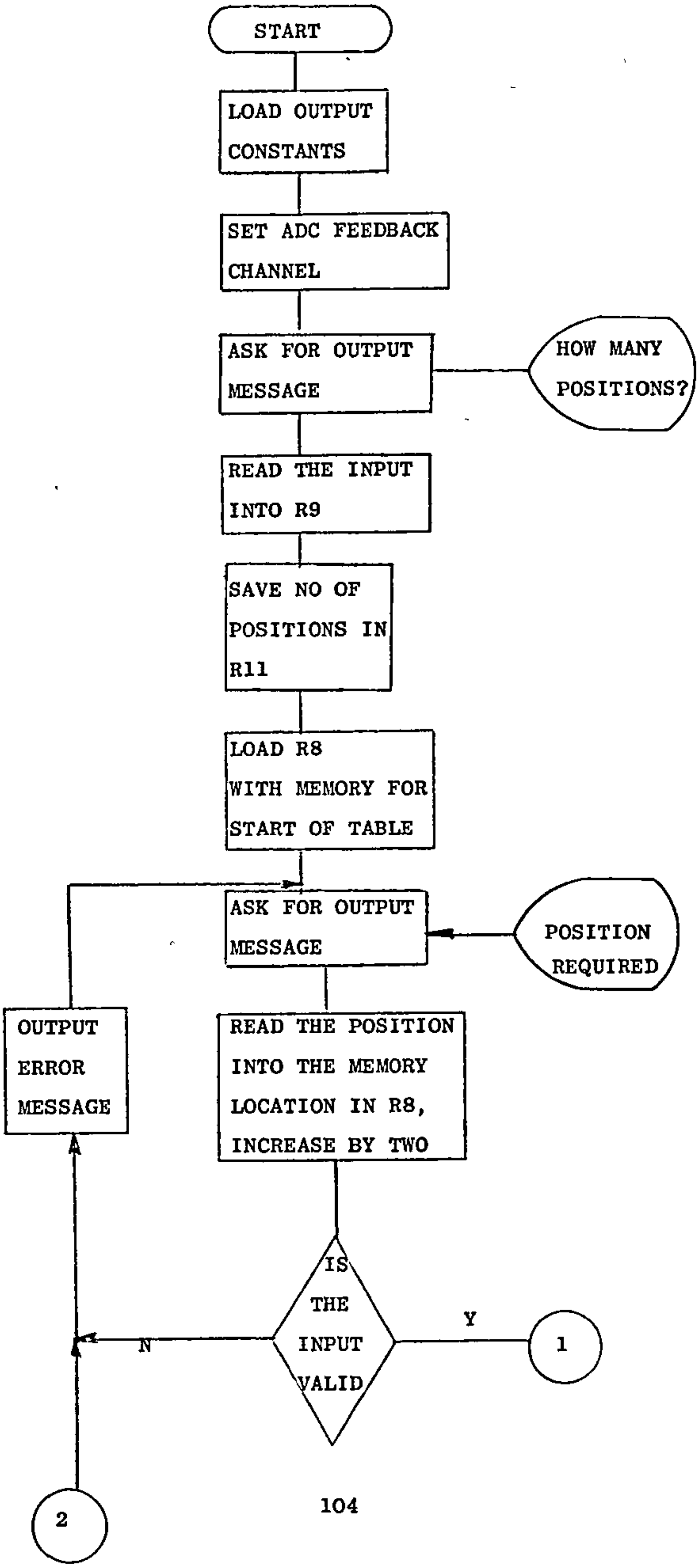
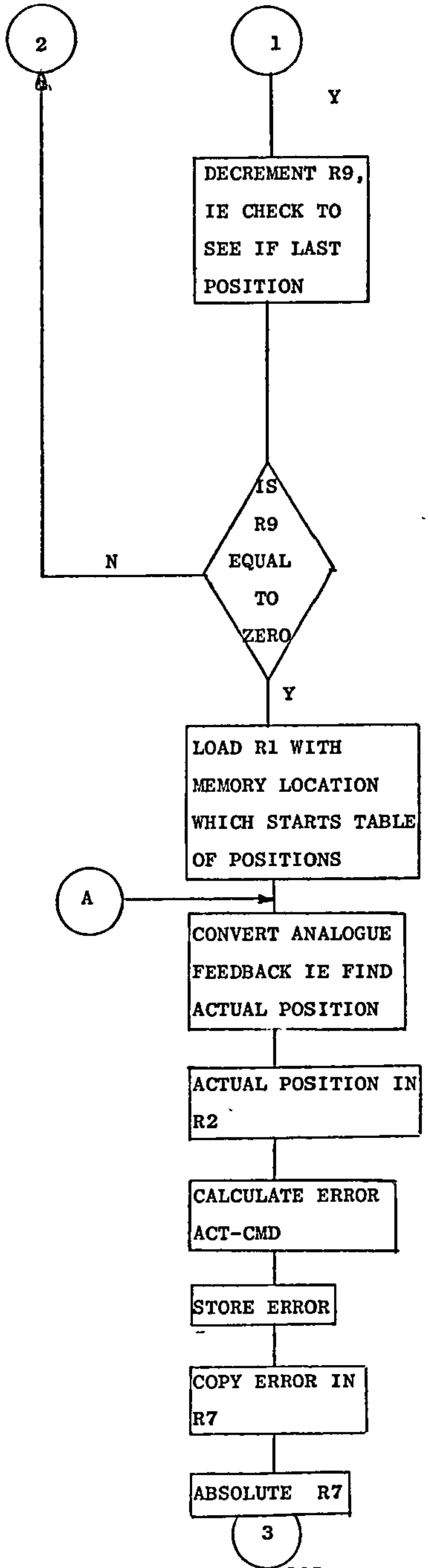
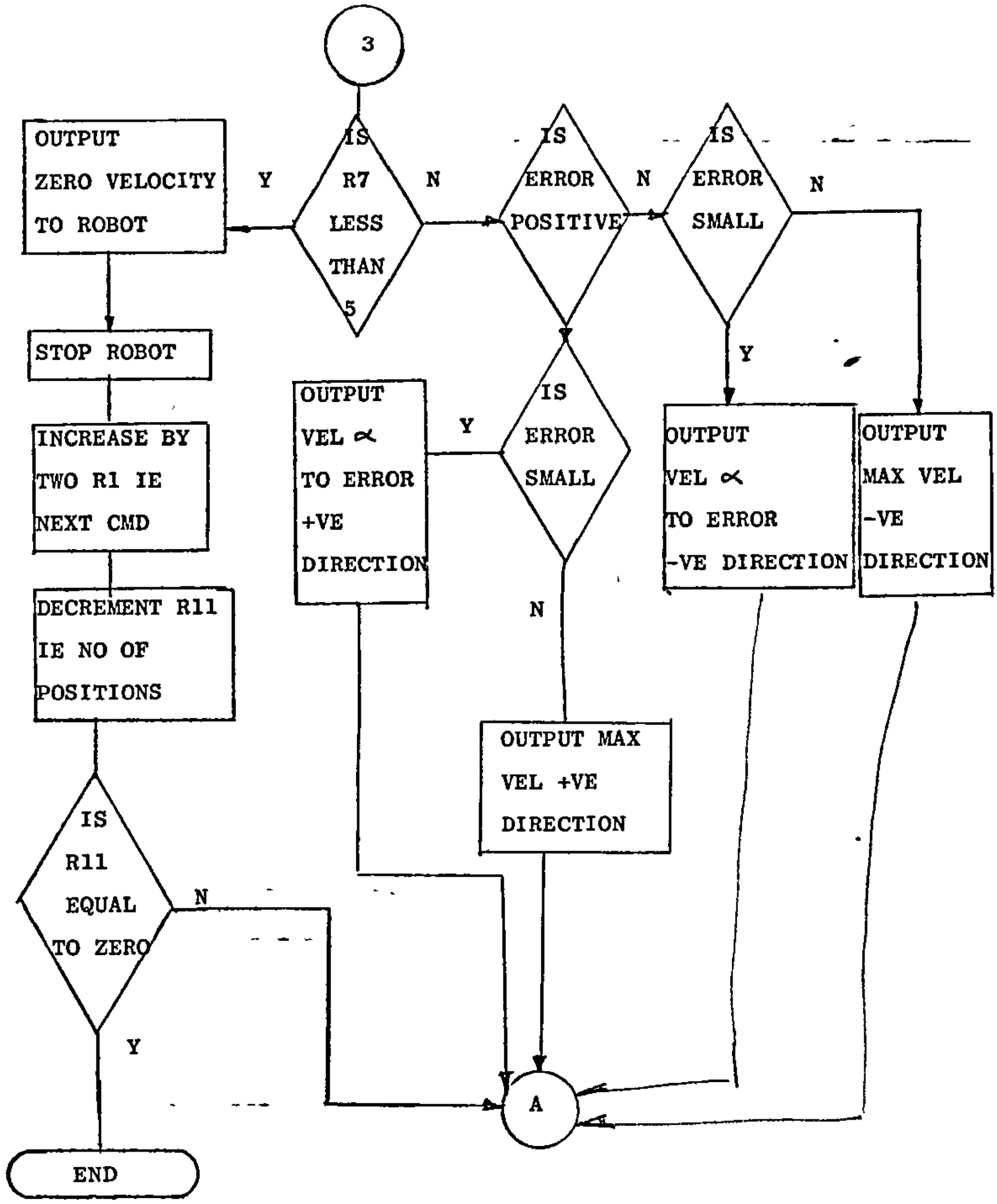


Figure 6.12 DETAILED FLOW CHART FOR TRY3




2
1
Y
DECREMENT R9, IE CHECK TO SEE IF LAST POSITION
IS R9 EQUAL TO ZERO
N
Y
LOAD R1 WITH MEMORY LOCATION WHICH STARTS TABLE OF POSITIONS
A
CONVERT ANALOGUE FEEDBACK IE FIND ACTUAL POSITION
ACTUAL POSITION IN R2
CALCULATE ERROR ACT-CMD
STORE ERROR
COPY ERROR IN R7
ABSOLUTE R7
3


3
OUTPUT ZERO VELOCITY TO ROBOT
Y
IS R7 LESS THAN 5
N
IS ERROR POSITIVE
N
IS ERROR SMALL
N
STOP ROBOT
Y
INCREASE BY TWO R1 IE NEXT CMD
OUTPUT VEL ∝ TO ERROR +VE DIRECTION
Y
IS ERROR SMALL
OUTPUT VEL ∝ TO ERROR -VE DIRECTION
OUTPUT MAX VEL -VE DIRECTION
DECREMENT R11 IE NO OF POSITIONS
N
OUTPUT MAX VEL +VE DIRECTION
IS R11 EQUAL TO ZERO
N
Y
A
END

The value in R9 is then compared to zero, if it is equal execution of the program to move the axis will commence, otherwise the next position will be read in. Register one is loaded with the base address for the table of positions. The conversion of the actual position takes place as previously described and is stored in R2. The error is calculated as ACT-CMD but this time indirect addressing is used for the CMD

```
S *R1,R2
```

and the error is then saved in R7 and its modulus is obtained using the ABS instruction.

```
MOV R2,R7
ABS R7
```

If the modulus of the error is within an acceptable tolerance, which is five, the program outputs zero at DAC2 which stops the robot. The next position is then obtained by incrementing R1 by two, this points to the next value in the table of positions. Decrementing R11, to see how many positions are remaining, the value of R11 is then compared to zero, if it is equal then the program has been executed otherwise the robot will be moved to this next position in the manner just described. If the modulus of the contents in R7 is greater than 5 then the corresponding velocity is output on DAC2 as described in the previous programs.

6.4 PROGRAM VERTHREE

The ideas developed in these programs were subsequently utilised to produce more structured software which could find application in the control of various types of robot. The structure adopted will now be considered.

6.4.1 Instruction Format

It was decided that the control software would be produced along similar lines to that with other microprocessor controllers within the department[72,73]. In these systems each operation consituted an instruction. Each instruction to be programmed is represented by a 16 bit word in memory. This 16 bit word has a pre-determined format depending upon instruction type. For other robot controls developed[72,73] each instruction has an op-code, which is designated by the six most significant bits of the word and which defines the operation which is to be peformed. The remaining ten bits of the word form the modifier, which was used, eg to denote the required position of an arm or the length of a time delay etc. The choice of a six bit op-code was somewhat arbitrary. This approach is maintained which means that the flexibility of the language is extended to a robot of complex form. However, the op-code used is only five bits. This alteration was made as I thought that thirty-one different operations was sufficient and also to allow the maximum traverse on an axis to be input without any modification, ie the maximum travserse on each axis is represented by digital values between zero and 2047 (>7FF) which can be specified be eleven digits.

Table 6.1 shows the set of instruction considered to be necessary in defining point to point tasks for the Versatran together with their associated op-codes and modifiers.

INSTRUCTION	OP-CODE	MODIFIER	COMMENT
MOVE VERTICAL	1	0-2047(>7FF)	POSITION REQ'D
MOVE HORIZONTAL	2	0-2047	POSITION REQ'D
MOVE IN SWING	3	0-2047	POSITION REQ'D
TIME DELAY	4	0-2047	NO OF ¼SECS
JUMP	5	0-512(>200)	NO OF INSTRUCTIONS
TURN WRIST VERTICAL	6	XXX	NO MODIFIER
TURN WRIST HORIZONTAL	7	XXX	NO MODIFIER
STOP	8	XXX	NO MODIFIER
CONTINUE	9	0-2047	NO OF REPEATS
CLAMP OPEN	A	XXX	NO MODIFIER
CLAMP CLOSED	B	XXX	NO MODIFIER
		X = DON'T CARES	

Table 6.1 Programmable Instructions

A more detailed description of this type of instruction format is given by Charles and Weston(76) and Mason(73) and Sahili(72). A brief description of the functions of this Versatran Instruction set is given below

MOVE INSTRUCTION

For positioning one of the third major axes by using an absolute address method the modifier contains a digital number (0-2047) which is equivalent to the required position.

STOP INSTRUCTION

Always the last instruction in a program placed in 'user memory' to terminate the program and bring control back to the operator.

DELAY INSTRUCTION

For producing a 'real' time delay in increments of 'quarter' seconds.

JUMP INSTRUCTION

To jump blocks of instructions, that are to be used later.

CONTINUE INSTRUCTION

To produce the desired number of repeats of a program, a 9000_{16} command will continue cycling the robot until stopped eventually.

TURN WRIST

The wrist can be either in a vertical or horizontal posture.

CLAMP OPEN/CLOSE

The gripper can be opened or closed.

6.4.2 Description of 'VERTHREE' Versatran Control Program

This is the full controller program. In this program the instructions, as described in 6.4.1 located in memory in sequence, are interpreted and the appropriate output to the robot results, once the program is executed. The program again has a modular design, with a separate sub-routine for each function. The general software configuration is shown in figure 6.13 the detailed software is shown in flowchart form in figures 6.14. to 6.25 A full program listing is included in appendix 7.

The program allows up to 512 instructions to be loaded in sequence, beginning at memory location >FB00. This section of memory is referred to as 'user memory'. The instructions are decoded by the program in sequence, one word of memory at a time. The jump instruction permitting sections of user memory to be jumped over if so desired. The complete program format is shown in flowchart form see end of this section.

Instruction Read and Decode Sub-Routine (IRD)

The IRD sub-routine sets up a pointer in memory that indicates the location of the next robot instruction (see fig 6.14). This program assumes that a sequence describing the robot task to be

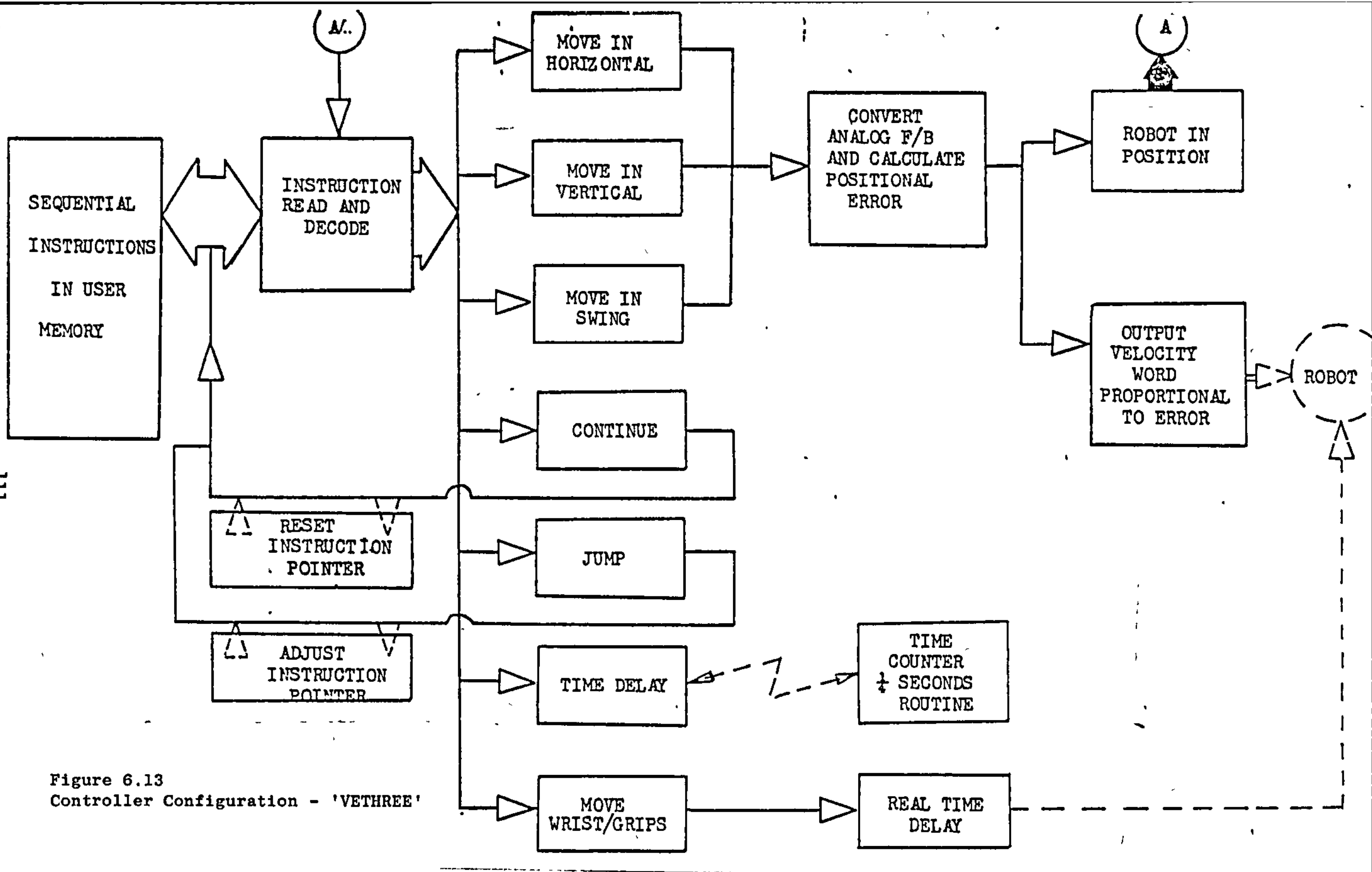


Figure 6.13
Controller Configuration - 'VETHREE'

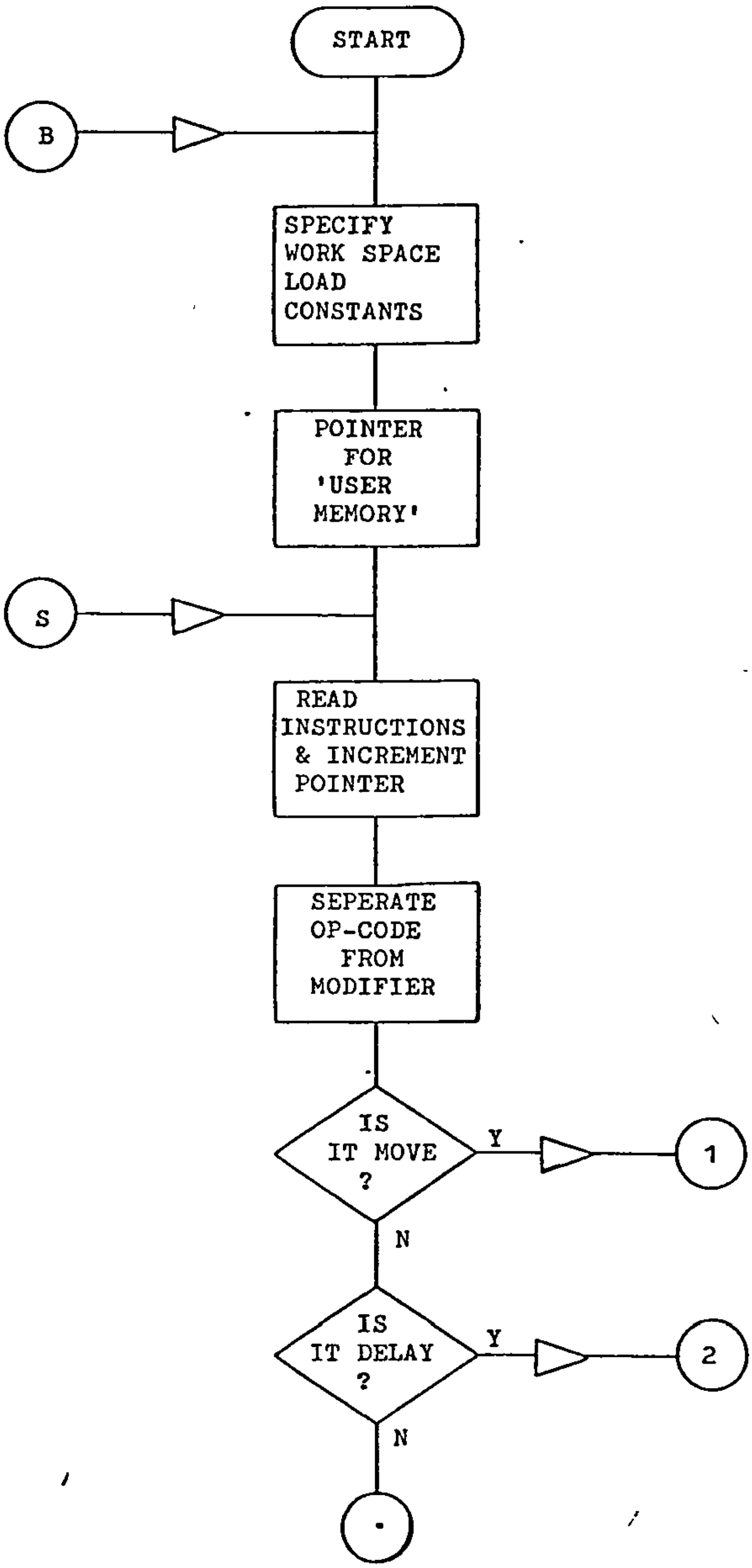
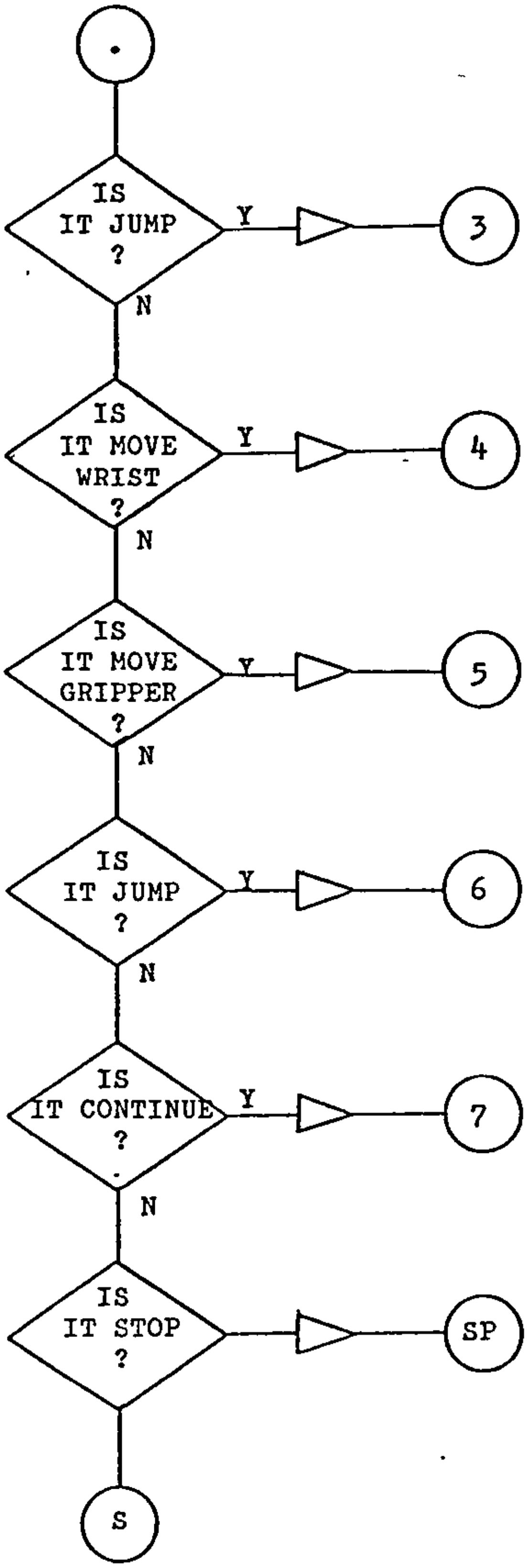


Figure 6.14
Instruction Read & Decode


IS
IT JUMP
?
Y
3
N
IS
IT MOVE
WRIST
?
Y
4
N
IS
IT MOVE
GRIPPER
?
Y
5
N
IS
IT JUMP
?
Y
6
N
IS
IT CONTINUE
?
Y
7
N
IS
IT STOP
?
SP
S

performed comprises a number of robot instructions residing in memory in the format described earlier. The current instruction is collected and placed in a register and the op-code separated from the modifier (this achieved using the logic ANDI instruction in a mask of any of the 16 bits in a word can be obtained with this instruction). Once the op-code has been separated, it is then used in a sequence of compare instructions which provides software decoding of the instruction. The IRD sub-routine then supervises a branch to the real-time control sub-routine associated with that instruction and the robot instruction pointer is automatically incremented to point to the next instruction in memory although this pointer can be modified by some other sub-routines. The modifier is separated and is available when the subroutine branch is made.

6.4.2.1 Robot Instructions - Real Time Control Routines

<u>To Initialise a Move</u>

If an op-code for a move instruction is recognised the program jumps to this routine and the appropriate analog feedback channel is selected by placing the correct multiplex code on the ADC (see figs 6.15-6.18). Register 10 is loaded with a displacement value which determines at which DAC the velocity word is output in the next sub-routine. The program then jumps to the convert routine.

<u>I/O Analog-Digital Handling Routines</u>

These routines are used to process the analog feedback signal, calculate the positional error and output the appropriate output velocity word (see fig 6.19). When a value is to be output to a DAC the following instruction is used:-

```
(LABEL) MOV R6, @DAC2 (R10)
```

With this instruction, the value in register 6 is copied at memory location (DAC2) plus the displacement value in register 10. In this way, using DAC2 as the base address memory location, the appropriate memory location receives the contents of Register 6. It should be noted that the three DAC's used reside at the memory locations >EFF0, >EFF2 and >DEFE.

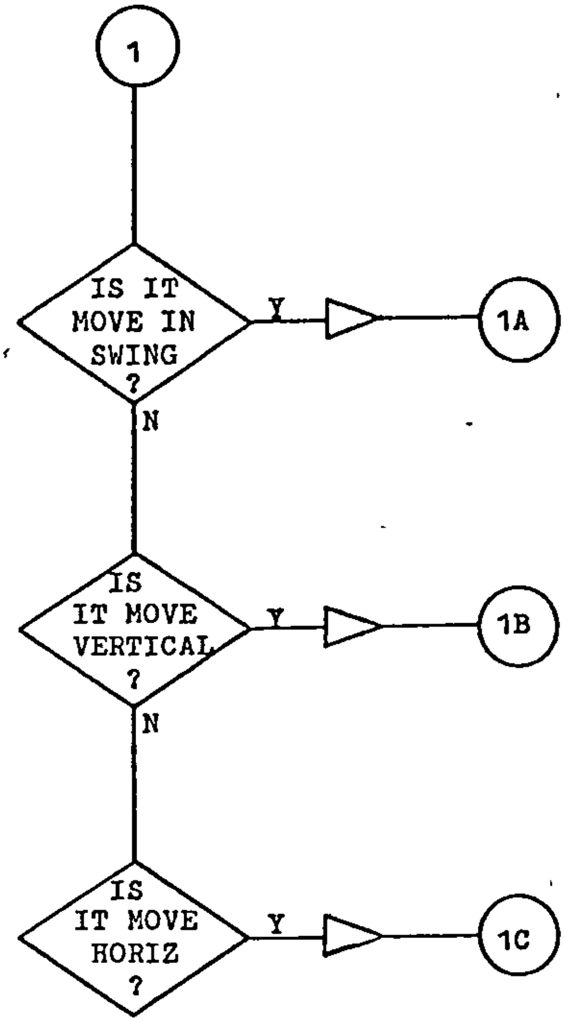


Figure 6.15
Select Axis to be moved routine

Figure 6.16 Initialise a move in swing

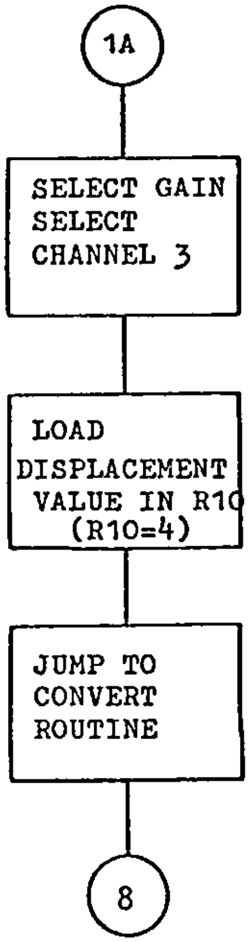
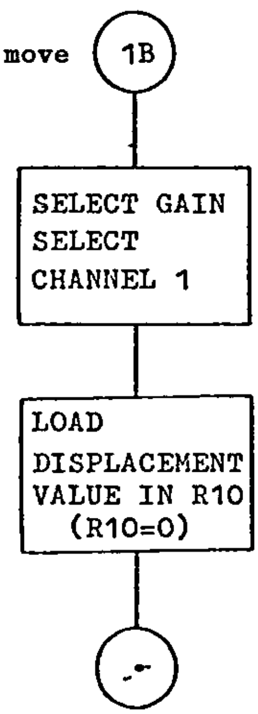


Figure 6.17
Initialise a move
in vertical



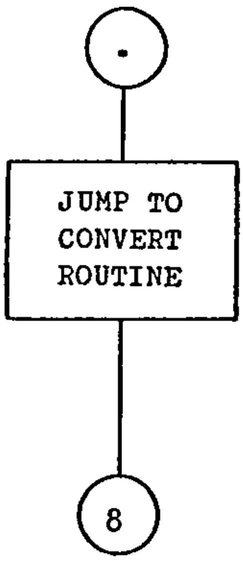


Figure 6.18 Initialise a move in horizontal

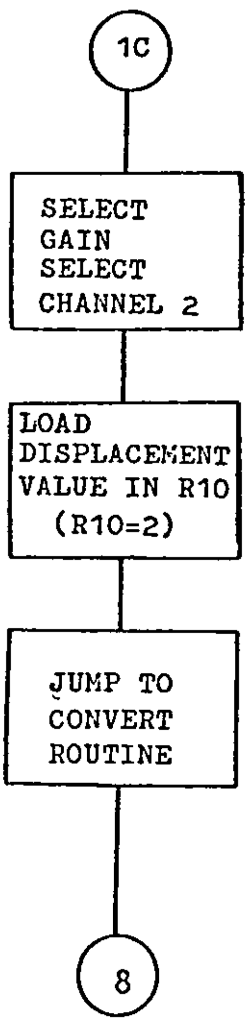
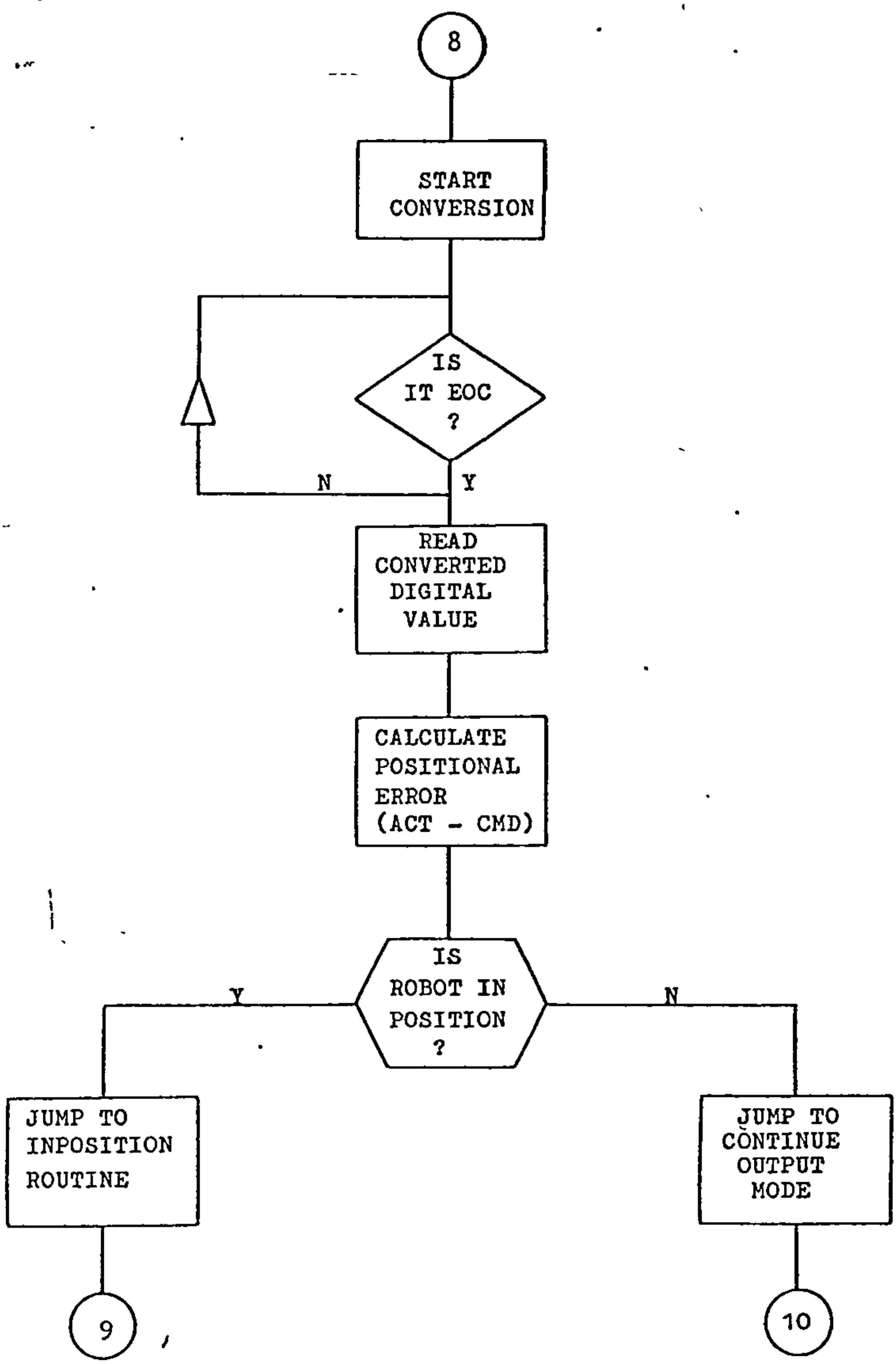


Figure 6.19
Convert Analog Feedback &
Calculate Positional Error



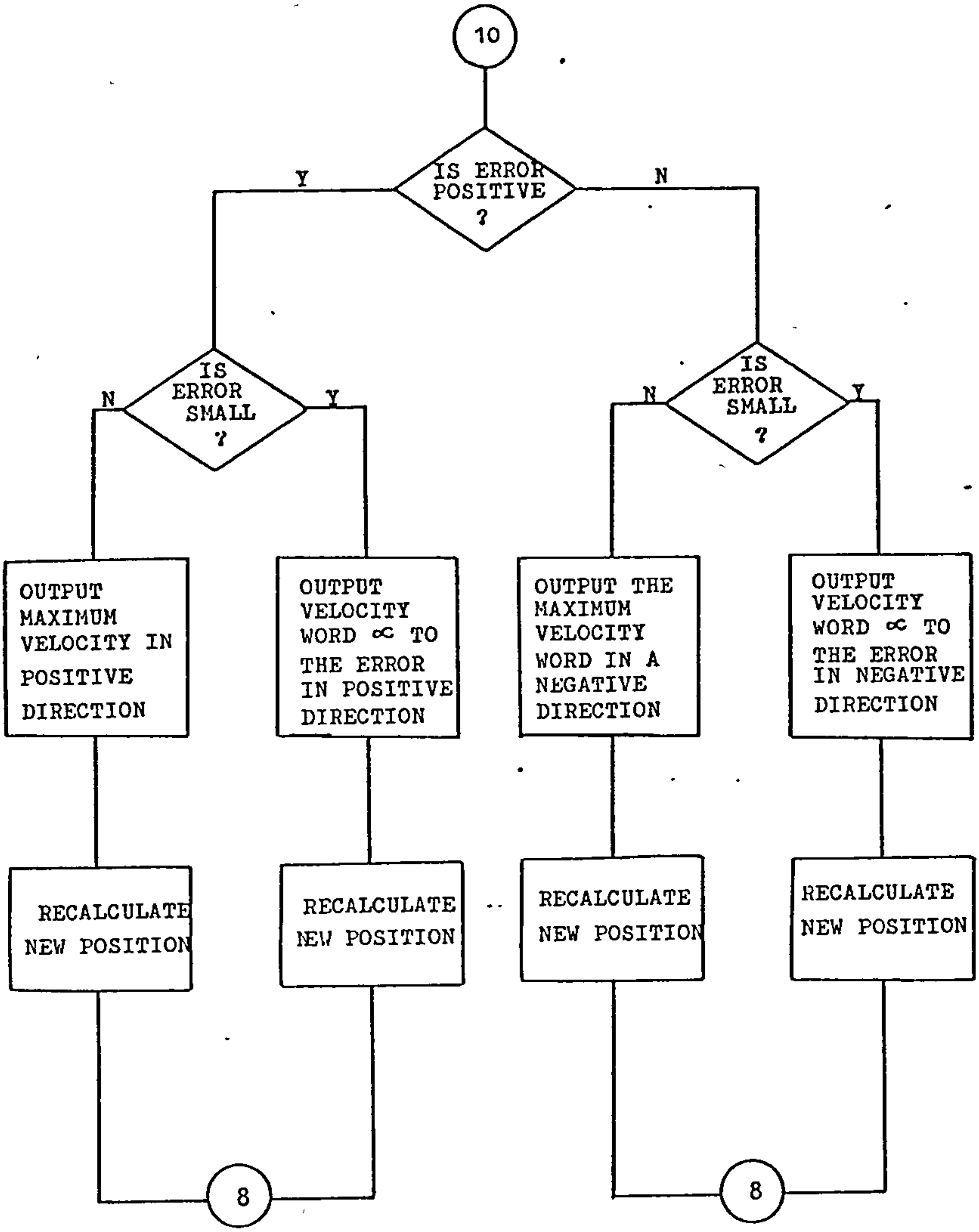


Figure 6.19 I/O Analog - Digital Routine - Continue Mode

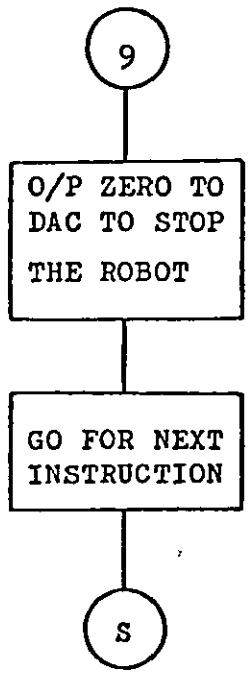


Figure 6.19 Robot Inposition Routine

Continue Routine

In this routine, the modifier of the instruction, contained in Register 3, determines the route taken by the program (see fig 6.20). If the modifier contains 0000, the instruction is interpreted as a 'continue cycling until stopped'. If it contains 0001, this indicates that the last cycle has been reached and the program branches to the stop routine.

Each time the sub-routine is entered the modifier is decremented by 'one', the new instruction is then formed by adding the 'continue' op-code to the new modifier. The instruction pointer is placed one memory location back (as it automatically increments to the next instruction in sequence) and the new instruction is loaded into user memory.

Stop Routine

When the program enters the stop routine, a prompt is issued to the operator via the VDU/Teletype 'YOUR PROGRAM IS COMPLETE' (see fig 6.21). Control is then returned to TIBUG MONITOR, either for the program to be re-executed, or to permit changes to the program.

Jump Routine

Here the modifier of the instruction is changed as the value input is the number of robot instructions to be omitted produce the jump in the number of bytes (see fig 6.22). This is achieved using SLA R3, 1, which shifts left by one position the contents of Register 3, which effectively doubles the value. The instruction pointer is then modified and the number of bytes to be jumped is added. The program then goes for the next instruction.

Wrist Move Routine

This is a routine that sets the base of address of the 9901 chip. A one or zero is then written to bit 2 of the I/O pins (see fig 6.23). A nought turns the wrist vertical, a 'one' turns the wrist to a horizontal position. The program then jumps to the real time delay routine.

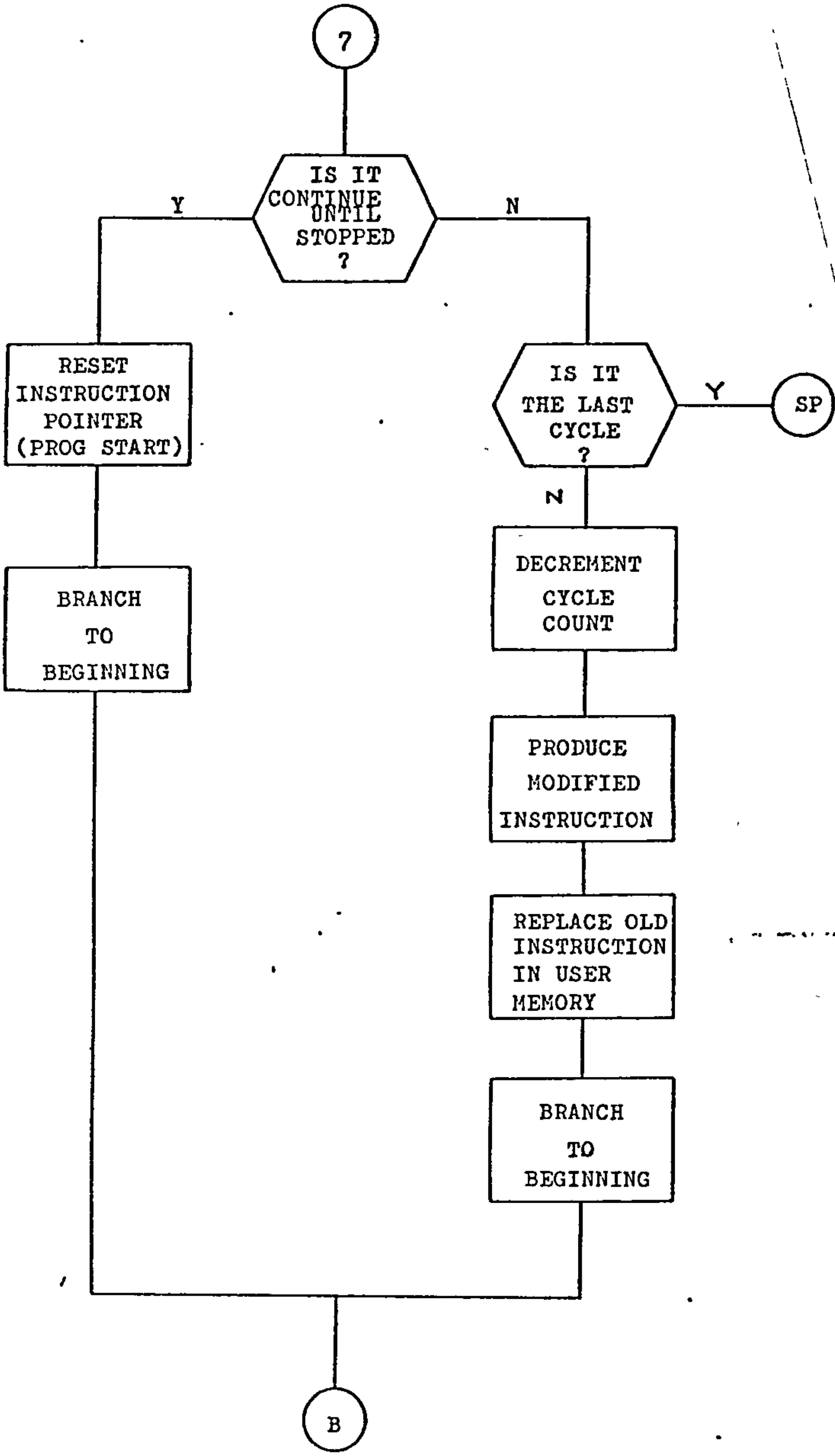


Figure 6.20 Continue Cycle Routine

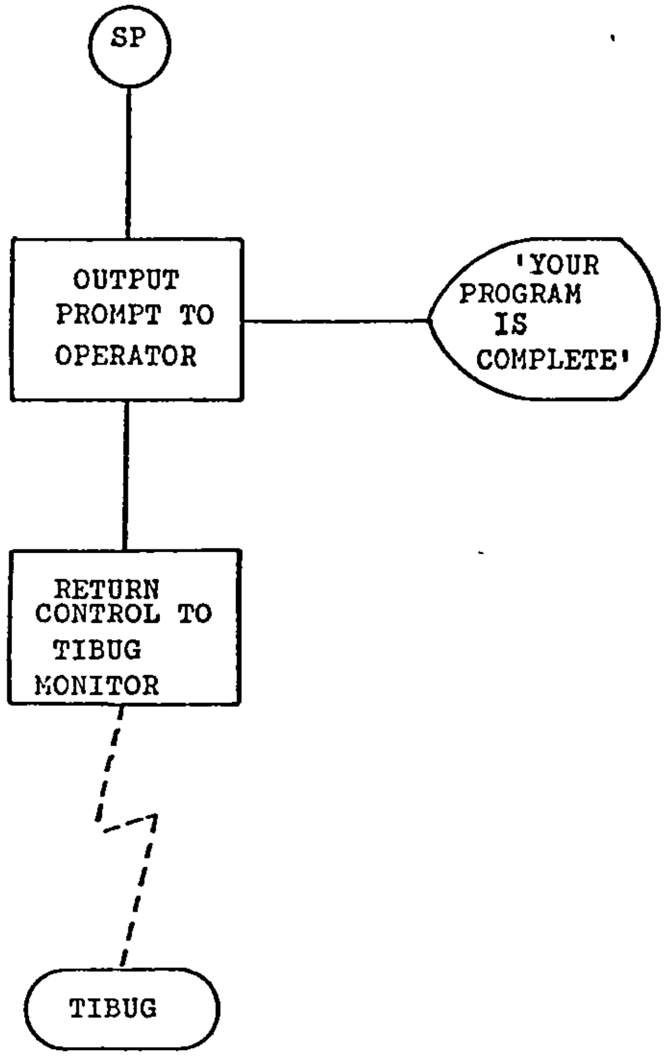


Figure 6.21 Stop Routine

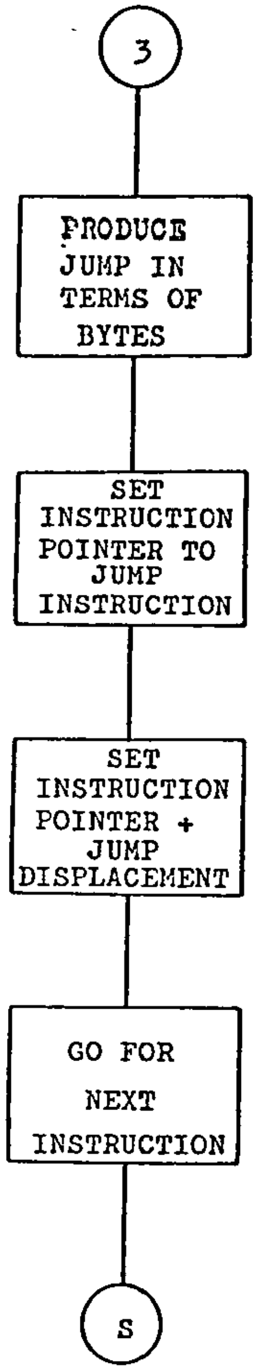


Figure 6.22 Jump Routine

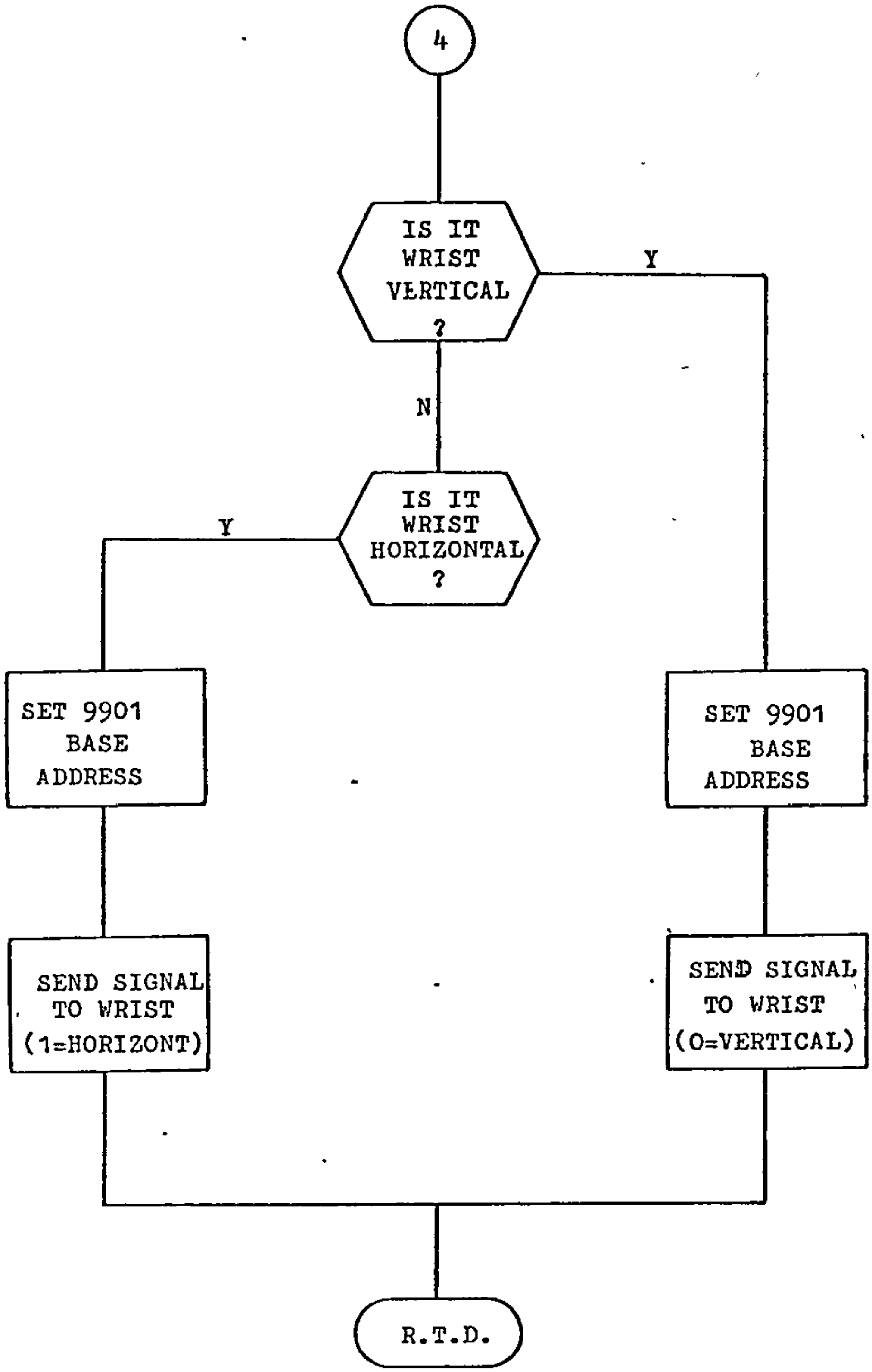


Figure 6.23 Move Wrist Routine

Gripper Open/Close Routine

This routine again loads the base address of the 9901 chip, it then writes either a one or a nought to bit 4 of the I/O pins one to open the gripper, nought to close the gripper (see figure 6.24). The program then branches to the real time delay routine. This real time delay is routine only enterable after a move wrist or gripper instruction. It produces a delay loop in the program to allow the robot to respond to the signal before the next instruction is read and implemented.

Time Delay Routine

In this instruction, the modifier contains the number of quarter seconds time delay required. The base address is set up in Register 12 of the main program workspace.(figure 6.25). Register 0 of the timer service routine (which is entered when an interrupt 3 occurs) workspace is cleared and an interrupt 3 is enabled. The number of quarter seconds are then copied into Register 1 of the timer service routine. The timer is then started for a single count and the program idles until interrupted by an interrupt 3.

Once an interrupt 3 is received by the microprocessor, the program jumps to pick up the interrupt 3 vectors. These vectors are located at memory address >000C (workspace pointer) and >000E (program counter). The program then jumps to the location indicated by the program counter, namely memory address >FFAA. At memory location >FFAA the program reads a branch instruction, and branches to the timer service routine.

Timer Service Routine

In this routine, each time it is entered a check is made to see if the delay is finished (see fig 6.25). A count of each quarter second is also incremented. The time is then loaded with the value to produce a quarter second count and the program jumps back to the time delay routine, where it idles until another interrupt is received.

If the time delay is completed when the routine is entered, the program then clears the cycle counter and also clears Register 15

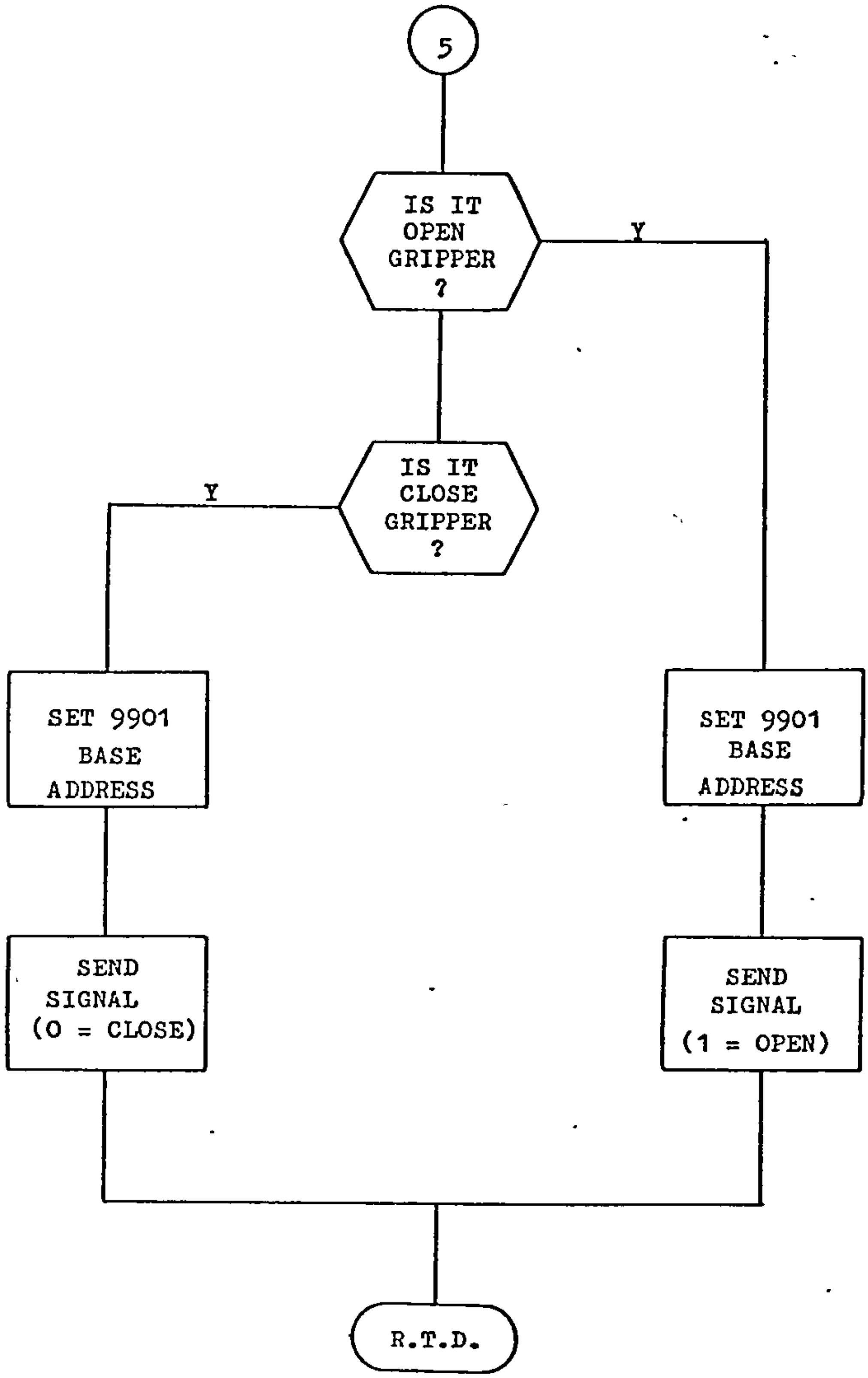
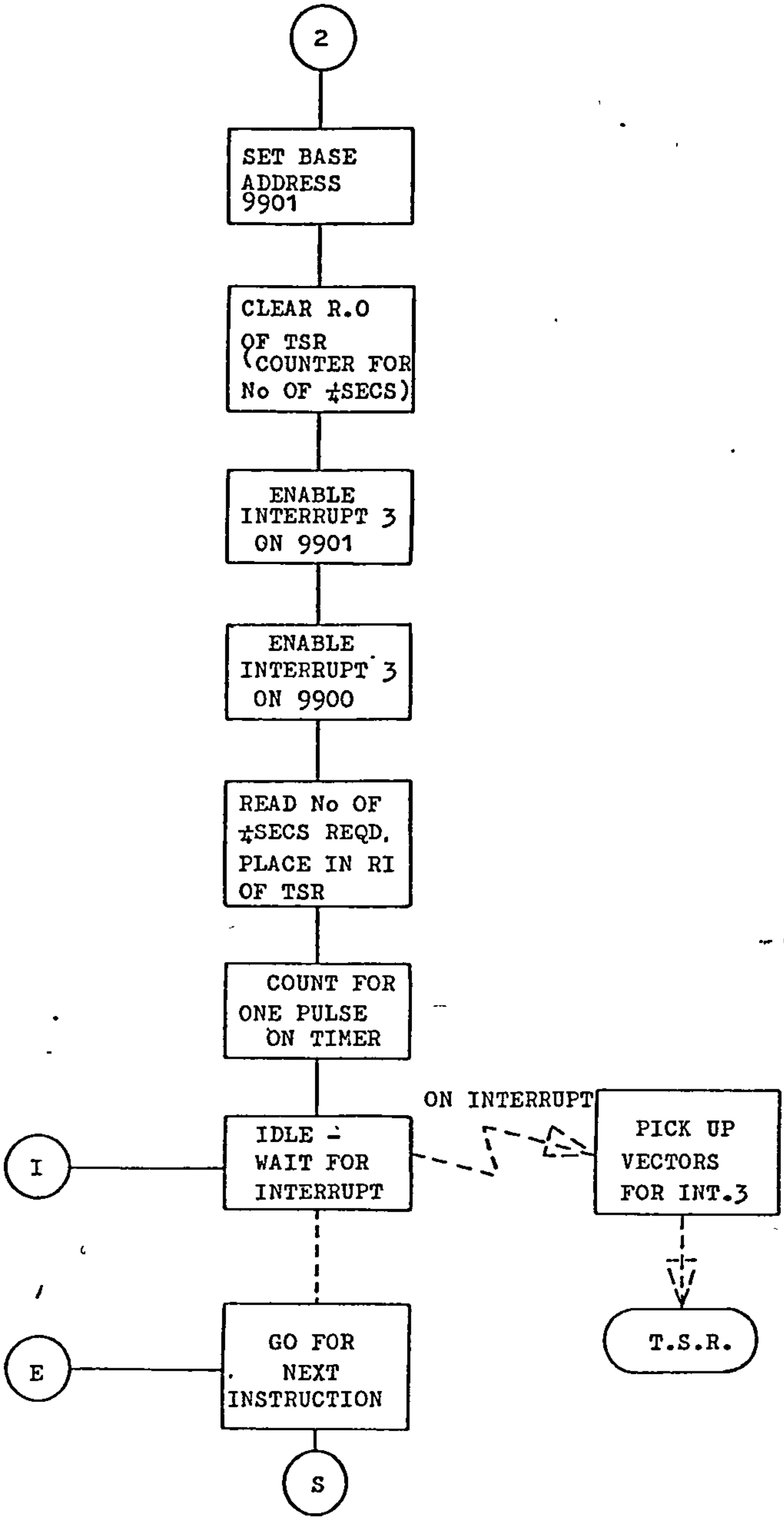
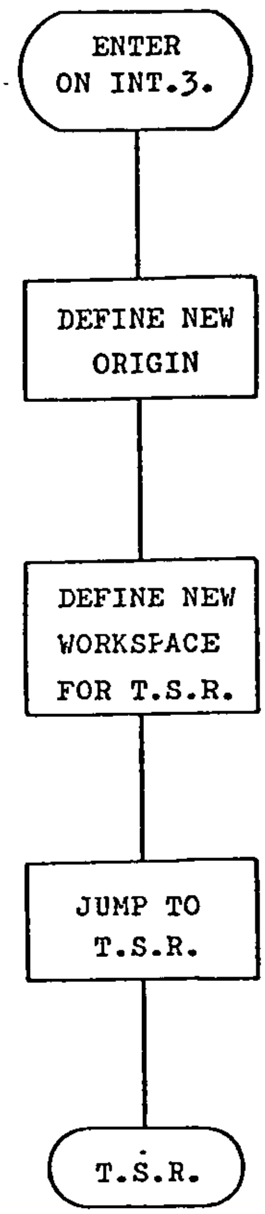


Figure 6.24 Open & Close Gripper Routine

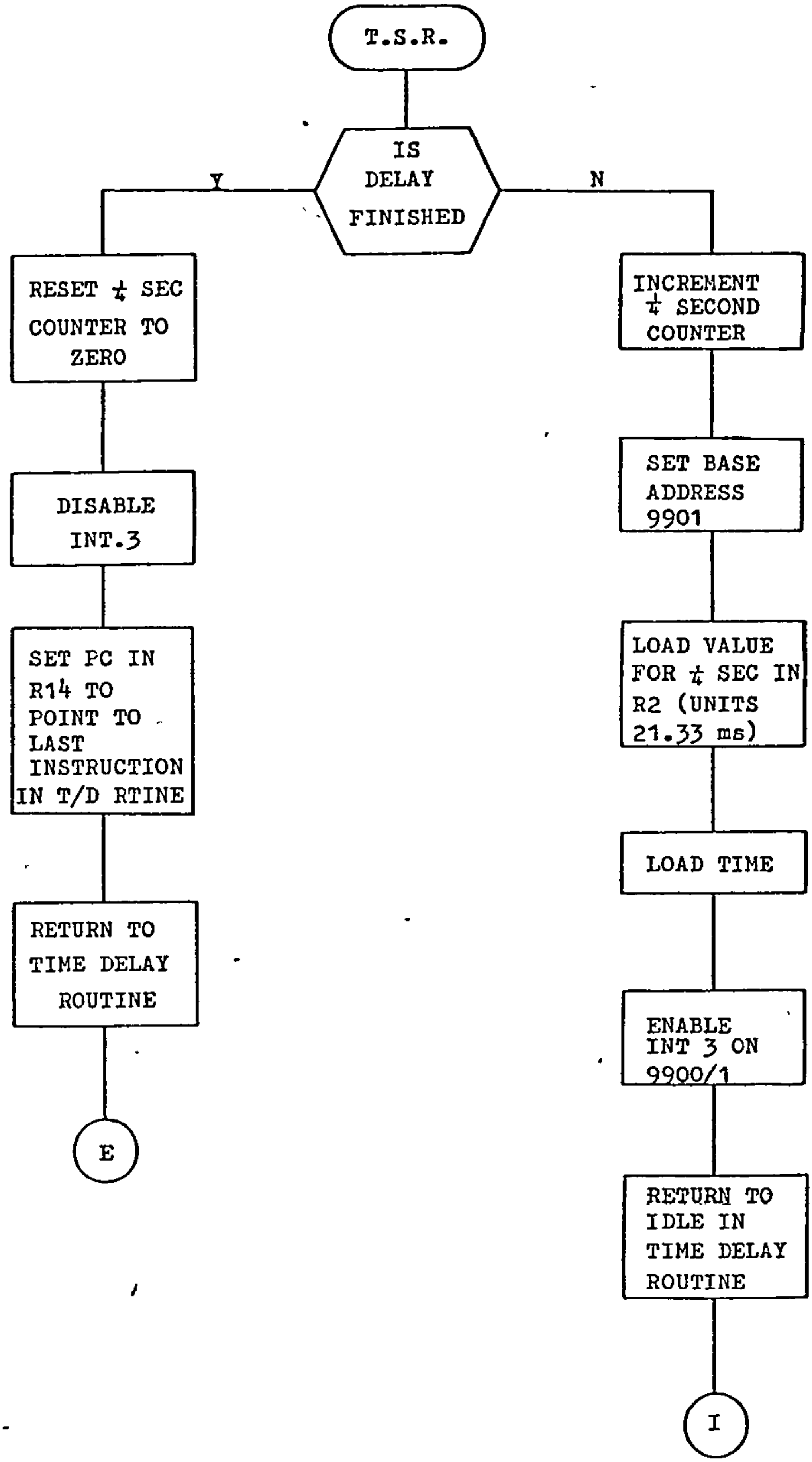
Figure 6.25 Time Delay Routine



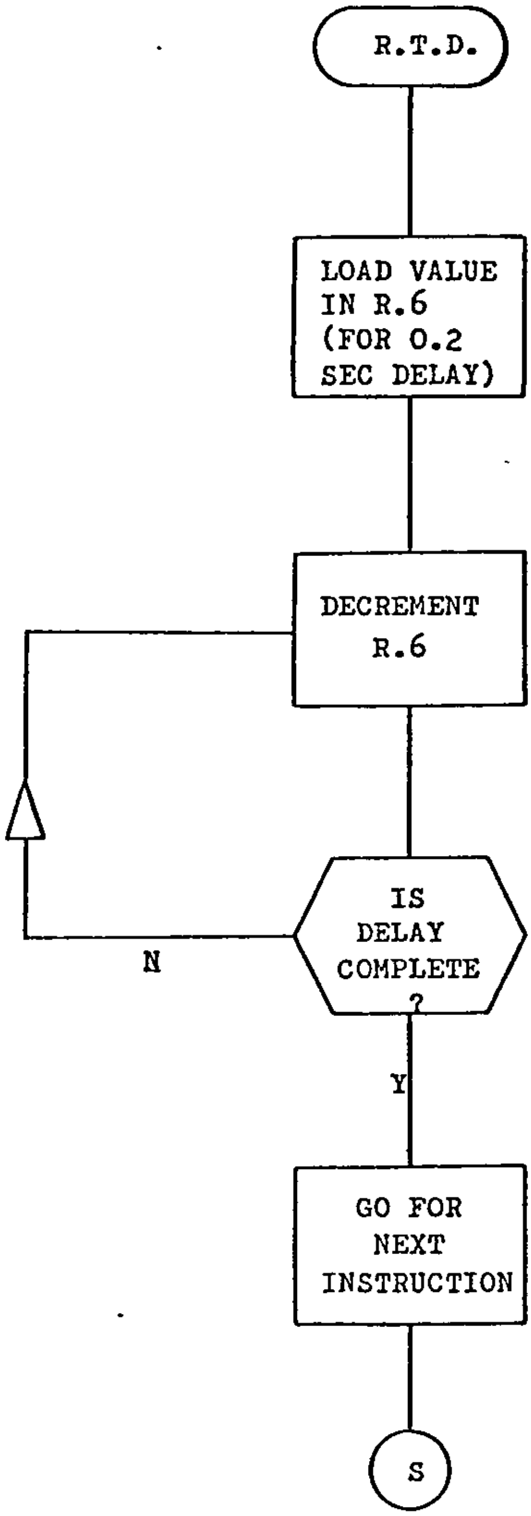
Reserve Memory Space for TSR



Timer Service Routine (TSR)



Real Time Delay Routine



so that on returning to main program, the interrupt mask is cleared. The program then adjusts the program counter so that a jump for a new instruction is initiated.

6.5 PROGRAMS USING CONTINUOUS CLOSED-LOOP CONTROL

During the operation of program VERTHREE as the axes are moved one at a time, when the first was in position and the second was being moved the first one may drift unless closed-loop control can be accomplished irrespective of a programmed move on each axis. To overcome this and also to give a constant sampling rate all the axes were moved simultaneously and were sampled at a controlled rate using the program detailed in the next section.

6.5.1 Program INT1

The flow charts which illustrate the operation of this program are shown in figures 6.26 and 6.27. The values for the positions are stored as shown below.

memory location		
FBØØ	AXIS 1	FIRST POSITION
FBØ2	AXIS 2	
FBØ4	AXIS 3	
FBØ6	AXIS 1	SECOND POSITION
FBØ8	AXIS 2	
FBØA	AXIS 3	
		etc

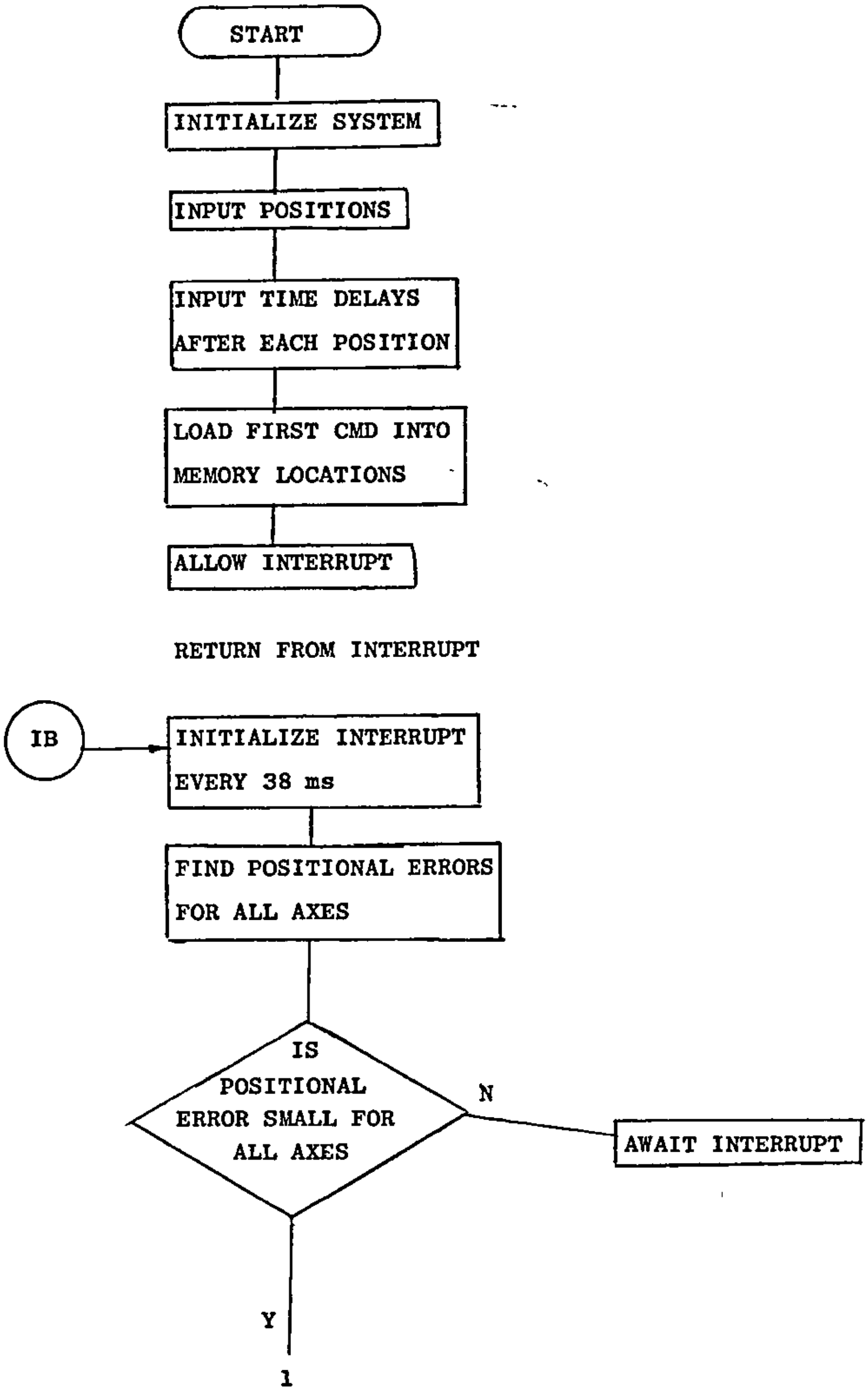
The delay times at the end of each position are also stored in another table commencing at >FD2Ø. Register 9 in the main workspace is used to store the 'time' of the current delay.

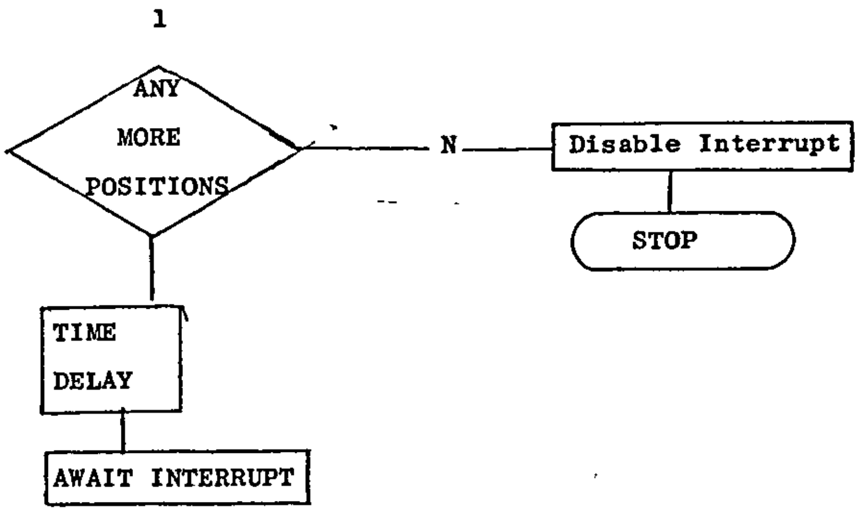
The CMD positions are moved to the memory locations >FDØ6,>FDØ8,>FDØA by repeating the following statement three times

```
MOV *R4+,*R5+
```

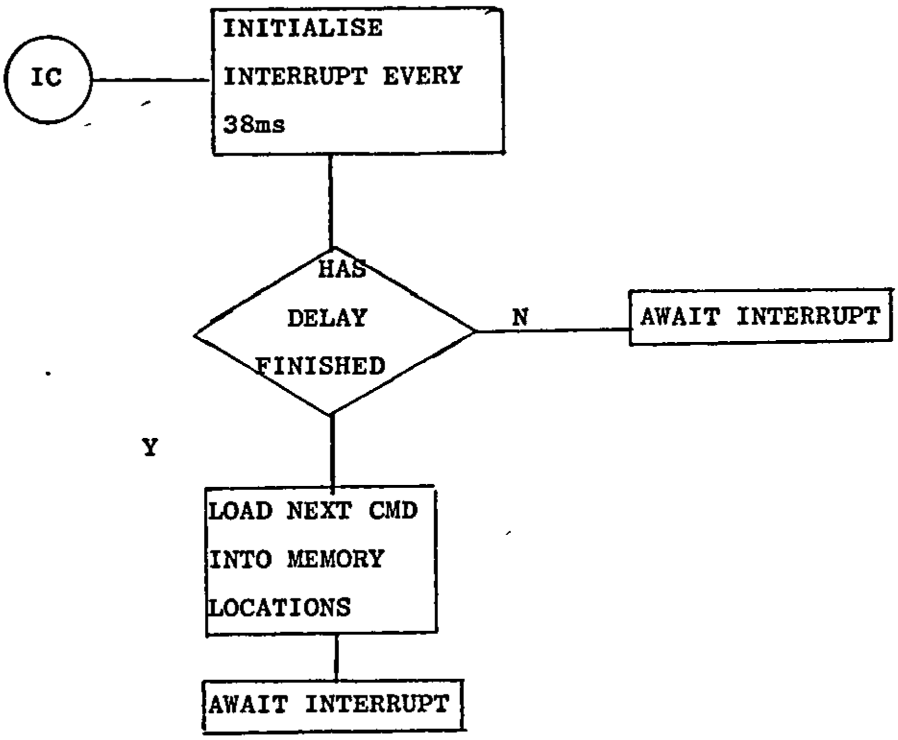
The interrupt 3 is then initialised and the count is 1 and so the interrupt occurs after one count. The new PC and WP are picked up by the following procedure, the interrupt vectors are blown in EPROM

Figure 6.26 Overall flow chart for INT 1





RETURN FROM INTERRUPT



INTERRUPT SERVICE ROUTINE

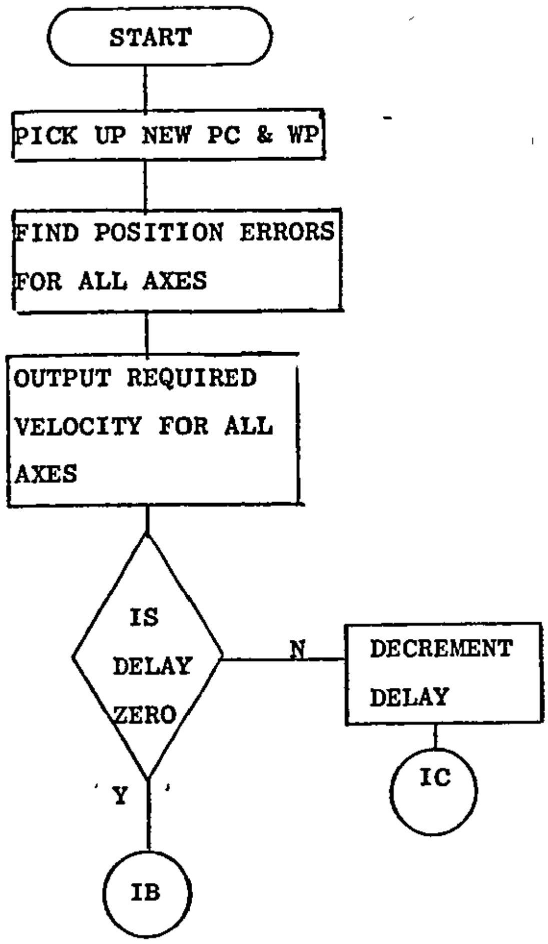
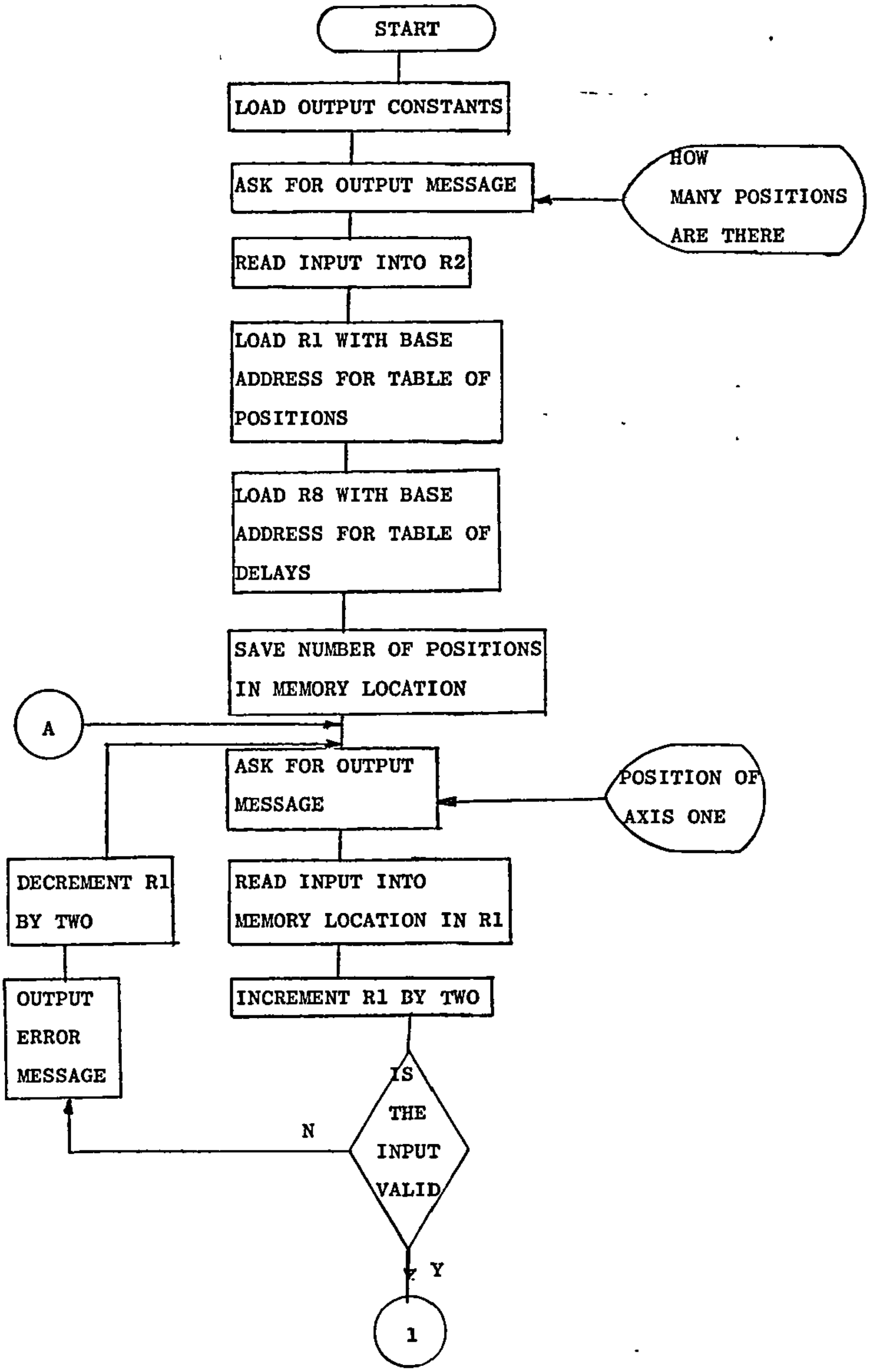
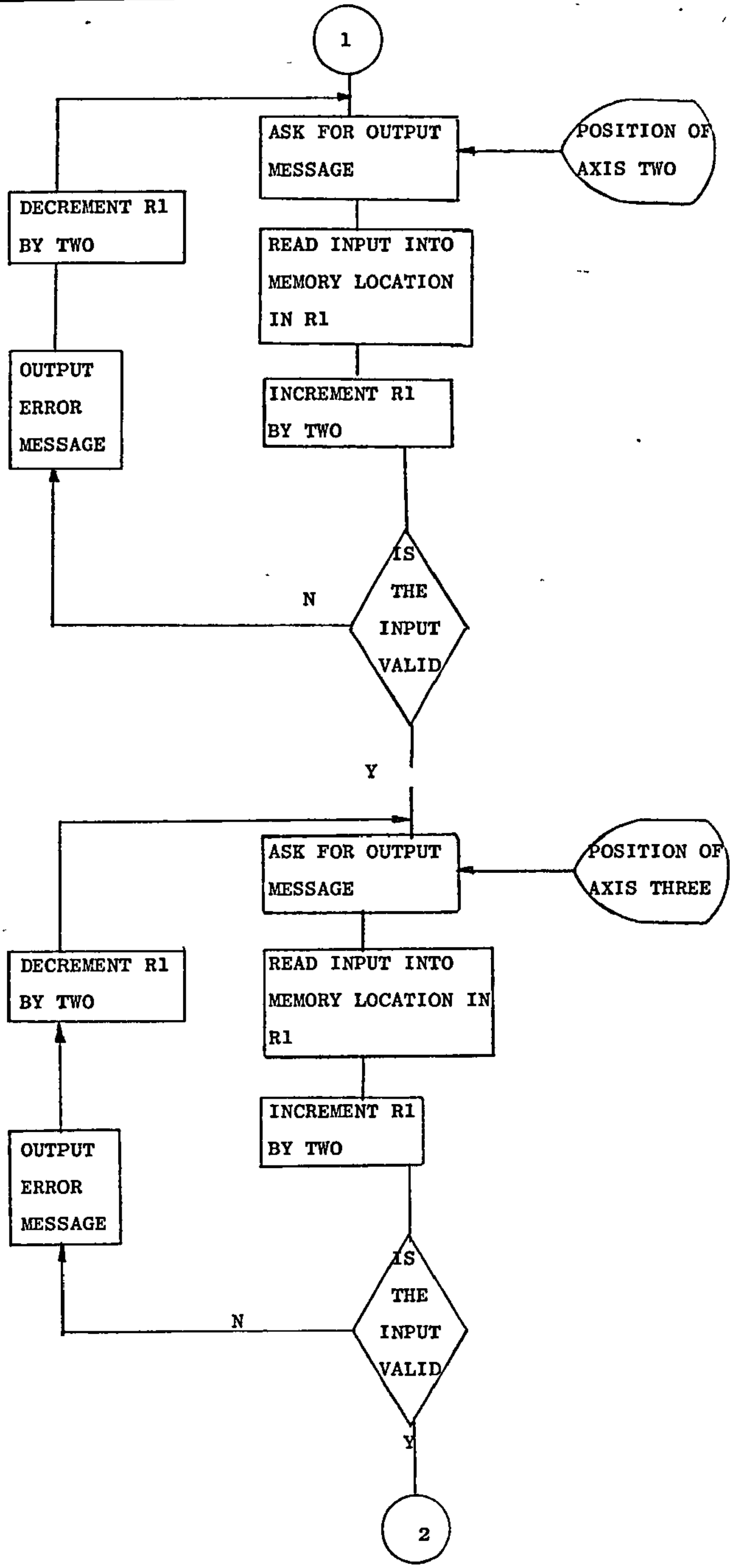
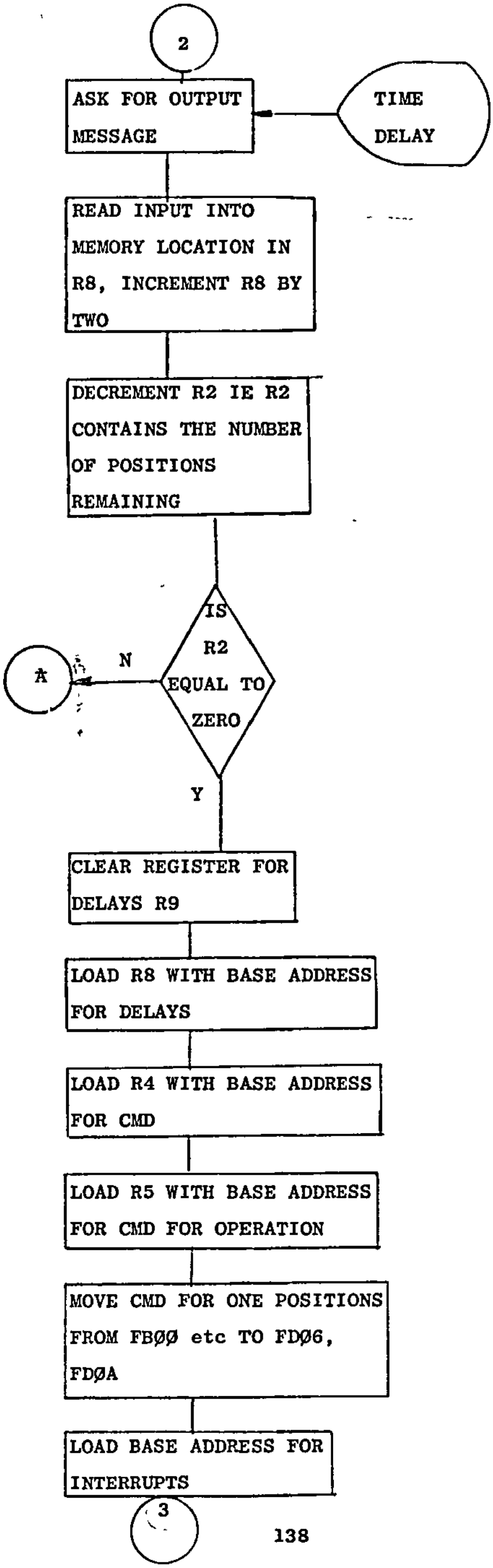
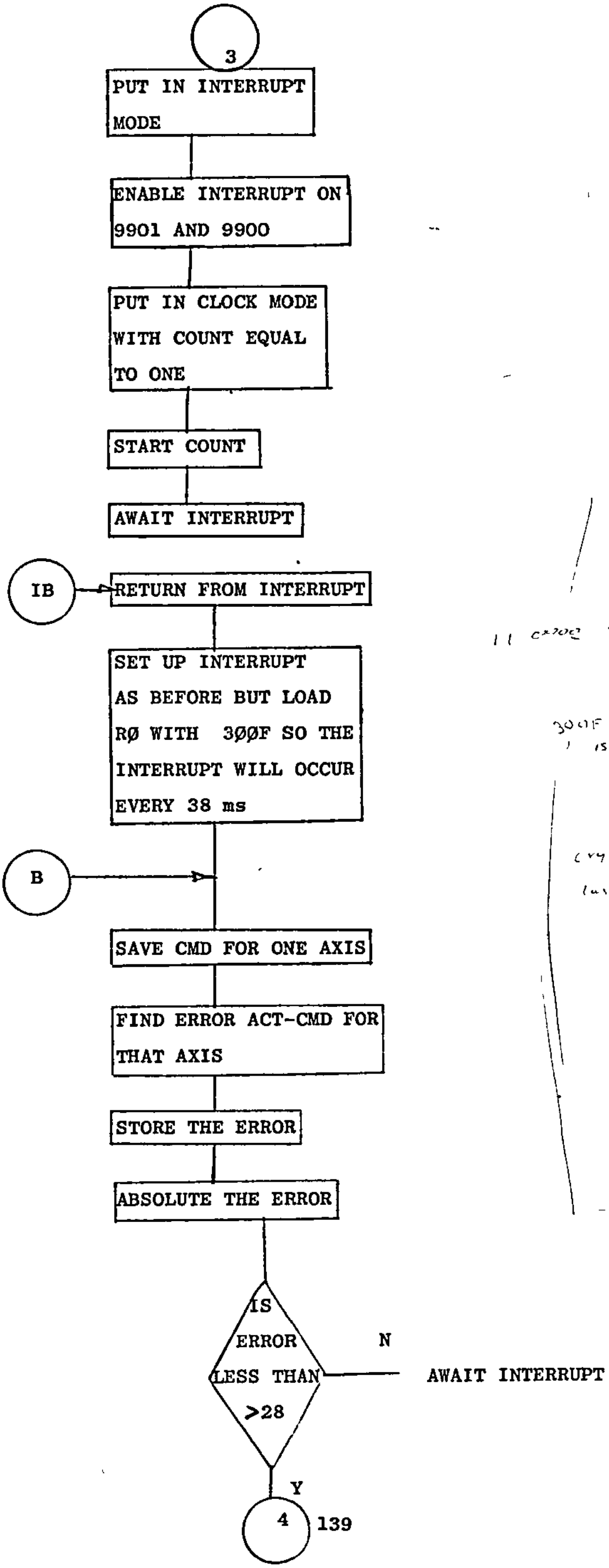


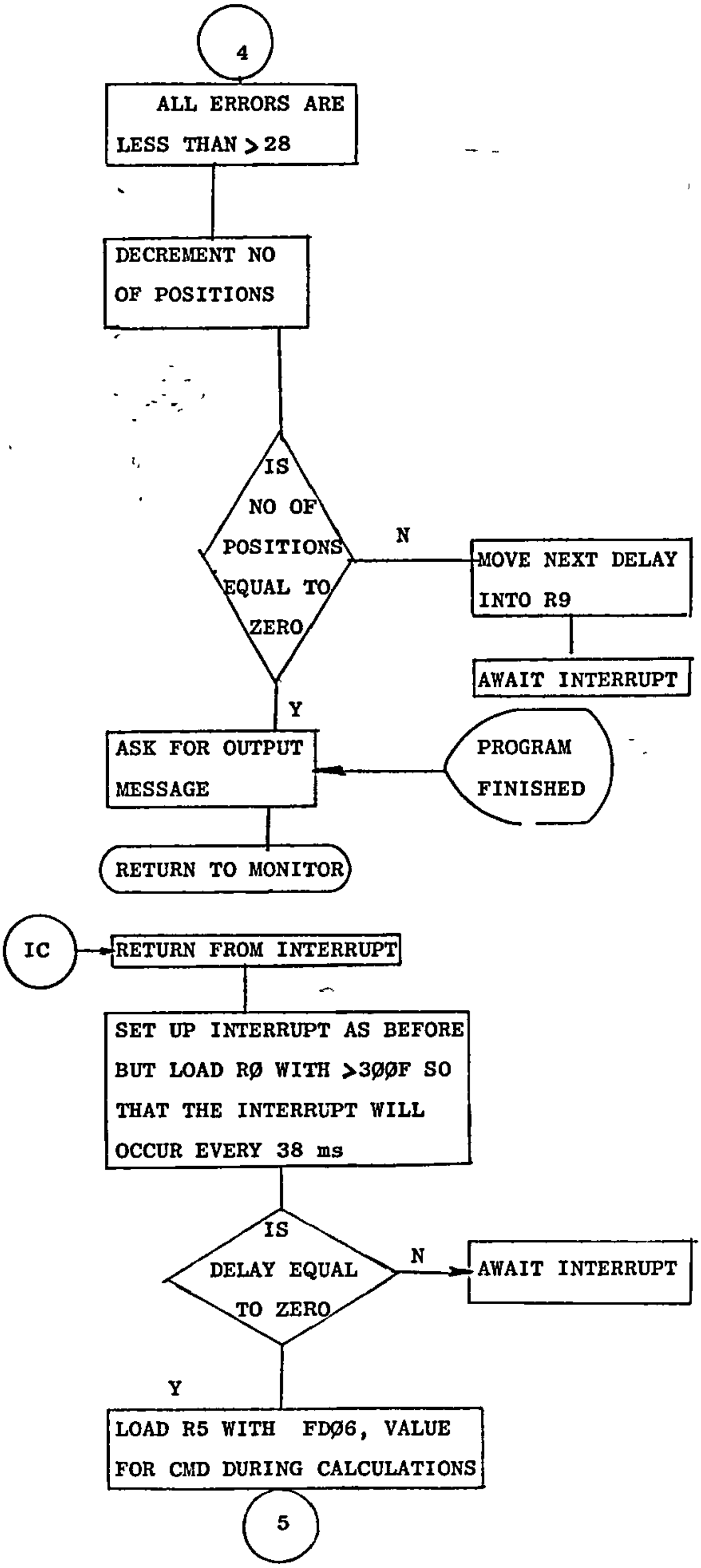
Figure 6.27 Detailed Flow Chart for INT 1

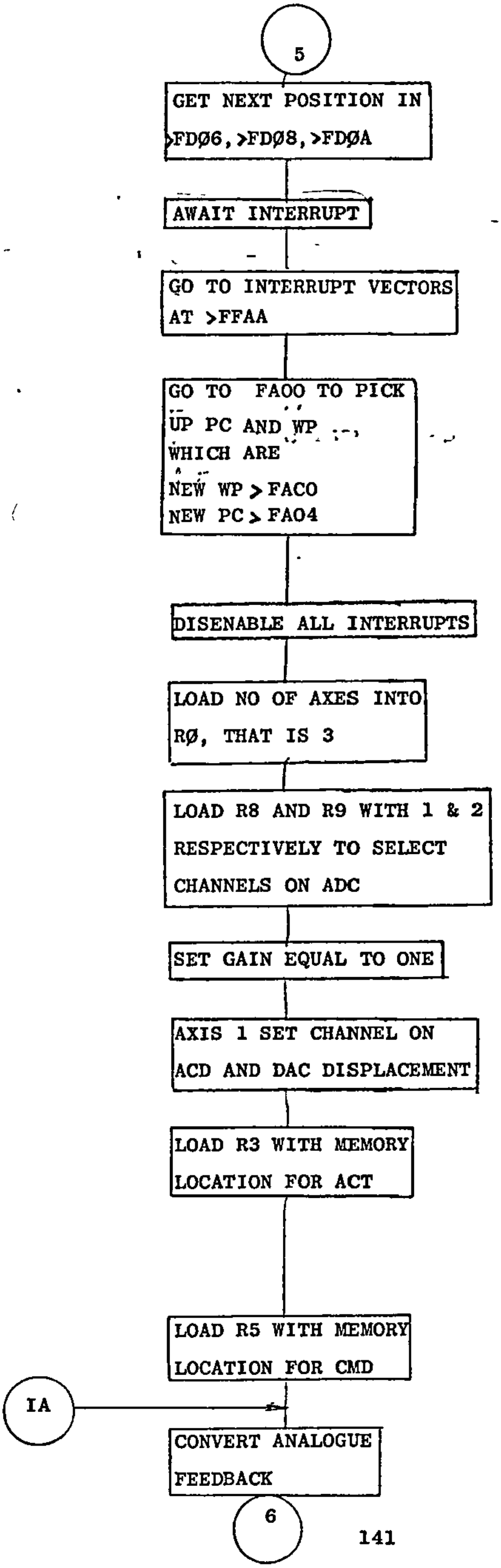


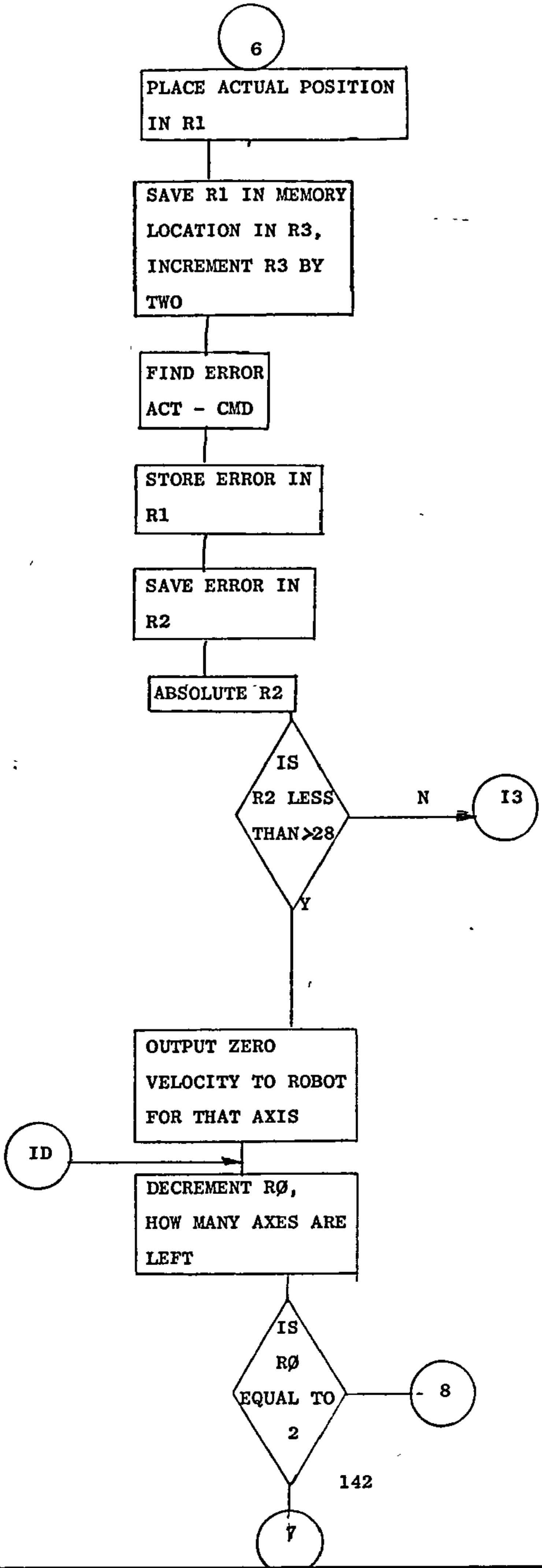

1
ASK FOR OUTPUT MESSAGE
POSITION OF AXIS TWO
DECREMENT R1 BY TWO
READ INPUT INTO MEMORY LOCATION IN R1
OUTPUT ERROR MESSAGE
INCREMENT R1 BY TWO
N
IS THE INPUT VALID
Y
ASK FOR OUTPUT MESSAGE
POSITION OF AXIS THREE
DECREMENT R1 BY TWO
READ INPUT INTO MEMORY LOCATION IN R1
INCREMENT R1 BY TWO
OUTPUT ERROR MESSAGE
N
IS THE INPUT VALID
Y
2

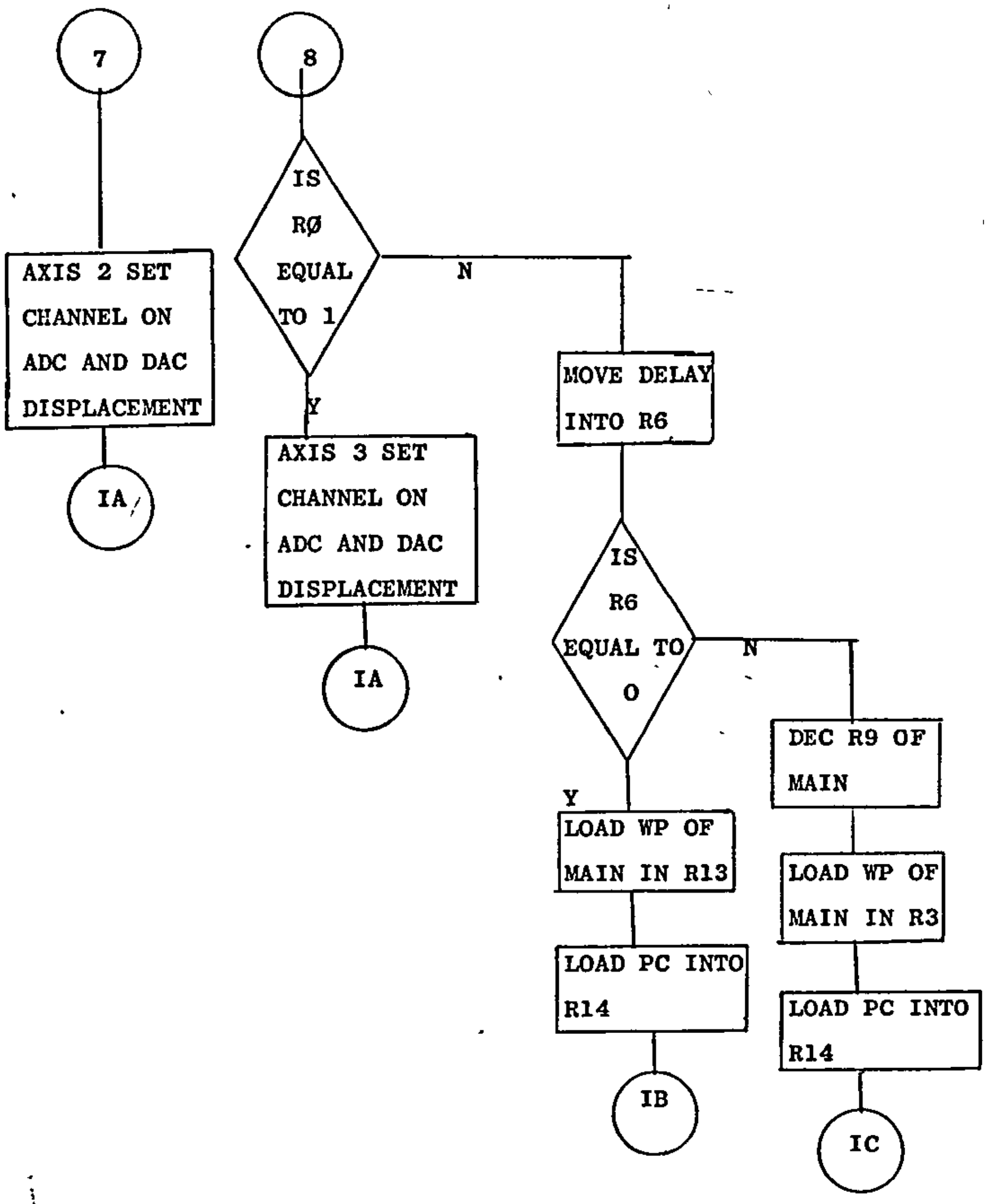

2
ASK FOR OUTPUT MESSAGE
TIME DELAY
READ INPUT INTO MEMORY LOCATION IN R8, INCREMENT R8 BY TWO
DECREMENT R2 IE R2 CONTAINS THE NUMBER OF POSITIONS REMAINING
IS R2 EQUAL TO ZERO
N
A
Y
CLEAR REGISTER FOR DELAYS R9
LOAD R8 WITH BASE ADDRESS FOR DELAYS
LOAD R4 WITH BASE ADDRESS FOR CMD
LOAD R5 WITH BASE ADDRESS FOR CMD FOR OPERATION
MOVE CMD FOR ONE POSITIONS FROM FBØØ etc TO FDØ6, FDØA
LOAD BASE ADDRESS FOR INTERRUPTS
3

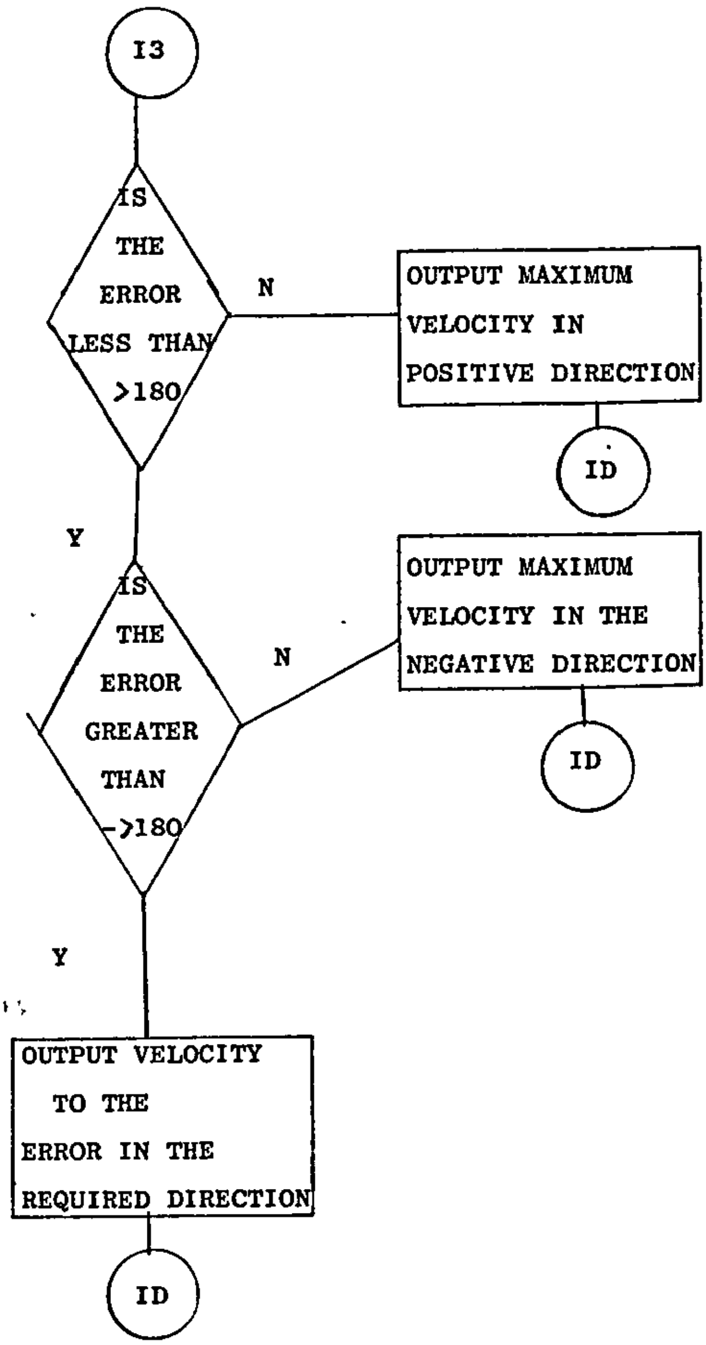

3
PUT IN INTERRUPT MODE
ENABLE INTERRUPT ON 9901 AND 9900
PUT IN CLOCK MODE WITH COUNT EQUAL TO ONE
START COUNT
AWAIT INTERRUPT
IB
RETURN FROM INTERRUPT
SET UP INTERRUPT AS BEFORE BUT LOAD RØ WITH 3ØØF SO THE INTERRUPT WILL OCCUR EVERY 38 ms
B
SAVE CMD FOR ONE AXIS
FIND ERROR ACT-CMD FOR THAT AXIS
STORE THE ERROR
ABSOLUTE THE ERROR
IS ERROR LESS THAN >28
N
AWAIT INTERRUPT
Y
4


4
ALL ERRORS ARE LESS THAN >28
DECREMENT NO OF POSITIONS
IS NO OF POSITIONS EQUAL TO ZERO
N
MOVE NEXT DELAY INTO R9
AWAIT INTERRUPT
Y
ASK FOR OUTPUT MESSAGE
PROGRAM FINISHED
RETURN TO MONITOR
IC
RETURN FROM INTERRUPT
SET UP INTERRUPT AS BEFORE BUT LOAD RØ WITH >3ØØF SO THAT THE INTERRUPT WILL OCCUR EVERY 38 ms
IS DELAY EQUAL TO ZERO
N
AWAIT INTERRUPT
Y
LOAD R5 WITH FDØ6, VALUE FOR CMD DURING CALCULATIONS
5


5
GET NEXT POSITION IN >FDØ6,>FDØ8,>FDØA
AWAIT INTERRUPT
GO TO INTERRUPT VECTORS AT >FFAA
GO TO FA00 TO PICK UP PC AND WP WHICH ARE
NEW WP > FAC0
NEW PC > FA04
DISENABLE ALL INTERRUPTS
LOAD NO OF AXES INTO RØ, THAT IS 3
LOAD R8 AND R9 WITH 1 & 2 RESPECTIVELY TO SELECT CHANNELS ON ADC
SET GAIN EQUAL TO ONE
AXIS 1 SET CHANNEL ON ACD AND DAC DISPLACEMENT
LOAD R3 WITH MEMORY LOCATION FOR ACT
LOAD R5 WITH MEMORY LOCATION FOR CMD
IA
CONVERT ANALOGUE FEEDBACK
6


6
PLACE ACTUAL POSITION IN R1
SAVE R1 IN MEMORY LOCATION IN R3, INCREMENT R3 BY TWO
FIND ERROR ACT - CMD
STORE ERROR IN R1
SAVE ERROR IN R2
ABSOLUTE R2
IS R2 LESS THAN >28
N
13
Y
OUTPUT ZERO VELOCITY TO ROBOT FOR THAT AXIS
ID
DECREMENT RØ, HOW MANY AXES ARE LEFT
IS RØ EQUAL TO 2
8
7


7
8
AXIS 2 SET CHANNEL ON ADC AND DAC DISPLACEMENT
IA
IS RØ EQUAL TO 1
N
Y
AXIS 3 SET CHANNEL ON ADC AND DAC DISPLACEMENT
IA
MOVE DELAY INTO R6
IS R6 EQUAL TO O
N
Y
DEC R9 OF MAIN
LOAD WP OF MAIN IN R13
LOAD PC INTO R14
IB
LOAD WP OF MAIN IN R3
LOAD PC INTO R14
IC



I3
IS THE ERROR LESS THAN >180
N
OUTPUT MAXIMUM VELOCITY IN POSITIVE DIRECTION
ID
Y
IS THE ERROR GREATER THAN ->180
N
OUTPUT MAXIMUM VELOCITY IN THE NEGATIVE DIRECTION
ID
Y
OUTPUT VELOCITY TO THE ERROR IN THE REQUIRED DIRECTION
ID

and at these vectors there is a branch statement which goes to another part of the program to pick up the WP and PC. The execution of the interrupt service routine then commences. First the interrupts are all disenabled by using LIMI Ø. The number of axes is loaded into RØ which in this case is three. The gain is set to unity and R8 and R9 contain the numbers 1 and 2 respectively which are used to select the required channel on the ADC. The ADC channel is selected and displacement for the DAC is loaded into R10 for the first axis. The memory location for the ACT, this is>FDØØ is stored in R3 and the memory location for the CMD which is>FDØ6 is stored in R5. The conversion of the analogue voltage then takes place and the value is stored in R1. It is then copied into the memory location in R3, R3 is then increased by 2 using the instruction

```
MOV R1, *R3+
```

the error is calculated using the command,

```
S *R5+, R1
```

autoincrement addressing is utilised. The error is then saved in R2 where the modulus is found. This time the acceptable tolerance of the error is allowed to be within >28 of the CMD. (This is equivalent to a positional error of 1.37%). This value was chosen as it was a long time for all the axes to be exactly in their correct positions. The correct voltage is output via the DAC as described earlier. The TEST subroutine is then entered where the number of axes still to be 'serviced' is deduced. When all the axes have been serviced the delay is examined. The instruction

```
MOV @>F812,R6
```

moves R9 (which stores the time of the delay) into R6, Register 6 is compared to zero and if it is equal, that is, the first delay has not beenreached or the delay has finished then the WP from the main program is loaded into R13 which is >F8ØØ, the PC of the subroutine START is loaded into R14 and the status is cleared. Control then returns to the main program commencing with START. If register 6 is non zero then register 9 from the main program is decremented by

DEC @>F812

and then the WP from the main program is loaded into R13, the PC of the subroutine CONT is loaded into R14 and the status is cleared. Control then returns to the main program commencing with CONT.

When START is the return PC the following sequence of events occur. Interrupt 3 is enabled and such an interrupt will occur after 38ms as the number loaded into RØ is >3ØØF. The following then occurs for each axis in turn. The CMD value is moved from its memory location into a register, the error is then calculated and stored in the same register, the modulus of the contents of this register is found and then compared with the value or >28. If the value is greater then the statement

JMP SELF

is executed which awaits the interrupt, if it is smaller then the error of the next axis is examined in the same way. If all three errors are less than 28 then one required position has been reached. The number of positions remaining is decremented, if it is zero then the STOP subroutine is executed. All interrupts are disenabled and a message is printed out to say that the program has finished and the control is returned to the monitor and the hydraulics then should be de-activated. If there are more positions remaining the next delay is moved into R9

MOV *R8+,R9

and then the interrupt is awaited.

When CONT is the return PC the following sequence occurs. The interrupt 3 is enabled and will occur after 38ms. The value of the time delay in R9 is compared to zero, if it is not equal, that is the delay has not yet finished the interrupt will be waited for. If the delay has finished the next position required is stored in memory locations >FDØ6, >FDØ8, >FDØ8 and then the interrupt will be awaited.

6.5.2 Program INT2

This program is an extension of INT1 which moves the jaws during the time delay (see fig 6.28). The value of 1 is input if the jaws are to be opened and 2 for closed. These values are entered in a table in memory commencing at memory location >FDAØ. The jaws are opened in the initial sequence just before the interrupt is enabled for the first time by using the instructions

```
LI R12,>120      Selects port area
SBZ 12           open jaws
```

After the time delay has been moved to R9 by the instruction

```
MOV *R8+,R9
```

the subroutine to open or close the jaws is executed (a flowchart of this subroutine is shown in figure 6.28). The value at the memory address stored in R10 is compared to the contents R1, (where 1 is stored), if it is equal the jaws are opened if not the jaws are closed and in both cases the interrupt is awaited.

6.5.3 Program INT3 and Data

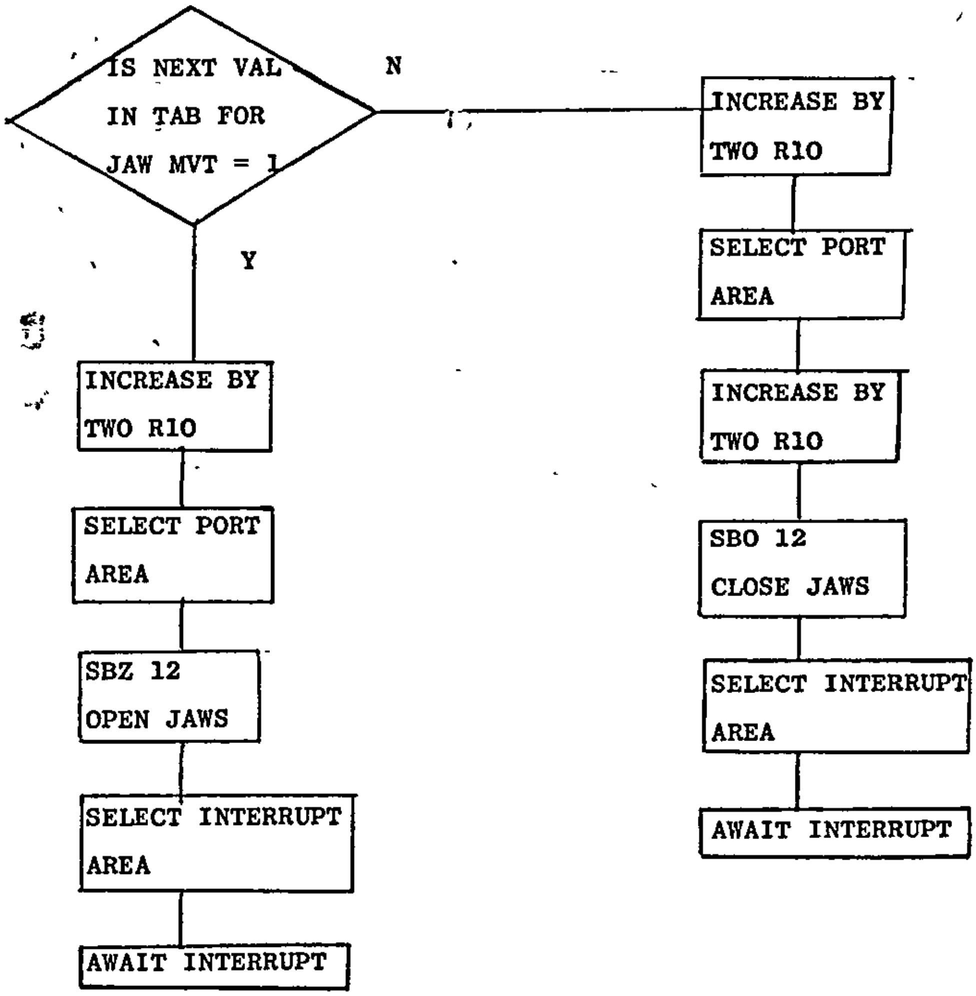
Every time the previous program is run the data has to be entered which is time consuming when the same sequence of operations is to be repeated. This problem can be overcome by dividing the software into two modules, one which is the operator communications module which enters the data and the other a real-time control module which controls the movement of the robot. The program listings are given in Appendix 7.

6.5.4 Program INT4

This program will move the wrist in the swing and jaw motion as well as open or close it during the time delay. The method utilised is the same as in INT2. The listing is given in Appendix 7.

For any of these programs the maximum positive and negative velocity can be chosed just before the program is executed by using an XOP to ask for the values and then reading them into registers and moving

Figure 6.28 Flow Chart to Open/Close Jaws



them to memory locations to be saved.

```
        XOP @VELP,14
NULL    XOP R3, 9
        DATA NULLP
        DATA NULLP
        MOV R3 @>FDØC
        XOP @VELN,14
NULLN   XOP R3,9
        DATA NULLN
        DATA NULLN
        MOV R3, @>FDØE

VELP    DATA >ODOA
        TEXT 'WHAT IS THE MAX +VE VELOCITY'
        BYTE Ø
VELN    DATA >ØDØA
        TEXT 'WHAT IS THE MAX -VE VELOCITY'
        BYTE Ø
```

CHAPTER 7

TESTS OBSERVATIONS AND RESULTS

The idea of testing presupposes the presence of standards against which products are to be tested. An important constituent of a standard is a unit of measurement or framework of measurement universally agreed. It is precisely the absence of these yardsticks that may cause problems for the potential buyer. Unfamiliarity with robots as a product, the difficulties of making realistic comparisons between robots all cause confusion and doubt.[63-66]

7.1 ROBOT TESTING

Robots are purchases for very specific purposes. Some are bought because the purchaser is concerned that their employees are exposed to specifically dangerous or harmful situations eg handling chemicals, welding and forging. Other robots are bought for economic reasons when productivity needs to be increased. Here the purchaser will be concerned that the robot increases flexibility, speed, accuracy or repeatability of the particular process.

When new products of any description are marketed the instinct of the potential buyer will be to try a sample. Although robots are not by any means new, the market is still in the "try-a-sample" stage. There are many unknowns for the purchaser, eg Will the robot be accepted by the labour force? What changes are needed to jigs and fixtures? Will the payback period be short enough? However confident the robot manufacturer is that these problems can be overcome and their particular robot is exactly what the customer requires, the customer has a confidence hurdle to overcome. Much doubt will be concerned with the application. In the majority of cases existing product or process knowledge will be high. Consequently the data gathering will be concerned with judging the effect of the robot on product or process. Occasionally the robot will be the only way to complete the process or product. Whichever the case, the starting point for most buyers is robot manufacturer's data. This is the beginning of a filtering process that may eventually lead to the purchase of a robot. The greater the understanding of the robot imparted by this information and the more empirically valid the data, the more efficient the filtering process will become.

Much of the information about a robot presents no real problem in conceptual terms. A robot will have a certain size and weight. Its working volume can be measured and related to the work to be done. The articulation will make the robot more or less suitable for certain tasks. Its ability to lift a certain load, the way it is motivated, its power requirements, its speeds, and the availability of ancillary equipment are all easily measured using well understood and commonly applied engineering techniques and judgements. Other features pose problems not so easily solved. In a five axis robot working in 3-D space what does an accuracy of $\pm$ 0.5 mm mean? How is repeatability related to accuracy and how is it measured? Does accuracy vary with the load applied or not? How reliable is the software? Here, to a person unfamiliar with robots, there are few common sense measures to be applied. Even to the robot manufacturer there are still some areas that lack definition. All of these problems relate to the sheer complexity of the machine as a series of levers and pivots, bearings and motor sources, measuring and storage systems, programming and operating systems.

Clearly the solution to some of these problems will be overcome during the process of developing a commercial robot. This is particularly true of operational factors. Other problems require deeper thought and are in themselves much more definitive of a system. In particular the very basic measurement of robot accuracy is worth greater consideration.

7.2 CONTROLLER OUTPUT AND SYSTEM RESPONSE ANALYSIS

One method to obtain a measure of the controller output and system response is to utilise facilities already available within the system, ie to use the positional feedback signal of the robot in its analogue voltage form, (the position on any axis being directly proportional to the wiper voltage generated on the specified axis). This system response signal can then be compared directly to output signal from the controller; which can be measured as an analogue voltage by measuring the output from the relevant digital to analogue converter. This signal is amplified before it reaches the servo-valve, however, the amplified signal contains the same characteristics. These signals were recorded using a digital storage oscilloscope (see figure 7.1) and subsequently on an x-y graph plotter. In this way the output signals could both be shown on the same trace. The start and finish position voltages were also recorded for each test carried out. The tests were carried out on the three major axes of the robot for various distances of arm traverse. The results obtained are recorded in Table 7.1.

A description of a typical trace is shown in figure 7.2. The traces are actually plots of voltage versus time, but in the case of the feedback signal, the voltage represents position on the axis and in the case of the output signal from the controller, the voltage is proportional to the velocity of traverse of the robot.

7.3 POSITIONAL REPEATABILITY TESTS

The aim of these tests was to attempt to assess the practical performance of the robot under control of the microprocessor. Obviously to carry out a comprehensive analysis would involve sufficient work for a project within itself, so only one aspect of robot performance was chosen for analysis. Positional repeatability tests were carried out on vertical, swing and horizontal axes of the robot individually. Tests were then carried out with the horizontal and vertical axes combined. A three inch traverse dial indicator gauge and support frame was the only ancillary equipment necessary for these tests.

Four series of tests were carried out on each axis separately, and two series of tests on combined axial motion. The tests on a single axis included the following series of motions of the arm of the robot:-

(i) Extremity to mid-point
(ii) Large distance arm traverse
(iii) Short distance arm traverse
(iv) Arm in central position - mid-point to extremity
(v) Arm in central position - short traverse
(vi) Arm at extremity - mid-point to extremity
(vii) Arm at extremity - short traverse

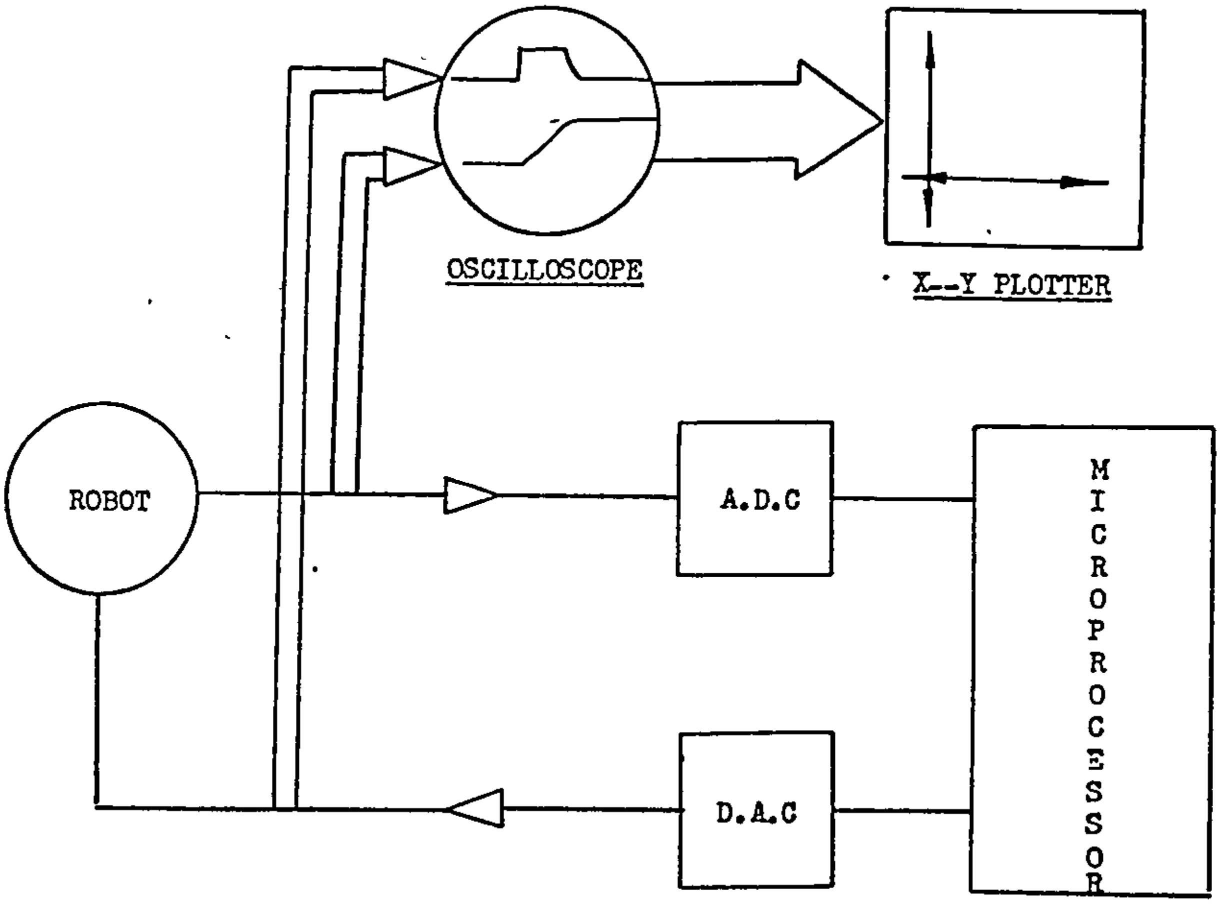


Figure 7.1 Trace Recording System

FIGURE 7.2. DESCRIPTION OF TEST TRACES

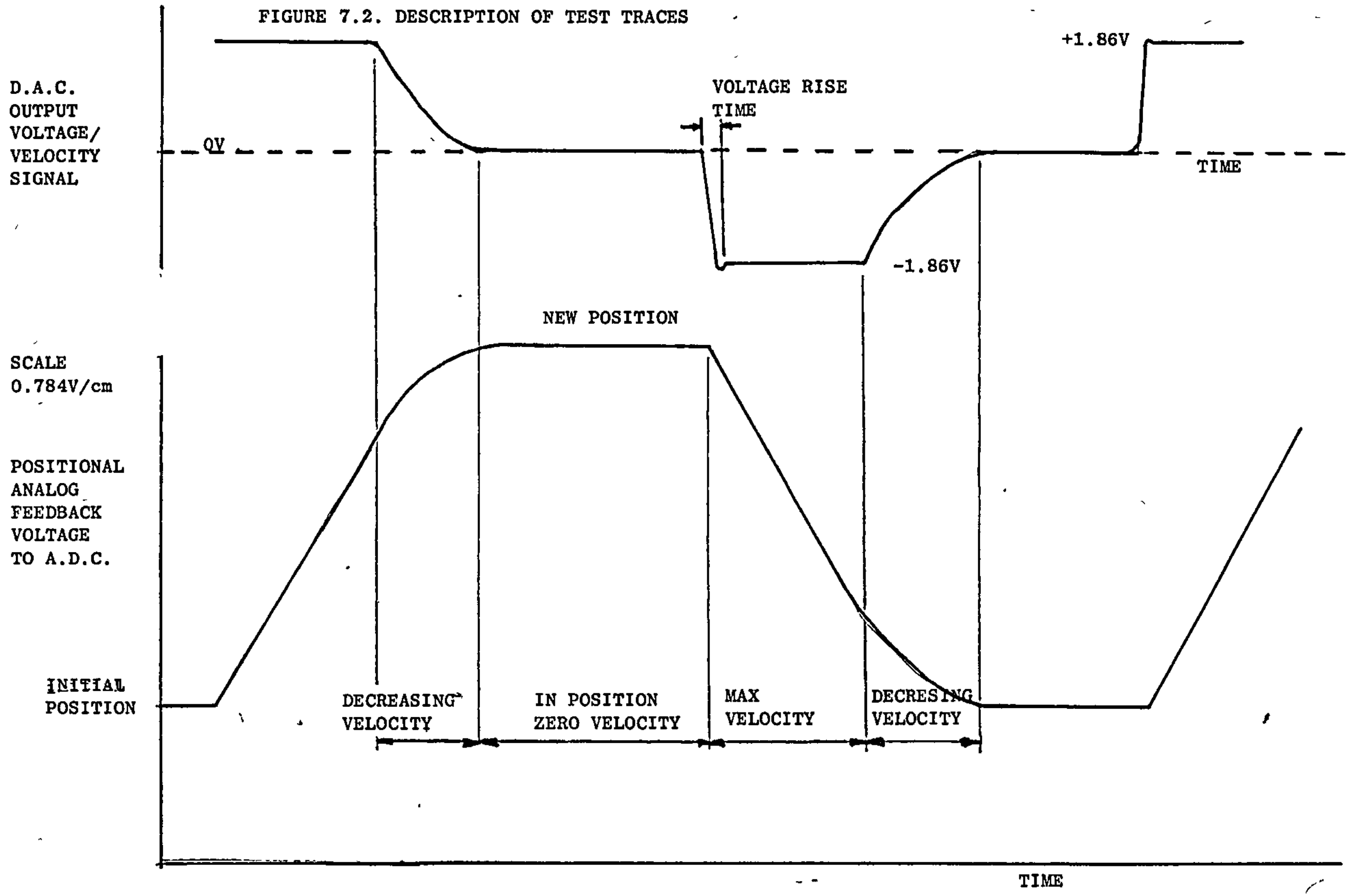


Table 7.1 Output and Response Test Results

AXIS	START LOCATION	FINISH LOCATION	INITIAL VOLTAGE (v)	FINAL VOLTAGE (v)	TEST No
HORIZONTAL	>50	>700	0.5	8.51	1
HORIZONTAL	>100	>650	1.246	7.67	2
HORIZONTAL	>150	>700	1.74	8.5	3
HORIZONTAL	>150	>600	1.74	7.32	4
HORIZONTAL	>200	>600	2.51	7.32	5
HORIZONTAL	>300	>600	3.79	7.32	6
VERTICAL	>400	>600	5.13	7.53	7
VERTICAL	>300	>600	3.90	7.53	8
VERTICAL	>200	>600	2.67	7.53	9
VERTICAL	>150	>600	1.78	7.53	10
VERTICAL	>150	>700	1.78	8.61	11
VERTICAL	>100	>650	1.4	7.88	12
VERTICAL	>50	>700	0.54	8.61	13
SWING	>50	>700	0.52	8.55	14
SWING	>100	>650	1.31	7.65	15
SWING	>150	>700	1.76	8.54	16
SWING	>150	>600	1.76	7.42	17
SWING	>200	>600	2.50	7.42	18
SWING	>300	>600	3.82	7.42	19

Each test involved the test cycle being repeated approximately ten times, a delay being included in each of the small programs written, to enable the DTI to be read after each cycle. In this way, the deviation in positional repeatability could be obtained with the robot working a series of different locations within its working area.

The maximum deviation in positional repeatability in all the tests was found to be 11.5 thousandths of an inch (approx 0.3 mm). The average value for deviations in position on the horizontal axes were:- 0.0016", swing 0.0018" and vertical axis 0.0037". The average value for deviation with combined axial movement was found to be 0.0023". On the horizontal axis, the distance of traverse of the arm does not greatly affect the positional repeatability of the robot. This can be seen by looking at the spread of results shown in table 7.2. Positional repeatability is always within 0.004" deviation. On the vertical axis of the robot the spread of positional deviation is increased - most cycles fall within 0.010" deviation, the maximum being 0.0115", and on the swing axis the positional repeatability is within 0.0076" deviation. On both these axes the distance of traverse and the position of traverse of the arm does not greatly affect the positional repeatability of the robot. The position of traverse would probably have a greater effect on positional repeatability when the robot is heavily loaded. With combined axial motion of the robot, the maximum deviation in postional repeatability is 0.008". The combined axial motion did not adversely affect the repeatability of the Versatran Robot.

All measurements are in imperial units as the dial gauge used was calibrated in these units.

The tests described assess repeatability within a single axis of motion, but more realistically the robot will normally operate in more than one axis. So the last two tests assess repeatability after a combined axial movement. In these tests a program for the controller was written to produce an 'L' shaped motion relative to the robot gripper, so that movement in two of the axes takes place. The two programs were as follows:-

TEST No M

		ACTUAL POSITION (mm) FROM DATUM
1	1600	525.0
2	2100	87.5
3	2650	528.0
4	1300	262.5
5	4008 (2 second delay)	
6	900A (10 repeats)	

TEST No N

		ACTUAL POSITION (mm)
1	2100	87.5
2	1600	525.0
3	1300	262.5
4	2650	528.0
5	4008 (2 second delay)	
6	900A (10 repeats)	

The test results obtained are shown in Table 7.2.

For these tests to be statistically analysed many more tests need to be performed within the working volume of the robot. Within the time scale of the project there was insufficient time to perform any more testing of the robot/controller combination. The results obtained so far have shown favourable repeatability

7.4 AN ACCURACY MEASUREMENT TECHNIQUE

In this section there is an explanation of an accuracy measurement technique which could be used to evaluate the performance of the Versatran.

Open-loop-measurement of positional accuracy is critical. It is of the utmost importance that the robot returns to the point in 3-D space that it has been taught, and having arrived at the point the workpiece is in the expected position. In certain applications it may be equally important that the 3-D route to the prescribed point is also accurate and predictable. A good example of this would be with arc welding robots where linear interpolation techniques are used to generate a required contour.

Table 7.2

HORIZONTAL AXIS

TEST No A 2100 - 2300 CYCLE	 DEV'N	TEST No B 2100 -2500 CYCLE	 DEV'N	TEST No C 2100 - 2650 CYCLE	 DEV'N	TEST No D 2200 - 2300 CYCLE	 DEV'N
1	3.75	1	1.5	1	1.25	1	2.75
2	1.75	2	2.25	2	1.25	2	3.75
3	3.75	3	2	3	0.25	3	4
4	0	4	0	4	1.25	4	2.25
5	0	5	1.75	5	0.75	5	0.75
6	3.5	6	2.25	6	0	6	0.75
7	0	7	2.25	7	1.25	7	2.75
8	2	8	1			8	0
9	3.5	9	0.5			9	0.25

VERTICAL AXIS

TEST No E HORZ @2300 1300 - 1600		TEST No F HORZ @2300 1300 - 1400		TEST No G HORZ @2650 1300 - 1600		TEST No H HORZ @2650 1300 - 1400	
CYCLE	DEV'N	CYCLE	DEV'N	CYCLE	DEV'N	CYCLE	DEV'N
1	5	1	0	1	6.75	1	0
2	1	2	1	2	3.5	2	3.5
3	1	3	8	3	3.5	3	0
4	0	4	3	4	6.5	4	2
5	3	5	5	5	0	5	0
6	4	6	3.5	6	11.5	6	3
7	1	7	4	7	1	7	4
8	3.5	8	7	8	1.5	8	6.5
9	8.5	9	6.5	9	1		
10	2.5	10	9	10	8.5		

.... cont'd

SWING AXIS

TEST No J HORZ @ 2300 VERT @ 1300 1300 - 1600		TEST No K HORZ @ 2600 VERT @ 1600 1300 - 1600		TEST No L HORZ @ 2300 VERT @ 1300 1300 - 1400	
CYCLE	DEV'N	CYCLE	DEV'N	CYCLE	DEV'N
1	3.25	1	6.15	1	0
2	3.05	2	5.25	2	0
3	2.15	3	6.05	3	2.10
4	0	4	2.55	4	0.15
5	0	5	0	5	0
6	1.50	6	7.65	6	0.75
7	2.00	7	2.15	7	2.15
8	0	8	1.15	8	1.75
9	0	9	0.75	9	0.25
10	1.00	10	1.25	10	1.65

COMBINED AXLE MOTION

TEST No M		TEST No N	
CYCLE	DEV'N	CYCLE	DEV'N
1	7	1	8
2	0	2	0
3	0.5	3	1
4	3	4	1
5	3	5	2
6	2	6	1
7	4	7	1
8	1	8	2
		9	3
		10	2
		11	3

Measurement of accuracy must be related to the working volume and articulation of the robot if it is to have any meaning at all. Accuracy of different robots suggests that no machine maintains constant accuracy over its working volume. Inevitably there are many reasons for these variations in accuracy. Bearings need to have some play to allow for rotation. Beams bend and twist under different loading conditions and when connected, as in a robot arm, they can display quite remarkable positional variations under the influence of unbalanced loads. Internal control systems often have ADC and DAC conversion devices that use approximation techniques. Here the dropping of one bit of information could be interpreted over the two or three metres of a robot arm to a positional accuracy of many millimetres.

Similarly data compression technique used in robot software storage algorithms can contribute to the bit-dropping paradigm already described.

Many potential sources of error - some may be catered for because they can be anticipated, measured and the information fed back into the robot system.

Others such as bending and twisting are more complicated. It is true that they could be measured but the error correction required would be inordinately expensive and commercially inviable to implement. Nevertheless, robot systems are produced which can maintain high levels of accuracy. However, it is necessary to find a measuring system that can confirm the accuracies claimed by robot manufacturers.

7.4.1 Volumetric Accuracy Mapping (VAM)

To be pedantically correct, one ought to measure the accuracy of the robot approach to the node from six directions[65] as in figure 7.4.4. This is practically dependent upon the position of the node within the working volume, different nodes will have a smaller number of practical articulation approaches which are illustrated in figure 7.4.1, 7.4.2 and 7.4.3 which are the nodes in figure 7.3.

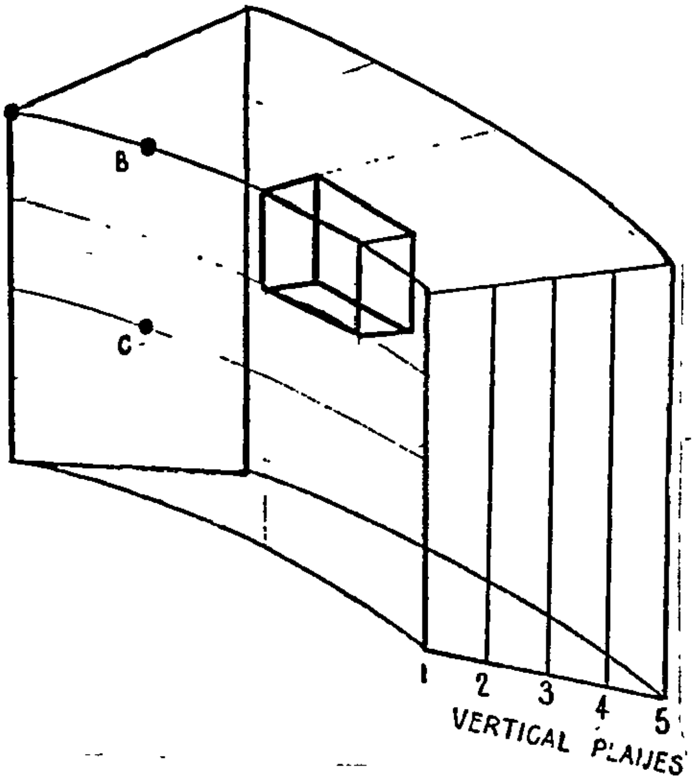


Figure 7.3.1 Polar Coordinate Robot Work Volume

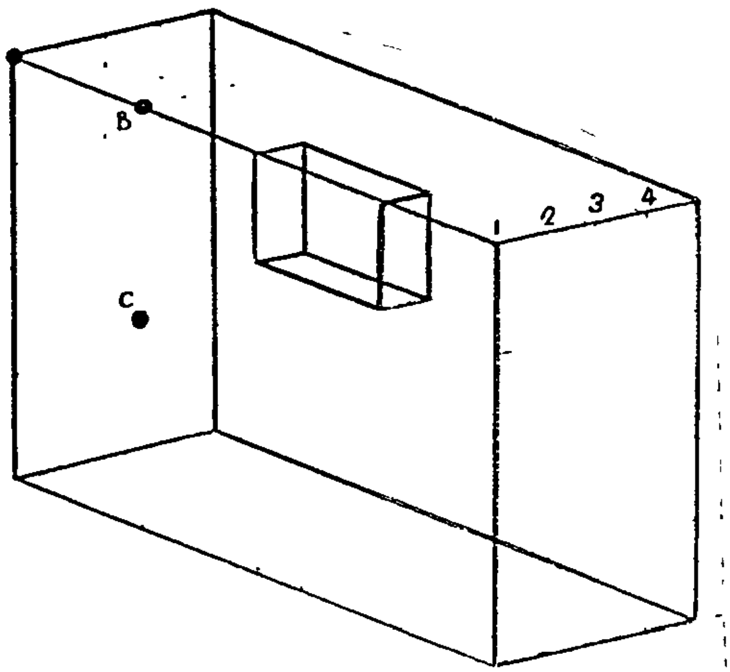
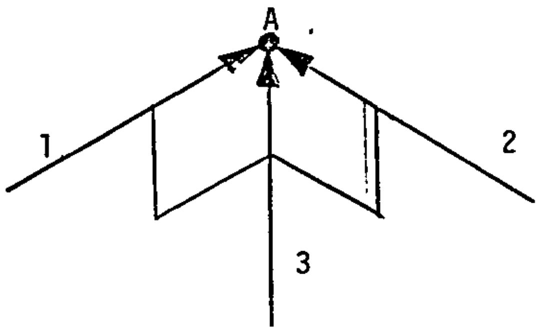
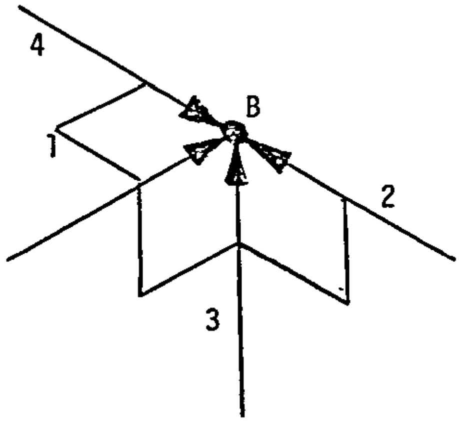


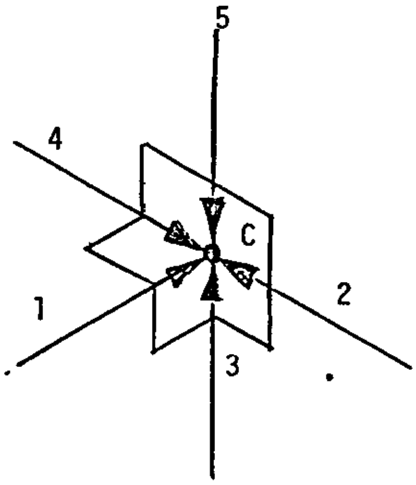
Figure 7.3.2 XYZ, Coordinate Robot Work Volume



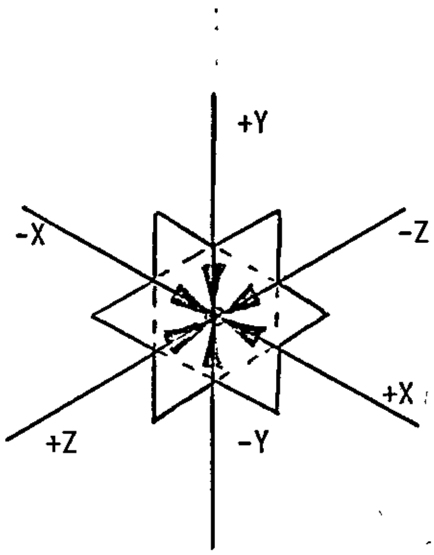
7.4.1. NODE A- THREE POINT NODE



7.4.2. NODE B - FOUR POINT NODE



7.4.3. NODE C - FOUR POINT NODE



7.4.4. MACHINE COORDINATES

FIGURE 7.4. NODAL MAGNITUDES AND MACHINE COORDINATES

However, quite adequate results can be obtained with only three approaches to the node. Typical results are given in table 7.3

Having divided the working area into nodes, the next task is to devise a system for measuring the XYZ accuracies at the individual nodes.

The basic equipment is a pair of vernier calipers, a clock gauge and a method of attaching it to the robot arm. Consider node C.

To begin with the objective will be to check the Z axis accuracy with the clock gauge. The stand (see figure 7.5) is adjusted until point O on the plate is facing the robot and in the vertical plane. The clock gauge is attached to the robot arm at its furthest point (this may be at the end of the gripper assembly for instance) and the maximum working load of the robot may be simulated with lead packing (again at its normal point of action). These two measures will simulate 'worst case' conditions. The plate is placed at the position such that position O corresponds to the position of node C with respect to the robot. Then the robot is taught to approach point O at right angles to the plate. The clock gauge must suffer a deflection that is greater than the quoted accuracy of the robot (eg if the quoted accuracy is $\pm$1.0mm the clock gauge depression should be at least 2.0 mm), a note should be made of this value. The robot should then withdraw from this position. Care must be taken to ensure adequate time is allowed for measurement. Now this is replayed and the value indicated on the clock gauge measured and the position of the finger with respect to point A is measured with the vernier calipers. The clock gauge readings will give a $\pm$ Z figure, and the vernier readings the values of $\pm$ X or $\pm$ Y depending upon the quadrant in which they fall. (eg quadrant LOT gives +X, +Y values). This process should be repeated enough times for the results to be adjusted so that the plate remains in the vertical plane but faces either to the left or right of the robot. Point O must still coincide with node C measurements with respect to the robot. The robot must be taught to bring the clock gauge in at right angles to the plate with constraints similar to the Z axis procedure. The measurements again

Plane 1 Average nodal X Readings (mm)

+0.19	+0.23	+0.15	+0.15	+0.33	+0.4	+0.40
+0.1	+0.21	+0.16	+0.1	+0.17	+0.21	+0.26
+0.21	+0.16	+0.15	+0.05	+0.12	+0.17	+0.20
+0.37	+0.22	+0.15	+0.13	+0.13	+0.10	-0.05
+0.52	+0.41	+0.30	+0.25	+0.1	-0.05	-0.10

Table 7.3

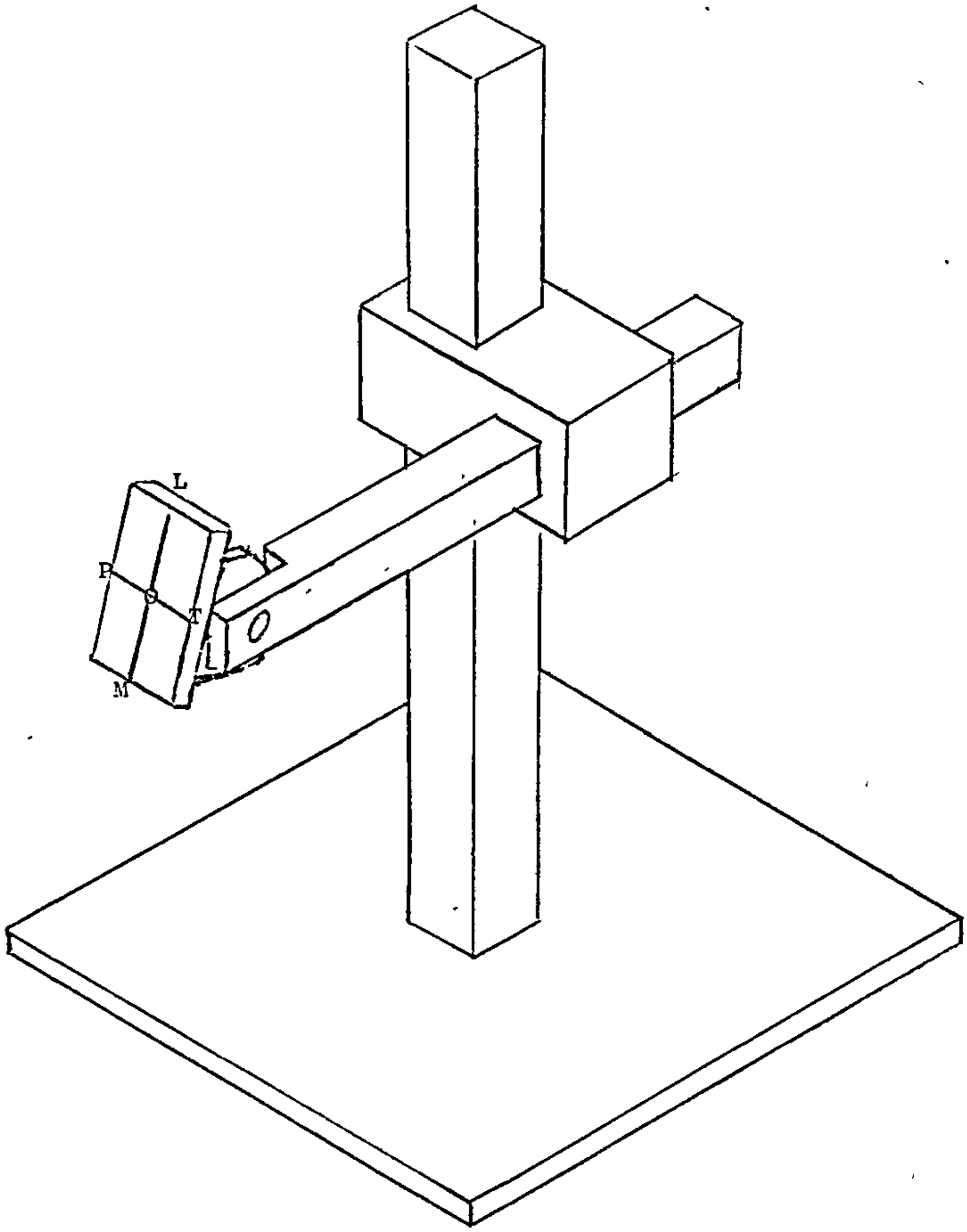


FIGURE 7.5 ADJUSTABLE STAND FOR NODE MEASUREMENT

must be taken except that the measurement on the clock gauge corresponds to the X axis accuracy, the line PT becomes Z axis and the line LM the Y axis. This whole procedure must be repeated with the adjustable plate in the horizontal plane with the clock gauge approaching from above or below depending upon the position of the face of the plate. Finishing one node the complete process is repeated at each node in the working volume.

By averaging all the values of X,Y and Z at each of the nodes, collected by both clock gauge and vernier, account is taken of variations that occur due to different robot articulation. Additionally if the number of repetitions is large enough it will be possible to observe drift in the system.

When the node XYZ accuracies have been obtained the VAM diagram for the robot can be completed. Figure 7.3 shows a set of results for the X readings. Similarly X,Y and Z sets for each of the five planes (see table 7.3), may be constructed.

The overall accuracies may be processed and the results presented in the form of accuracy distribution curves for the robot, which is illustrated in figure 7.6. This technique is long and tedious, however, once it has been repeated enough times with robots of the same design the results may be normalised and a sampling/comparison technique used for production testing. Also much of the measurement may be automated.

Accuracy Distribution Curve

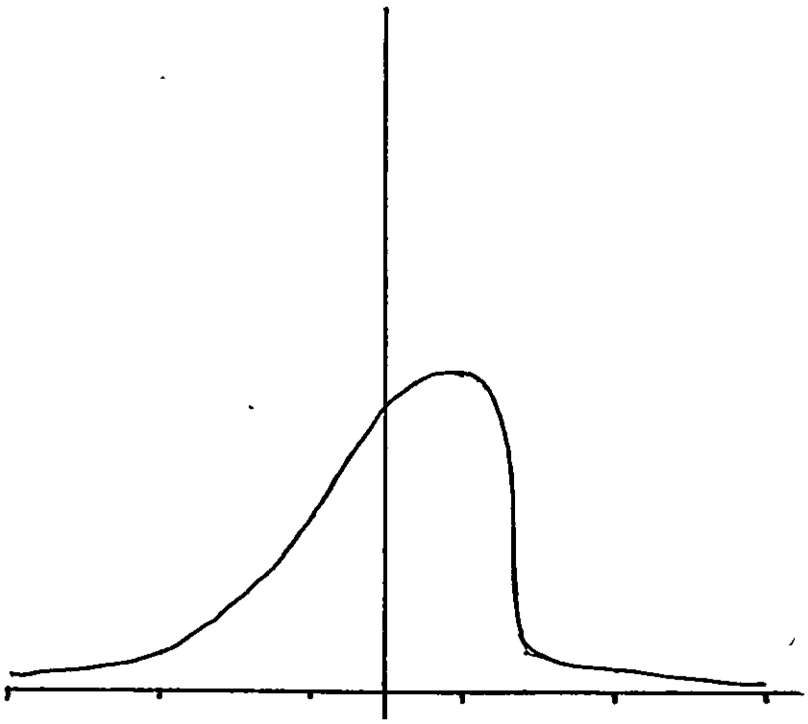


Figure 7.6

CHAPTER 8

DEVELOPMENT OF A SOFTWARE LIBRARY

In this chapter the development of modular microprocessor-based equipment is considered which is designed to serve the need to retrofit early generations of industrial robots of different types and the need to control a wide range of special purpose handling systems[70].

Software algorithms should be developed in modular form to provide a library of software modules from which appropriate modules could be chosen for a particular application. Figure 8.1 shows such a software library and the function of the modules are explained in the following sections. Modules can be classified into two groups, "real-time control" modules and "operator communication" modules.

8.1 REAL TIME CONTROL MODULES

As considered in Section 6.4 each module is chosen using an operation code and modifier addressing method forming an "instruction" which corresponds to a particular operation. The op-codes are listed in figure 8.2. Thus a handling sequence for the robot is determined by the corresponding "instruction sequence" which is a series of instructions stored in read/write (RAM) memory. The modules considered here could be used to form the basic framework of real-time control software for a wide range of robot structures with various servo-drive systems. For each robot or handling structure, "customised" software can be generated by including the appropriate modules.

A Activate an Output Module

The output port is set to one. This will result in the activation of a two state drive eg activate solenoid.

Instruction Format

Op-Code 5 bits	Modifier 11 bits
00001	

Figure 8.1 Software Structure for Versatran Robot

REAL TIME CONTROL MODULES

A System Initialisation and Test

B Activate an Output

C Deactivate an Output

D Test for Input High

E Test for Input Low

F Time Delay

G Jump Unconditional

H Jump Conditional on Input Condition

I Sequence Repeat

J Stop and Re-initialise

K Closed-loop Position Control

L Control Parameter Handling

OPERATOR COMMUNICATIONS MODULES

A	Terminal Driver	(TD)
B	User Instruction Prompts and Sequence Encoder	(IP)
C	User Instruction/Robot Sequence Cross Reference	(CR)
D	Communications Parameter Handling	(CPH)
E	Sequence Edit	(SE)
F	Teach Prompt and Sequence Encoder	(TP)
G	Instruction Sequence Selection and Network Protocol	(NP)

Figure 8.2

INSTRUCTION	OP CODE	MODIFIER	COMMENT
Activate an Output Module	1	0-2047(>7FF)	Activate a Two State Drive
Deactivate an Output Module		0-2047	Deactivate a Two State Drive
Wait for Input High Module	3	0-2047	See if switch opened or closed
Wait for Input Low Module	4	0-2047	See if switch opened or closed
Time delay	5	0-2047	
Jump Unconditional Module	6	0-2047	No of instructions not to be executed
Jump Conditional on Input Condition Module	7	0-2047	Increment for new PC
Sequence Repeat Module	8	0-2047	No of times sequence repeated
Stop and Re-initialise Module	9		No modifier
Closed-Loop Position Control Module	A	0-2047	Up to 6 axes can be controlled
	B	0-2047	
	C	0-2047	
	D	0-2047	
	E	0-2047	
	F	0-2047	

The modifier gives the two state drive which is to be activated. and can lie between 0 and 2047.

B Deactivate an Output Module

Here an output port is reset low thereby deactivating a two state drive. The op-code is 2.

C Wait for Input High Module

This module tests the state of a CRU input port and waits until that port becomes high ie it waits until a switch closes or opens. The modifier contains the value of the CRU port which is to be high.

D Wait for Input Low Module

Here an input port is tested and the robot forced to wait until that port is low.

E Time Delay Module

This module allows a programmed delay so that operations to be carried out, for example, pick up or put down a part when there is no feedback on the drives. The modifier contains the value of the delay.

F Jump Unconditional Module

This is used when part of the programmed movement is not to be executed. The modifier contains the number of instructions which are not to be performed.

G Jump Conditional on Input Condition Module

This module is used to modify the robot sequence which is dependent on some external event, for example, the arrival of a part could be sensed and could cause a current operation to cease and the robot to pick up the part and perform some operation on it.

H Sequence Repeat Module

This module allows any sequence to be repeated without re-entering the data.

I Stop and Re-initialise Module

This module will stop the robot in a safe condition, ie will not drop a part if there is one in its jaws.

J Closed-Loop Position Control Module

This module will service a number (n) of point-to-point position control servomechanisms of the type described in section 6.4.1. The feedback loop is continuously closed by using an interrupt service subroutine which is entered once per sampling interval. In this way, even if axis movements are not programmed, the drift on each axis can be overcome. The modifier contains the digital value of the required position.

The above modules are independant of robot type and have been designed in this way to allow the same modules to be used irrespective of axis configuration etc. However, it is necessary to "customise" some of these modules to suit a particular control task, to allow system initialisation and testing of a particular robot type to be achieved. To achieve this customising and system initialisation two other modules are required. It should be stressed that it is only these two modules which are robot dependant.

K Control Parameter Handling Module

This module will store data which is relevant to a particular robot, for example, to achieve closed-loop position control information such as the sampling interval, the compensated velocity command, the limits of the velocity, number of axes, types of axes, limits of position for the axes, is stored in a data table so that is available to the other real-time control modules.

L System Initialisation and Test Module

This will, for example switch on the hydraulics and allow it to reach the required operating temperature and pressure. It will keep testing these conditions and until they are within acceptable limits not allow the robot to move.

When the required 'instruction sequence' has been loaded into RAM, it first has to be de-coded. An instruction pointer points to the first operation, the op-code is separated from the modifier which are stored in separate tables. The instruction pointer is automatically incremented to point to the next instruction which is de-coded, this process is repeated until all the instructions have been de-coded. The instruction sequence is then executed.

8.2 TASK PROGRAMMING OPERATOR COMMUNICATIONS MODULES

The operator communications modules were developed to aid an operator in producing "instruction sequences" to allow robot tasks to be programmed without the need for a skilled operator. To achieve this, prompts are given to the operator concerning the programming options available and the operator responses result in "instruction sequences" being stored in RAM. This operator communications software was also developed in modular form to attempt to provide a base software development. The reader is referred to figure 8.1 which gives a list of the modules.

The communications parameter handling (CPH) module customises the software to a particular robot. This module was used to access and transfer parameters from a number of text and data files. The various axis parameters include axis indentification, axis I/O addressing, axis limits and permissable velocities, position and velocity loop gains, ADC channel addresses and sampling periods, gripper identification, gripper actuation sequences, interlock sequences and indicators, and string text concerning operator prompts and error messages. The "user instruction prompts and sequence encoder" module asks the relevant questions so that the parameters can be loaded into the CPH and also the "instruction sequence" of operations can be obtained and stored in the robot sequence cross reference module. The "sequence edit" module will allow the "instruction sequence to be altered when required.
The "teach prompt and sequence encoder" module (TP) will depend on the mechanical structure of the robot. For the Versatran only limited teach facilities can be incorporated as the only method of moving the robot is using its controller so a teach pendant may be used to move it in small predetermined steps. Other robots can

"walked" through a sequence and so are easier to teach.

The "instruction sequence selection and network protocol" (NP) will allow the control of more than one robot. This will enable the robots to "work together" to perform operations.

CHAPTER 9

FURTHER DEVELOPMENT OF AN OPERATOR COMMUNICATIONS MODULE

The previous chapter gives an outline of the various modules that are required for flexible software programming. This chapter describes an operator communications module which describes the axes eg are they linear or rotary, what is the maximum permissable velocity on each axis etc and then input a sequence of operations. An edit facility is available so that this sequence can be altered if required.

The program is written in Pascal as it is a highly structured language and it requires only a small amount of documentation.

9.1 PROGRAM (DATA_INPUT)

A flow chart showing the outline of this program is shown in figure 9.1 and a program listing is shown in Appendix 7.

The constants are declared first and MAX_ NO refers to the number of axes and MAX_ POSITIONS to the possible number of positions. The types are declared and REC 1 is a record which contains various elements which are

NAME_ AXIS	The name of the axis eg vertical
AXIS TYPE	Is the axis linear or rotary?
MAX_TRAVEL	Limit of digital number for position ie > 7FF for the Versatran
MAX_VEL	The maximum +ve velocity allowed
FEEDBACK	Is the feedback analogue or digital
POSIT	Is the axis point-to-point ie 2 positions or are there many positions

REC 2 is a record which will contain the positions for one axis in an array called STORE. TOT is an array which contains elements of type REC 1 and TOT 2 is an array which contains elements of type REC 2.

The variables are then declared. POINT is an array which will contain the next instruction number. TOTAL AND TOTAL 2 are variables of type TOT and TOT 2 respectively. The integers are then declared

Figure 9.1 Flow Chart for DATA_ INPUT program

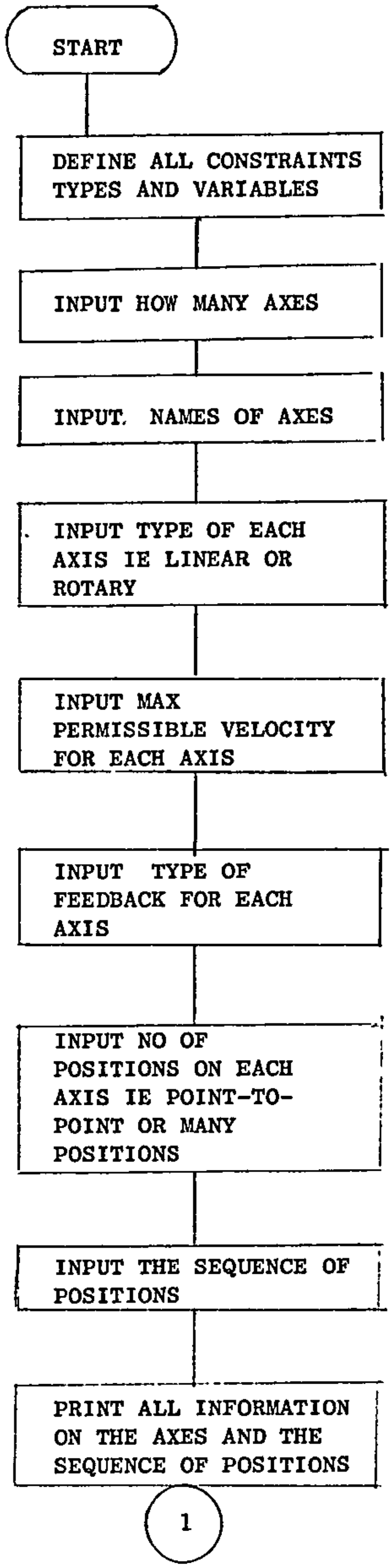
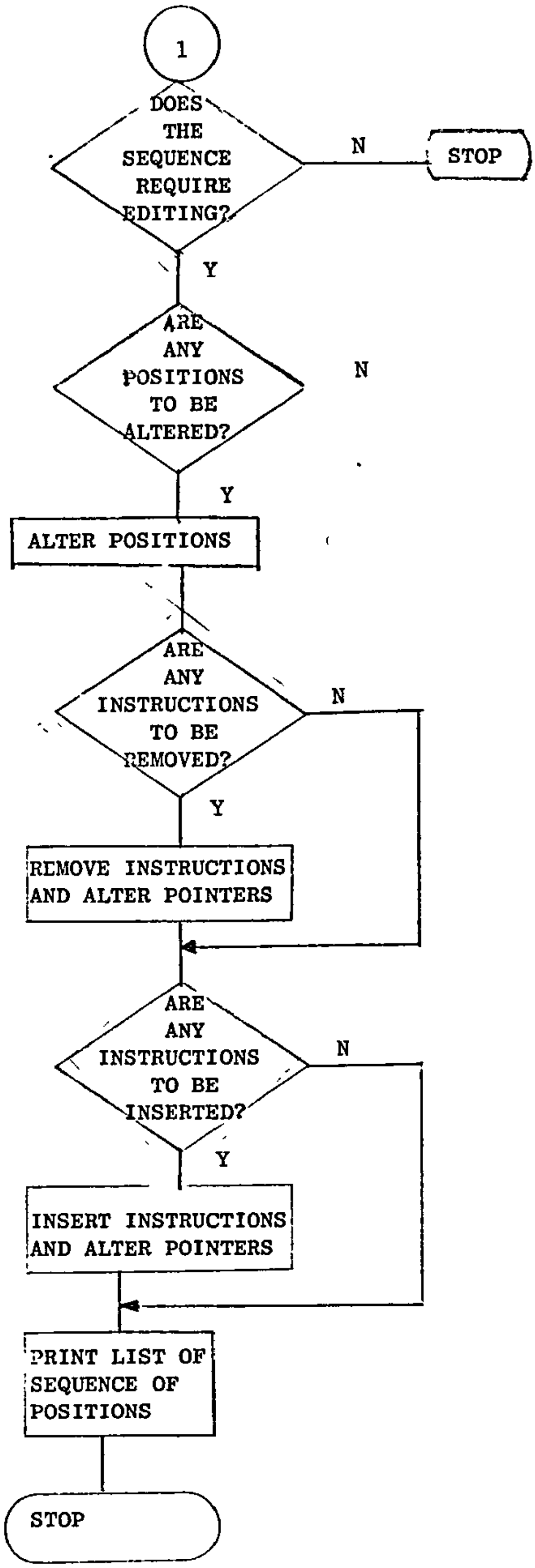


Figure 9.1 cont'd



which are :-

NO_ OF_ EDITS	The number of instructions to be altered
NO_ OF_ INSERTS	The number of lines to be inserted
NO_ OF_ REMOVES	The number of lines to be removed
NO_ OF_ AXES	The number of axes on the robot
NO_ OF_ POS	The number of positions
NUM	This permits the value to be stored in the correct place in the STORE and POINT arrays
COUNT	This permits numbers to be stored in the correct place in an array
LINS	This is equal to the new number of instructions after inserts
INSTS	This increases the number of positions when inserting instructions
NOS	This permits numbers to be stored in the correct place in an array
LNOS	This is the LAST NOS and is used to find out when something is finished eg all the instruction numbers of the alterations are in the array
INST	This allows the pointer to be altered when removing the next instruction

The variables are type CHAR are:-

REMOVE)
ALT)
INSERT) These all contain Y or N depending on the editing that is required
EDIT)

There are then three arrays, ALTER, REM, and INS which respectively contain the instruction numbers of the instructions which require altering, the instruction numbers of the instructions which are to be removed and the instruction numbers of the instructions before an instruction is to be inserted, that is instruction no

3
4
insert new instruction

the value in the array will be 4 to insert this new instruction.

The actual program then starts at line 1 when the details of the axes are recorded.

The writeln statement will write to the terminal what is in the inverted commas, this value is then read into the variable NO_ OF_ AXES. Line 6 starts a loop using COUNT from 0 to NO_ OF_ AXES, in this loop all the names of the axes are inserted into the array TOTAL in the part of the record NAME. AXIS, COUNT decides where the values are placed. Each name is on a separate line due to the READLN being used.

When COUNT equals NO_ OF_ AXES statement 10 is executed, which is END; COUNT is then reset to zero and more details about each axis are recorded which continues up to line 35.

BEGIN (*INPUT POSITIONS*) on line 35 starts the insertion of the positions into the correct array. The number of positions is first selected and read into NO_ OF_ POS, this is then used to terminate the loop when NUM is equal to it. Another loop commences on line 39 concerned with the number of axes, line 41 is a writeln statement and the following is an example of what may be written on the VDU.

POSITION FOR VERTICAL

The readln statement then reads the value which is input into the correct part of TOTAL 2 eg the first time around COUNT and NUM will be equal to one as so the value will be stored in the first place in an array of TOTAL 2 in the first place in an array STORE. This inner loop continues until the first position for all the axes has been read in. The pointer to the next instruction is then read into the array POINT. This sequence is repeated until all the positions have been read in.

A list of what has been entered is then printed out which is shown in the debug routine in figure 9.2. this finishes on line 76. From line 76 to line 149 is an edit which can make changes to the positions which have been entered into the record TOTAL 2.

Figure 9.2.Typical Print-Out from the Program DATA_INPUT

```
VERTICAL

    THE TYPE OF AXIS IS        LINEAR
    THE MAX VALUE FOR POSITION IS  7FF
    THE MAX VELOCITY IS        1.8V
    THE METHOD OF FEEDBACK IS DIGITAL
    NO OF POSITIONS ON THE AXIS  MANY


                 VERTICAL           POINTER

    1               100                2
    2               250                3
    3               150                0
```

The first question asks if the sequence requires editing and either Y for YES or N for NO is read into EDIT, a case system is then used so the correct portion of the program is executed. If N is input the statement Your program is correct will appear on the VDU and the same list as previously will appear on the VDU, if Y is input the next question asks if any alterations are to be made if yes the number of alterations is read into NO_ OF_ EDITS, which is then used to control how many times the loop to input the instruction numbers into the array ALTER is executed. These instructions are then altered, by inserting the positions for all axes for each instruction.

If there are no alterations the case 'N' is executed and the statement No alterations to values required will be written out on the VDU. The next case statement concerns with removing of instructions if the case is 'N' then the statement NO LINES TO BE REMOVED appears on the VDU. If the case is 'Y' then the number of instructions, and instruction numbers to be removed are entered. The pointers of the previous instructions before the ones which are to be removed are then changed

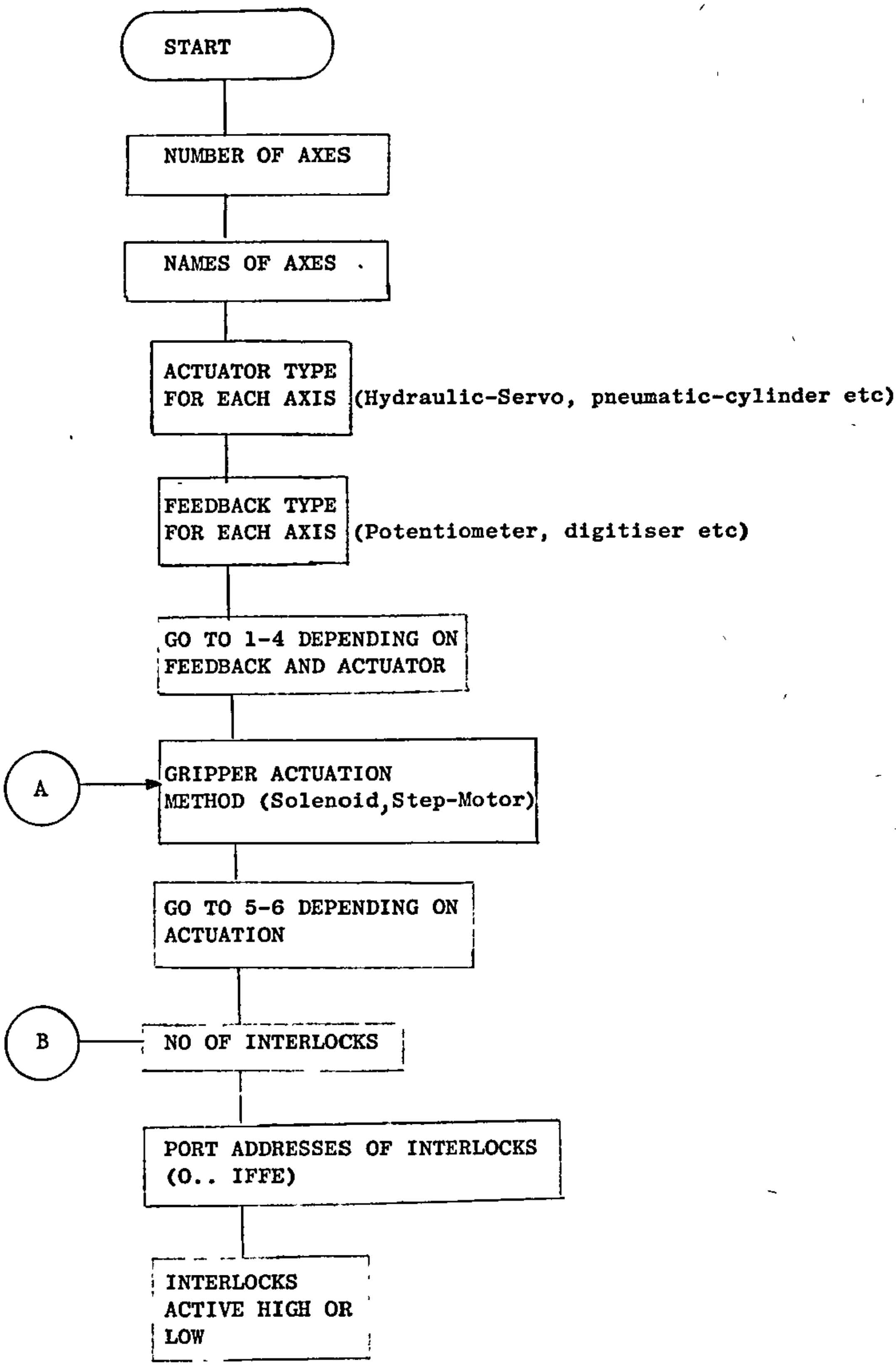
INSTRUCTION NO			POINTER
	1	VALUES OF POSITIONS	2
	2		3
remove	3		

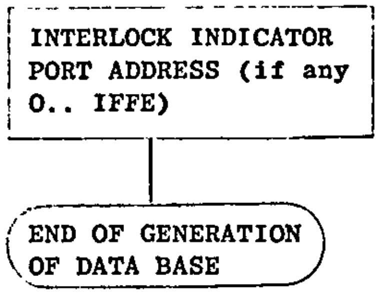
the pointer of instruction 2 will be changed to 4 (lines 115-120).

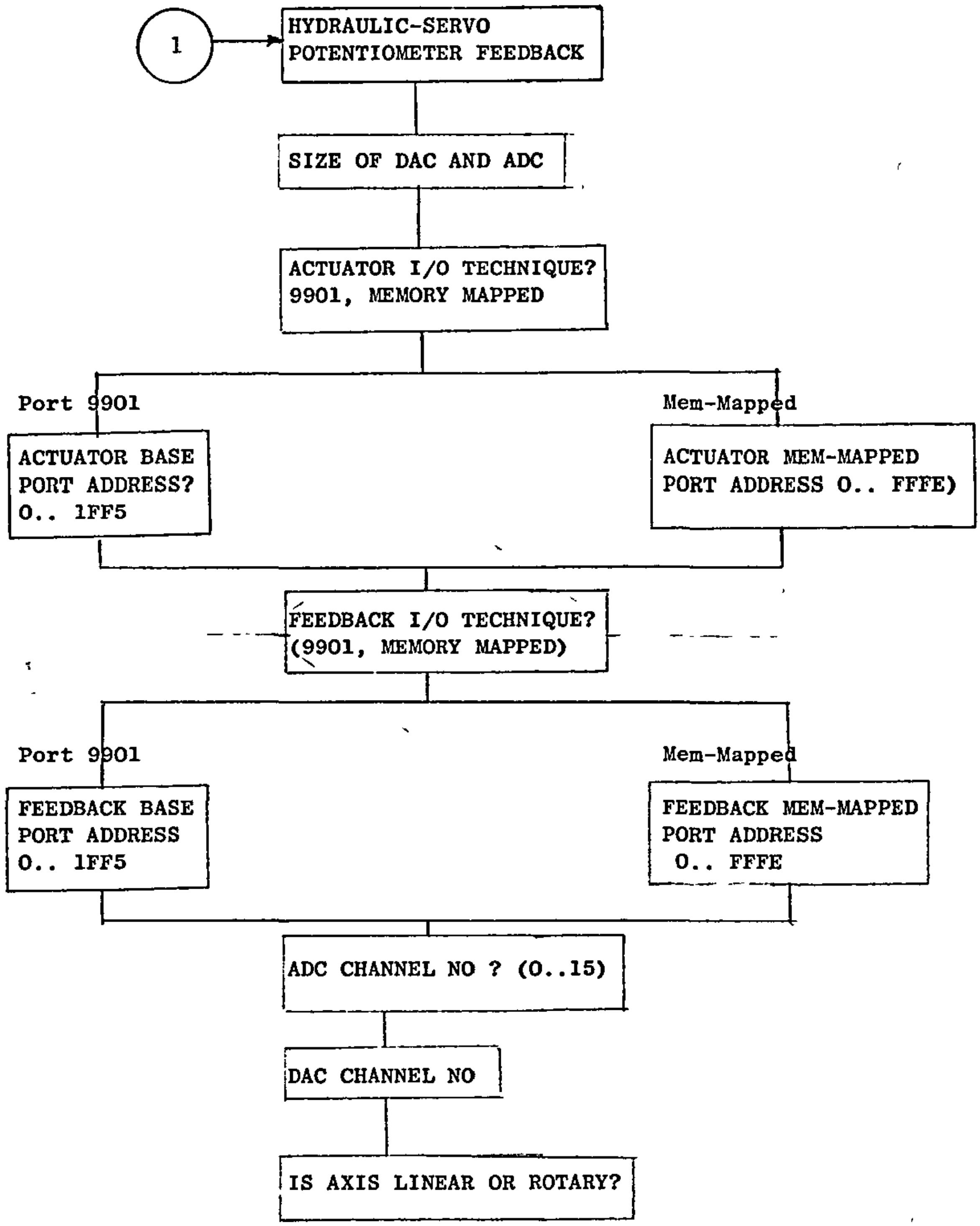
The final case statement is to insert any instructions. If the 'Y' case is to be executed, the number of instructions to be inserted and the instruction numbers which are to have new instructions following them are entered. The pointer of one of the instructions which is to be followed by new instructions is altered then the new instruction is entered, this sequence is repeated until all the instructions have been entered. Then the list of all the positions is re-printed.

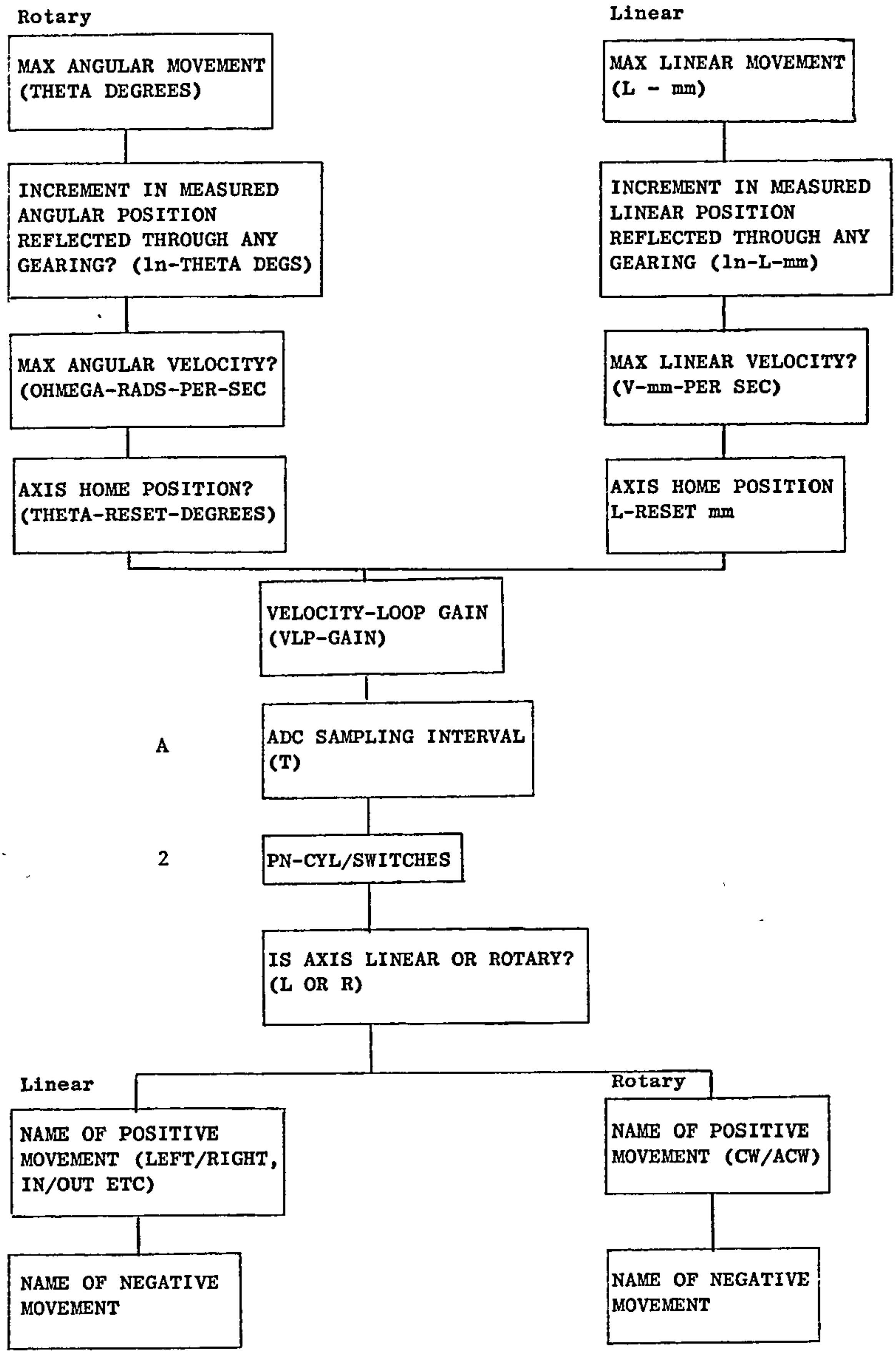
This program is by no means entirely finished figure 9.3 shows a flow chart of a flexible operator communications module which can be utilised for a wider range of robots.

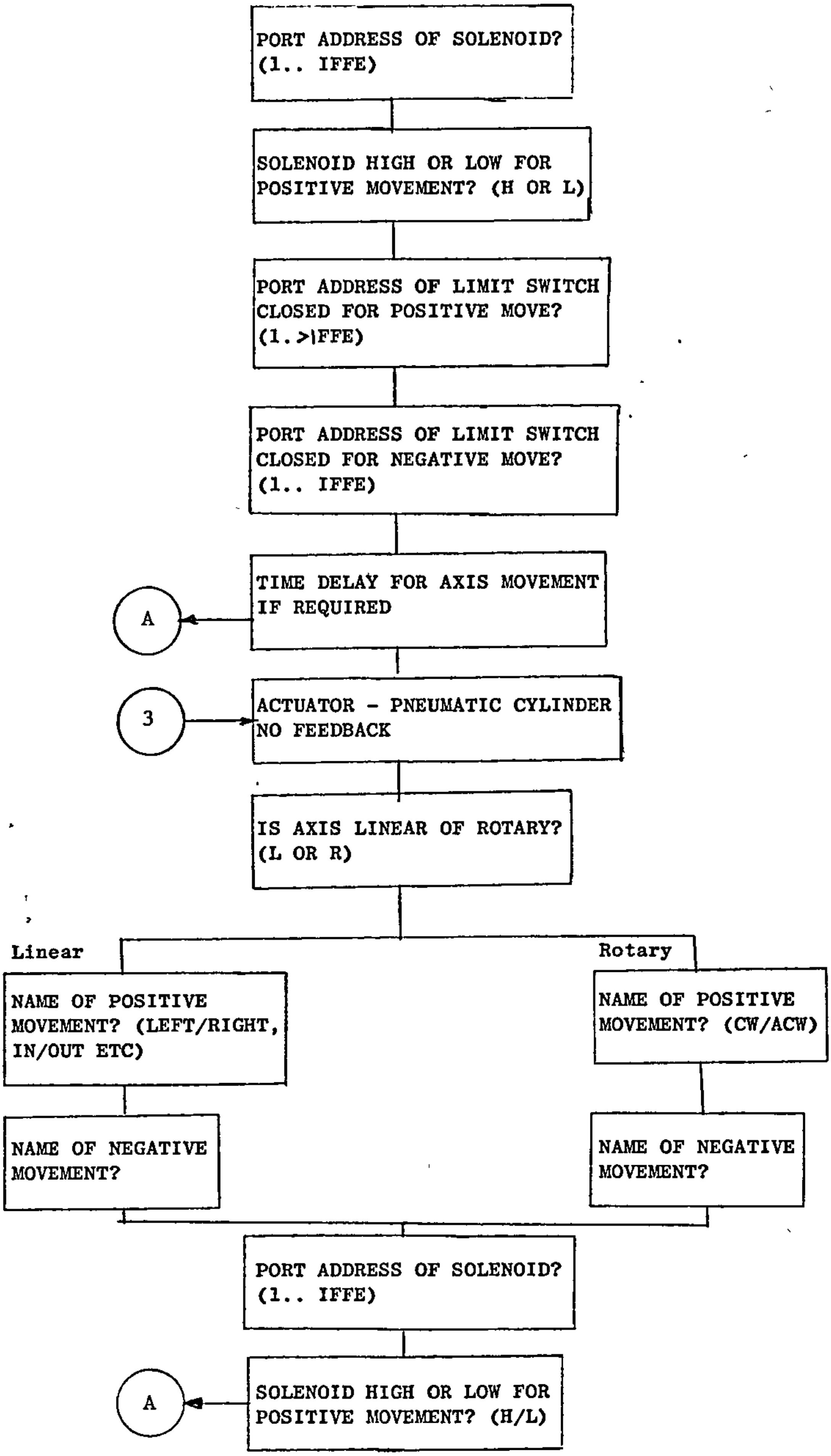
Figure 9.3 A Flexible Operator Communications Module

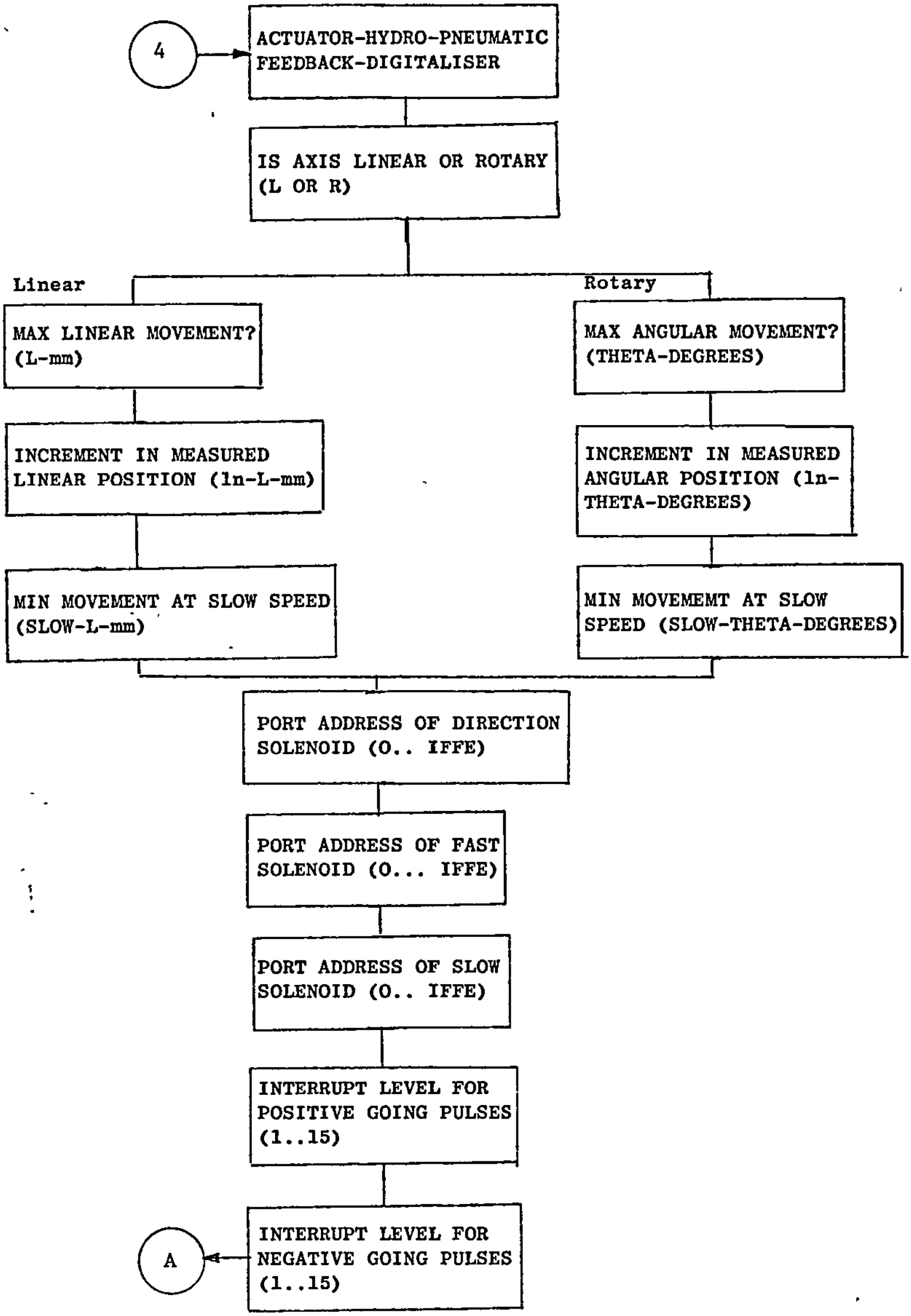


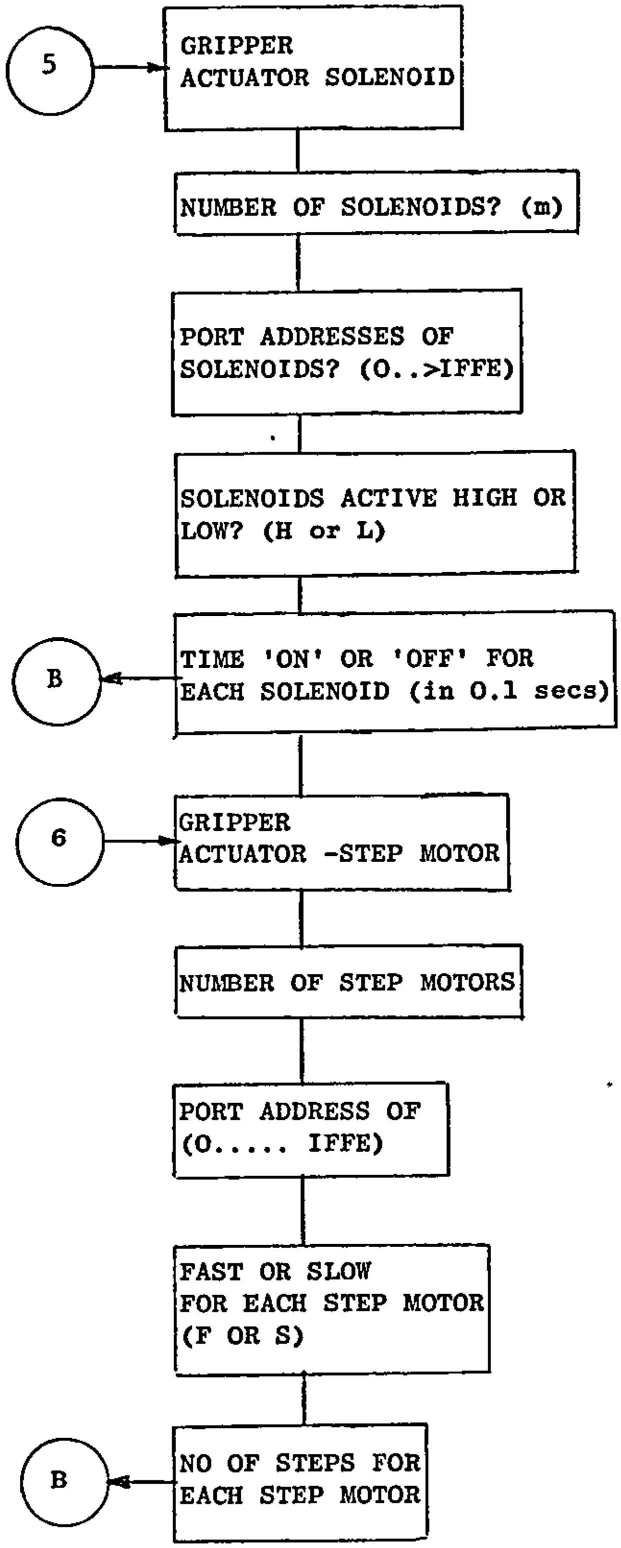

INTERLOCK INDICATOR PORT ADDRESS (if any O.. IFFE)
END OF GENERATION OF DATA BASE


1
HYDRAULIC-SERVO POTENTIOMETER FEEDBACK
SIZE OF DAC AND ADC
ACTUATOR I/O TECHNIQUE? 9901, MEMORY MAPPED
Port 9901
Mem-Mapped
ACTUATOR BASE PORT ADDRESS? O.. 1FF5
ACTUATOR MEM-MAPPED PORT ADDRESS O.. FFFE)
FEEDBACK I/O TECHNIQUE? (9901, MEMORY MAPPED)
Port 9901
Mem-Mapped
FEEDBACK BASE PORT ADDRESS O.. 1FF5
FEEDBACK MEM-MAPPED PORT ADDRESS O.. FFFE
ADC CHANNEL NO ? (O..15)
DAC CHANNEL NO
IS AXIS LINEAR OR ROTARY?


Rotary
MAX ANGULAR MOVEMENT (THETA DEGREES)
INCREMENT IN MEASURED ANGULAR POSITION REFLECTED THROUGH ANY GEARING? (1n-THETA DEGS)
MAX ANGULAR VELOCITY? (OHMEGA-RADS-PER-SEC
AXIS HOME POSITION? (THETA-RESET-DEGREES)
Linear
MAX LINEAR MOVEMENT (L - mm)
INCREMENT IN MEASURED LINEAR POSITION REFLECTED THROUGH ANY GEARING (1n-L-mm)
MAX LINEAR VELOCITY? (V-mm-PER SEC)
AXIS HOME POSITION L-RESET mm
VELOCITY-LOOP GAIN (VLP-GAIN)
A
ADC SAMPLING INTERVAL (T)
2
PN-CYL/SWITCHES
IS AXIS LINEAR OR ROTARY? (L OR R)
Linear
NAME OF POSITIVE MOVEMENT (LEFT/RIGHT, IN/OUT ETC)
NAME OF NEGATIVE MOVEMENT
Rotary
NAME OF POSITIVE MOVEMENT (CW/ACW)
NAME OF NEGATIVE MOVEMENT


PORT ADDRESS OF SOLENOID? (1.. IFFE)
SOLENOID HIGH OR LOW FOR POSITIVE MOVEMENT? (H OR L)
PORT ADDRESS OF LIMIT SWITCH CLOSED FOR POSITIVE MOVE? (1.>IFFE)
PORT ADDRESS OF LIMIT SWITCH CLOSED FOR NEGATIVE MOVE? (1.. IFFE)
TIME DELAY FOR AXIS MOVEMENT IF REQUIRED
A
3
ACTUATOR - PNEUMATIC CYLINDER NO FEEDBACK
IS AXIS LINEAR OF ROTARY? (L OR R)
Linear
Rotary
NAME OF POSITIVE MOVEMENT? (LEFT/RIGHT, IN/OUT ETC)
NAME OF POSITIVE MOVEMENT? (CW/ACW)
NAME OF NEGATIVE MOVEMENT?
NAME OF NEGATIVE MOVEMENT?
PORT ADDRESS OF SOLENOID? (1.. IFFE)
SOLENOID HIGH OR LOW FOR POSITIVE MOVEMENT? (H/L)
A


4
ACTUATOR-HYDRO-PNEUMATIC FEEDBACK-DIGITALISER
IS AXIS LINEAR OR ROTARY (L OR R)
Linear
Rotary
MAX LINEAR MOVEMENT? (L-mm)
MAX ANGULAR MOVEMENT? (THETA-DEGREES)
INCREMENT IN MEASURED LINEAR POSITION (1n-L-mm)
INCREMENT IN MEASURED ANGULAR POSITION (1n-THETA-DEGREES)
MIN MOVEMENT AT SLOW SPEED (SLOW-L-mm)
MIN MOVEMEMT AT SLOW SPEED (SLOW-THETA-DEGREES)
PORT ADDRESS OF DIRECTION SOLENOID (O.. IFFE)
PORT ADDRESS OF FAST SOLENOID (O... IFFE)
PORT ADDRESS OF SLOW SOLENOID (O.. IFFE)
INTERRUPT LEVEL FOR POSITIVE GOING PULSES (1..15)
INTERRUPT LEVEL FOR NEGATIVE GOING PULSES (1..15)
A


5
GRIPPER
ACTUATOR SOLENOID
NUMBER OF SOLENOIDS? (m)
PORT ADDRESSES OF
SOLENOIDS? (O..>IFFE)
SOLENOIDS ACTIVE HIGH OR
LOW? (H or L)
B
TIME 'ON' OR 'OFF' FOR
EACH SOLENOID (in 0.1 secs)
6
GRIPPER
ACTUATOR -STEP MOTOR
NUMBER OF STEP MOTORS
PORT ADDRESS OF
(O..... IFFE)
FAST OR SLOW
FOR EACH STEP MOTOR
(F OR S)
B
NO OF STEPS FOR
EACH STEP MOTOR

CHAPTER 10

PROJECT CONCLUSIONS AND FURTHER WORK

CONCLUSIONS

The objectives of this project were to develop a microprocessor based controller for robotic devices which could demonstrate considerable flexibility with regard to operator facilities when sequence programming and which could be structured to allow the future inclusion of various enhanced features as demonstrated by "state of the art" industrial robots. A Versatran Industrial Robot was used as a "test bed" to evaluate the features of the controller developed. From a literature survey undertaken it was evident that there is a need for the development of a controller which could be used to control a wide range of robotic forms both for retrofitting to conventional pedestal industrial robots which presently are served by outdated control systems and for the control of other forms of handling structure not necessarily demonstrating conventional co-ordinate orientation.

At present the Texas Instruments TMS 9900 family of microprocessors is favoured within the Department as comprehensive support for software development coupled with hardware and software "debugging" aids is available. The hardware for the controller comprised:- a Texas Instruments single board computer; standard analogue interface printed circuit boards; specially designed interface drivers for the axis servo-valves,gripper solenoids and hydraulic supply interlocks. Power supplies completed the hardware structure which was all held in a racking system. The completed hardware was constructed and tested as part of the project, however, the majority of the project concered the implementation of software to control the robot. Initially this related to the positioning of a single axis to a pre-programmed position and developed through the control of one axis to many pre-programmed positions to the control of all the major axes in point-to-point mode with additional features such as open/close jaws being incorporated. The final version of the real time software provided flexibility by using "OP codes" to specify robot sequences in a "textural manner".

The real time control strategy for each axis was developed around a loop closure within the microprocessor controller. Each axis position was sampled every 38ms within an interrupt service routine which was controlled by an interval timer. Control is obtained by evaluating a velocity command for each axis every sampling interval and updating the output voltage to each servo-valve. This approach was adopted to allow maximum flexibility in future control algorithms. As the loops were closed internally digital compensation algorithms can be introduced to improve the response of each of the axes for both point-to-point and contouring applications. Another approach is to close each loop external to the microprocessor controller and this method would make the interfacing simplier. However, in providing future enhancements to a system utilising external loop closure problems could be experienced due to an inherent inability to modify system response particularly if contouring capability is required. All the real time control modules were written in Assembly Language as the only available Texas Instruments implementation of Pascal, through the project duration, was an interpretive (p-code) version which imposed a significant time overhead and made its use impractical for axis control. Subsequently, Texas Instruments have released a native code Pascal compiler which could now be used to derive equivalent real time control software. However, a memory overhead of 15-20K is required to provide a Pascal environment for native code derived software, although such a memory overhead is becoming less significant with fast reducing cost of memory devices. The operator communications modules were written both in Pascal and Assembly languages. However, the Pascal programs were debugged and run on a microprocessor development system but not run on the target system due to insufficient memory. Using a high level language such as Pascal improves significantly the transportability and inherent documentation of the software and only slight modifications would be required to run the programs on another computer. The operator communications software developed to aid sequence programming provides considerable flexibility but there are inherent disadvantages in some applications as the spacial co-ordinates of each axis position must be known and programmed for any handling sequence. These positions are input via a VDU and even if a teach program was implemented using a VDU as a terminal it would be difficult to achieve the required position as the VDU is remote from the robot. To overcome these difficulties a teach pendant is being designed in subsequent work which has followed the developments described here and this will offer the opportunity of utilising the advantages of both teach and textural programming facilities. However,it was not possible to provide teach facilities within the project duration.

After the control software had been developed and fully debugged a limited amount of testing was undertaken which included the monitoring of feedback and repeatability. Feedback was monitored to investigate dynamic response and a measure of repeatability evaluated by measuring errors for a series of moves by using a dial guage. For the results of these tests to be statistically complete many test need to be performed at various positions within the working volume of the robot. However, for the Versatran robot/ controller combination favourable repeatability test results have been achieved within the duration of the project. It is necessary that any measurement of accuracy must be related to the working volume and articulation of the robot if it is to have any meaning at all. No machine maintains constant accuracy over its working volume due to various reasons which include:- bearings need to have some play to allow for rotation; beams bend and twist under different loading conditions and when connected can display positional variations under the influence of unbalanced loads. Volumetric accuracy mapping is a technique which could be utilised to test the accuracy throughout the entire working volume, this method would be enhanced if it could be automatically performed as it is a long and tedious method of assessing repeatability.

The performance of the robot/controller combination could also be assessed for various manufacturing applications such as spot welding, loading of presses, component feeding and inspection of machine tools and if continous path algorithms were incorporated within the controller structure, arc welding and paint spraying .

RECOMMENDATIONS FOR FURTHER WORK

An extremely wide range of enhancements could be incorporated within the overall hardware and software adopted for the controller. Furthermore the performance of existing and enhanced controls should be studied, particularly with regard to manufacturing applications. Possible enhancements and studies are listed below.

i) Implement software as described in Chapter 9 so that an extensive library of modules could be made available.

ii) Implement software algorithms for continuous path movement to enable the robot to be used for operations such as painting and arc welding.

iii) Develop a hand held teach pendant as an alternative sequence programming method.

iv) Design and construct additional interface circuitry, in modular form, so that a library of software modules can be utilised with other robotic systems.

v) The performance of the robot should be evaluated using Volumetric Accuracy Mapping as described in Chapter 7.

vi) Perform a number of application studies to evaluate the facilities incorporated within the control system.

vii) Consider the use of various sensing devices in relation to such application studies.

viii) Implement network software to allow the controller via a node to access a commercially available "open" local area network to allow the integration of the robot functions with those of the manufacturing environment in which it is to be used.

REFERENCES

1 M W THRING
Theory of Robots
3rd Symposium on Theory of Robots and Manipulators 1978 pp 581-586

2 J F ENGELBERGER
Robots Make Economic and Social Sense
Atlanta Economic Review, pp 4-8, July, Aug 1977

3 J F ENGELBERGER
Robots Thrive in Hot, Hazardous and Boring Jobs
Presented at Westinghouse Machine Tool Forum, 1976

4 M W THRING
Robotics and Telechirics
Industrial and Commerical Training, June 1980, pp 60-63

5 W B HEGINBOTHAM
Reasons for Robots
1st Conference on Industrial Robot Technology, 1973 pp R1/1 - R1/12

6 J McCOOL
Microprocessors in Control of Robots
Electronics and Power Nov/Dec 1979 pp. 796-799

7 J F ENGELBERGER
Robots in Practice
Kogan

8 R H MASKREY
Microprocessor role in closed loop systems
Design Engineering December 1978 pp 27-34

9 M CORWIN
Benefits of Computer-Controlled Robots
9th Int. Symp on Ind. Robots, 1979, pp 453-461

10 J A GUPTON
Microcomputers for External Control Devices
Dilithium, U.S.

11 H McCALLION et al
A Compliant Device for Inserting a Peg in a Hole
The Industrial Robot, June 1979, pp 81-87

12 H VAN BRUSSEL AND J SIMONS
A Self-Learning Robot for Automatic Assembly
1st Conference on Automatic Assembly 1980, pp 295 - 308

13 A R JOHNSTON
Proximity Sensor Technology for Manipulator End Effectors
Mechanism and Machine Theory, Vol 12, 1977, pp 95-109

14 E D TANNER
Basics of Robotics
SME TECH 2nd Conf on Robots, Detroit, Mich. Oct 31 1977, pp MS 77-734

15 A COOKE
Running Robots
Systems International, November 1980, pp 27-28

16 The new promise of the power of vision
Computing, June 12, 1980, pp 6-10

17 A N ANNUSHVILI et al
Simple real-time visual system for industrial robots
7th International Symposium on Industrial Robots 19-21 Oct 1977, Tokyo, Japan, pp 507-514

18 G BELFORTE et al
Some New Systems for Recognition and Positioning of Mechanical Parts
10th International Symposium on Industrial Robots March 5-7, 1980, Milan, pp 203-213

19 R BOLLES & R PAUL
The Use of Sensory Feedback in a Programmable Assembly System
Stanford Artificial Intelligence Lab, Stanford University, Stanford, Calif, Memo AIM 220 Oct 1973

20 S H DRAKE et al
High Speed Robot Assembly of Precision Parts Using Compliance Instead of Sensory Feedback
7th International Symposium on Industrial Robots, 19-21 Oct, 1977 Tokyo, Japan, pp. 87-93

21 W B HEGINBOTHAM et al
A Practical Visual Interactive Robot Handling System
The Industrial Robot, June 1975, pp 61-66

22 L C DRISCOLL
Steps Toward Blue-Collar Robot Vision
3rd International Symposium on Industrial Robots, 1973

23 G J AGIN
An Experimental Vision System for Industrial Application
9th International Symposium on Industrial Robots, 1979

24 A N ANUASHVILI AND V D ZOTOV
A Simple Real-Time Visual System for an Industrial Robot
7th International Symposium on Industrial Robots, 19-21 Oct, 1977, Tokyo, Japan, p 507-514

25 Y SHIRAI
On Application of 3-Dimensional Computer Vision
Bulletin of the Electrotechnical Laboratory Vol 43, No 6, 1979, pp. 358-376

26 E R ERSKINE CROSSLEY
Design for a Three-Fingered Hand
Mechanism and Machine Theory, 1977, Vol 12, pp 85-93

27 B SEGER
Control Systems for Industrial Robots
2nd Conference on Industrial Robot Technology, 1974, pp B1/2-B1/16

28 A J BARBERA, J S ALBUS and M L FITZGERALD
Hierarchical Control of Robots Using Microcomputers
9th International Symposium on Industrial Robots 1979 pp 405-422

29 R COIFFET
Real-Time Problems in Computer Control of Robots
7th International Symposium on Industrial Robots, 1977, pp 145-152

30 R A NORBEDO
Structured Software System for Industrial Automation
7th International Syposium on Industrial Robots, 1977, pp 139-144

31 W E SNYDER et al
Microcomputer control of Manipulators
9th International Symposium on Industrial Robots, 1979, pp423-456

32 W T PARK
Minicomputer Software Organisation for Control of Industrial Robots
Stanford Research Institute, California, Proc JACC, San Francisco CA, 77
pp. 164-171

33 G GINI et al
Introducing Software Systems in Industrial Robots
9th International Symposium on Industrial Robots, 1979, pp. 301-321

34 R PAUL
WAVE: A Model-Based Language for Manipulator Control
Technical Paper MR76-615, Society of Manufacturing Engineers
Dearbon

35 R FINKEL et al
AL, A Programming System for Automation
Memo AIM-243, Stanford University Artificial Intelligence
Laboratory, Stanford, November 1974

36 T LOZANO-PEREZ
The Design of a Mechanical Assembly System
Memo AI-TR-397, MIT Artificial Intelligence Laboratory, Dec 1976

37 L LIEBERMAN
AUTOPASS, A Very High Level Programming Language for Mechanical
Assembler Systems
IBM Research Report RC 5599, No 24205 (1975)

38 D GROSSMAN
Procedural Representation of three Dimensional Objects
IBM Research Report RC-5314

39 P WILL and D GROSSMAN
An Experimental System for Computer Controlled Mechanical Assembly
IEEE Transactions on Computers, Vol C-24, No 9 September 1975,
pp 879-888

40 A D'AURIA and M SALMON
Examples of Applications of the SIGMA Assembly Robot
3rd Conference on Industrial Robot Technology, University of
Nottingham, 1976 pp G5-37-G5-48

41 A AMBLER et al
A Versatile System for Computer-Controlled Assembly
Artificial Intelligence, Vol 6, No 2, Summer 1975, pp 129-156

42 J ALBUS and J EVANS
Robot Systems
Scientific American, Vol 234, No 2 February 1976 pp 83-84

43 R HOLN
Application Flexibility of a Computer-Controlled Industrial Robot
Technical Paper MR76-616 Society of Manufacturing Engineers, Dearbon, Michigan 1976

44 C S CUNNINGHAM
Robot Flexibility through Software
9th International Symposium on Industrial Robots, 1979, pp 297-307

45 W B HEGINBOTHAM et al
Robot Application Simulation
Industrial Robot, June, 1979, pp. 76-80

46 W B HEGINBOTHAM et al
Assessing Robot Performance with Interactive Computer Graphics
Robotics Today, Winter 1979-1980, pp. 33-35

47 G DONATO and A CAMERA
A High Level Programming Language for a New Multi-Arm Assembly Robot
10th International Symposium on Industrial Robots, 1980, pp. 67-75

48 J ASPINALL
The Microprocessor and its Applications
Camb. Uni. Press

49 D H SWAIN
Microprocessors and Microcomputer Systems
Edwards Arnold

50 L RICH
Understanding Microprocessors
Reston

51 J D LANE
Introduction to Microprocessors
Camb. Uni. Press

52 TMS 9900 Family System Development Manual
A Texas Instruments Application Report

53 TM 990/101m Microcomputer User's Guide
Texas Instruments

54 TM 990 Introduction to Microprocessors, Hardware and Software
Texas Instruments

55 J RAGAZZINI and G F FRANKLIN
Sampled-Data Control Systems
Mc Graw-Hill

56 RTI-1240/1241 User's Reference Manual, Texas Instruments

57 RTI-1242/1243 User's Reference Manual, Texas Instruments

58 Operations Handbook for Versatran Point-to-Point Machine
Hawker Siddeley Dynamics Limited

59 C DORF
Modern Control Theory
A-W

60 J SCHWARZENBACH and K F GILL
System Modelling and Control
Edward Arnold

61 Assembly Language Programmer's Guide
Model 990 Computer
TMS 9900 Microprocessor

62 Y HASEGAWA et al
Programming and Teaching Methods for Industrial Robots
4th International Symposium on Industrial Robots, 1974, pp. 301-310

63 H J WARNECKE
Comparative Evaluation of Industrial Robot Accuracy
Precision Engineering, 1980, pp. 89-92

64 S INAKAKI
A Discussion on Positioning Accuracy on Industrial Robots
9th International Symposium on Industrial Robots, 1979, pp. 679-690

65 C MORGAN
The Rationalisation of Robot Testing
10th International Symposium on Industrial Robots, 1980 pp 399-405

66 J F ENGELBERGER
Performance Evaluation of Industrial Robots
Performance Evaluation of Programmable Robots and Manipulator
Conference 1976, pp. 113-120

67 I MASUDA et al
Development of Basic Control Programs for Industrial Robots
with Artificial Intelligence
7th International Symposium on Industrial Robots, 1977, pp. 227-234

68 K W NIELSEN and A G MAKHLIN
Task Selection for - and Development of a Vision Control System
for Industrial Robot Use
10th. Int. Symp. on Ind. Robots, 1980 pp. 495-506

69 J A PEPERSTRAETE
Survey on Microcomputer Architecture
10th International Symposium on Industrial Robots, 1980, pp 31-46

70 J MIDDLETON and R H WESTON
Structured Hardware and Software for Robots
To be published

71 H E MERRIT
Hydraulic Control Systems
John Wiley & Sons Inc

72 A SALIHI
A Microprocessor Based Controller for a Point-to-Point Robot Arm
MSc Thesis 1980 Loughborough University of Technology

73 W V MASON
The Development of a Programmable Controller for Application to Materials Handling Equipment
BSc Project 1980 Loughborough University of Technology

74 The Engineering Staff of Texas Instruments
TX 990 Operating Systems

75 The Engineering Staff of Texas Instruments
Assembly Language Programmers Guide

76 G.P. CHARLES & R.H. WESTON
Microprocessor Controls for Limited-Function Robots
The Radio and Electronic Engineer, Vol 52, No.1, pp41-45

APPENDIX ONE

ROBOT ECONOMICS

The success of any commercial industrial undertaking has to be measured in terms of financial performance. The most brilliant technical innovation is a failure if it results in money lost by the entrepreneur or its shareholders or at divisional or operating level. Robots are no exception ot this rule. No matter what the social benefits are, no matter how advanced the technology, every proposed investment in robotics has to pass the test of critical financial appraisal.

The following headings provide a framework for management analysis of the costs and benefits of the robotics installation.

1 Robot Costs:

- a) Purchase price of the robot
- b) Special tooling
- c) Installation
- d) Maintenance and periodic overhaul
- e) Operating power
- f) Finance
- g) Depreciation

2 Robot savings:

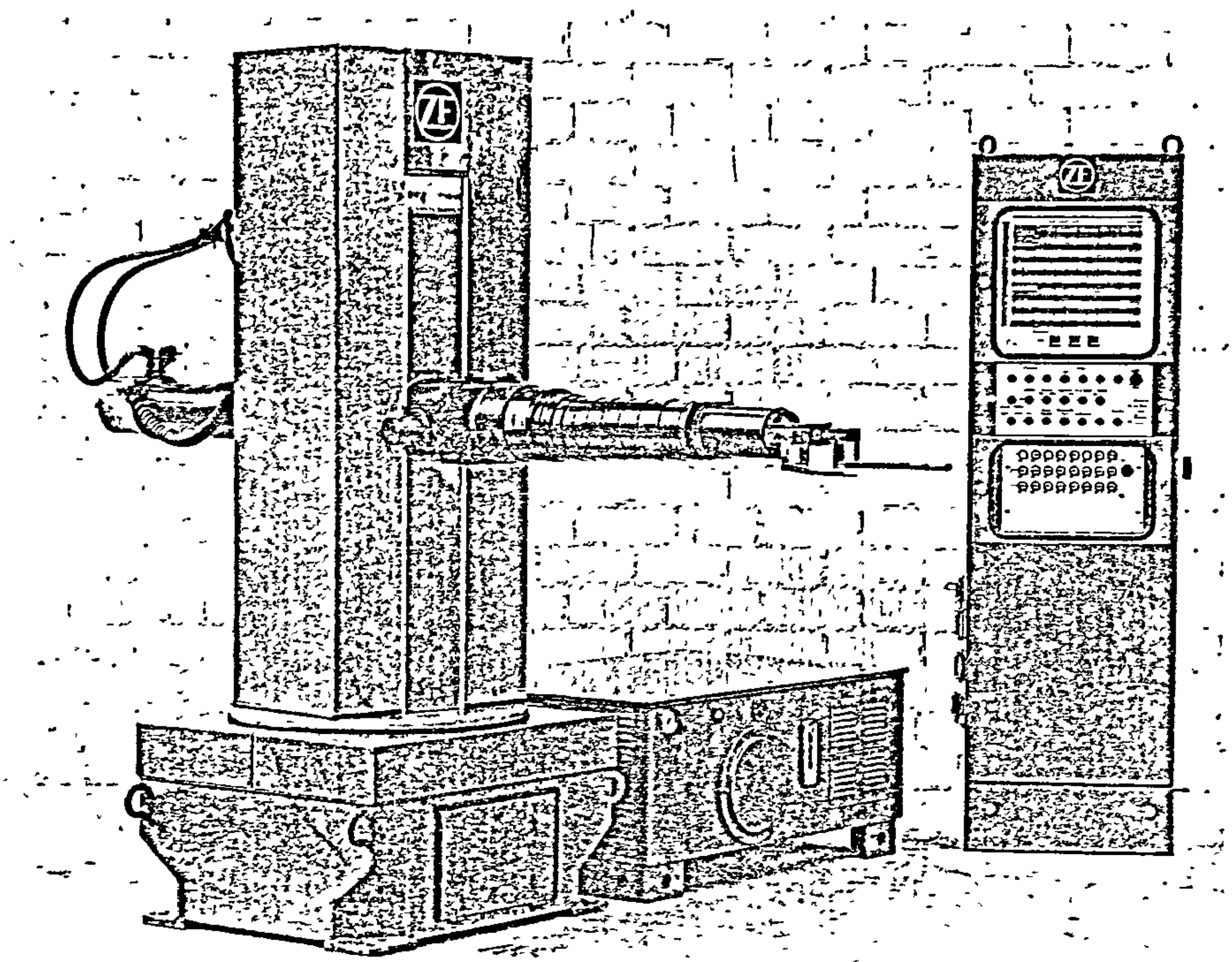
- a) Labour displaced
- b) Quality improvement
- c) Increase in throughput

APPENDIX TWO

SPECIFICATIONS FOR INDUSTRIAL ROBOTS

This appendix contains specifications for various industrial robots. Section one contains the simpler point-to-point robots and section two the continuous path robots suitable for welding and painting.

ZF Handling Technology
Handling Robot T III
for loads up to 40 kg



echnical data

ZF Handling Robot T III, Type CXZ-A 1060 and type CXZ-A 1260

Application: an all-purpose unit, particularly suitable for plants where considerable heat is generated, e.g. forges, hardening shops, injection molding shops etc.

Design: 3 main axes (C, X, Z) } protected against dirt and heat
1 gripper axe (A)

Standard gripper hydraulically actuated, with 40° clamping range
Special gripper available on request, with pneumatic, magnetic or vacuum actuation

.oads: component weight up to 40 kg.

lain and gripper movement axis characteristics:

Axes / Characteristics		C (slewing)	X (horizontal stroke)	Z (vertical stroke)	A (slewing)
Working range	type CXZ-A 1060	200° or 280°	1 000 mm	600 mm	360°
	type CXZ-A 1260	200° or 280°	1 200 mm	600 mm	360°
Mean speed*		110°/s	1 200 mm/s	800 mm/s	120°/s
Position setting reproducibility **		$\leqq$ 2 mm	$\leqq$ 2 mm	$\leqq$ 2 mm	$\leqq$ 0.1 mm

Load weight can be increased by operating at lower speeds

* Reproducibility can be rendered more accurate by heating the hydraulic fluid before operation commences.

lectrical connection rating 11 kW

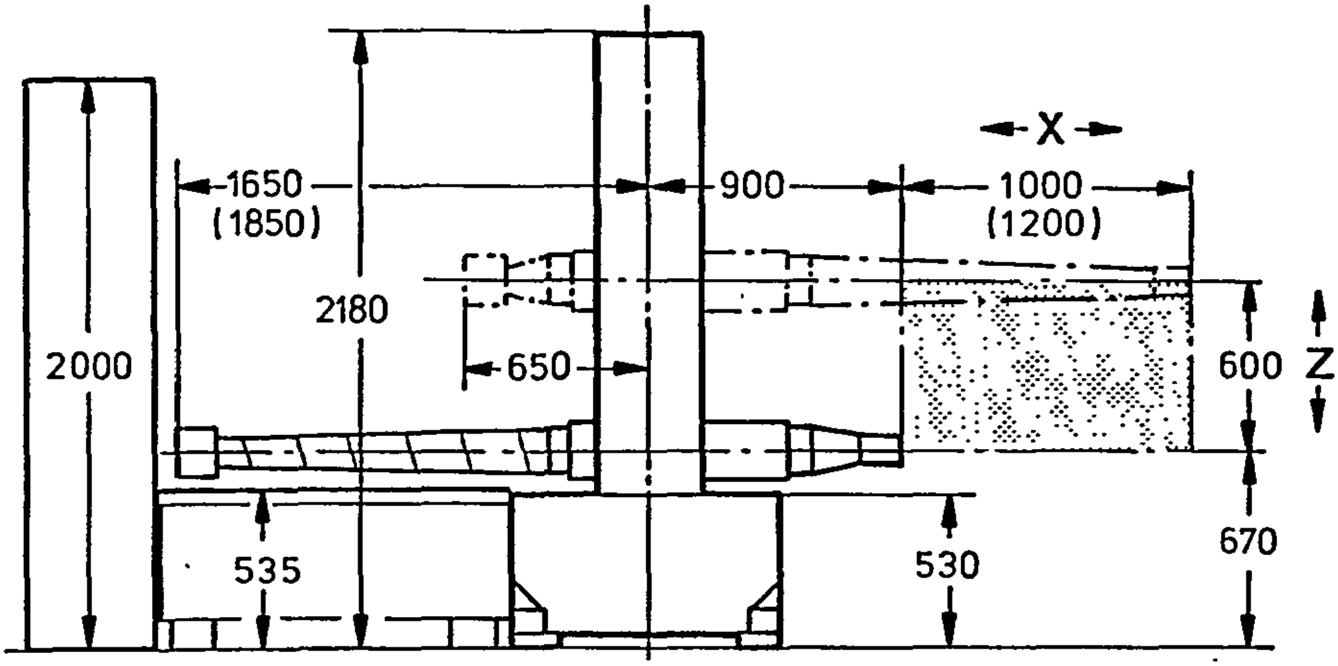
ontrol system: Optimum suitability for various types of work is assured by provision being made for various forms of control:

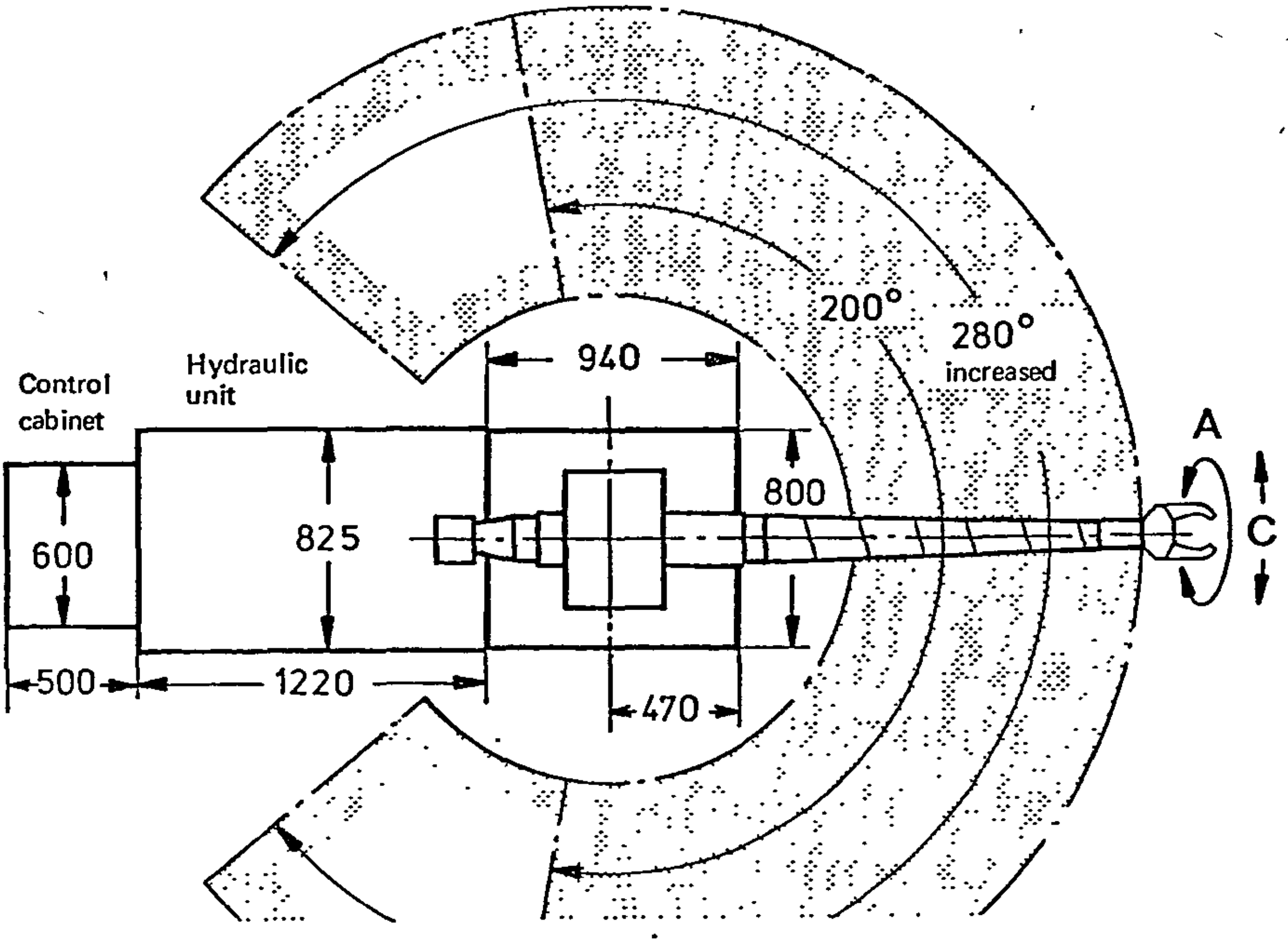
Positioning (alternatives)
- with adjustable fixed stops (2 positions/axis)
- with cam shutdown (up to 6 positions/axis)
- with servo-hydraulic PTP control (up to 8 positions/axis), adjustable via digital-display set-value potentiometers

Program sequence and programming (alternatives)
- PC system (free programming). The program sequence can be programmed via a crossbar distributor with diode matrix store or a direct PC program with EPROM memory.
- by NC microprocessor control with teach-in programming (PTP positioning control, 200 points/axis)

lasses:

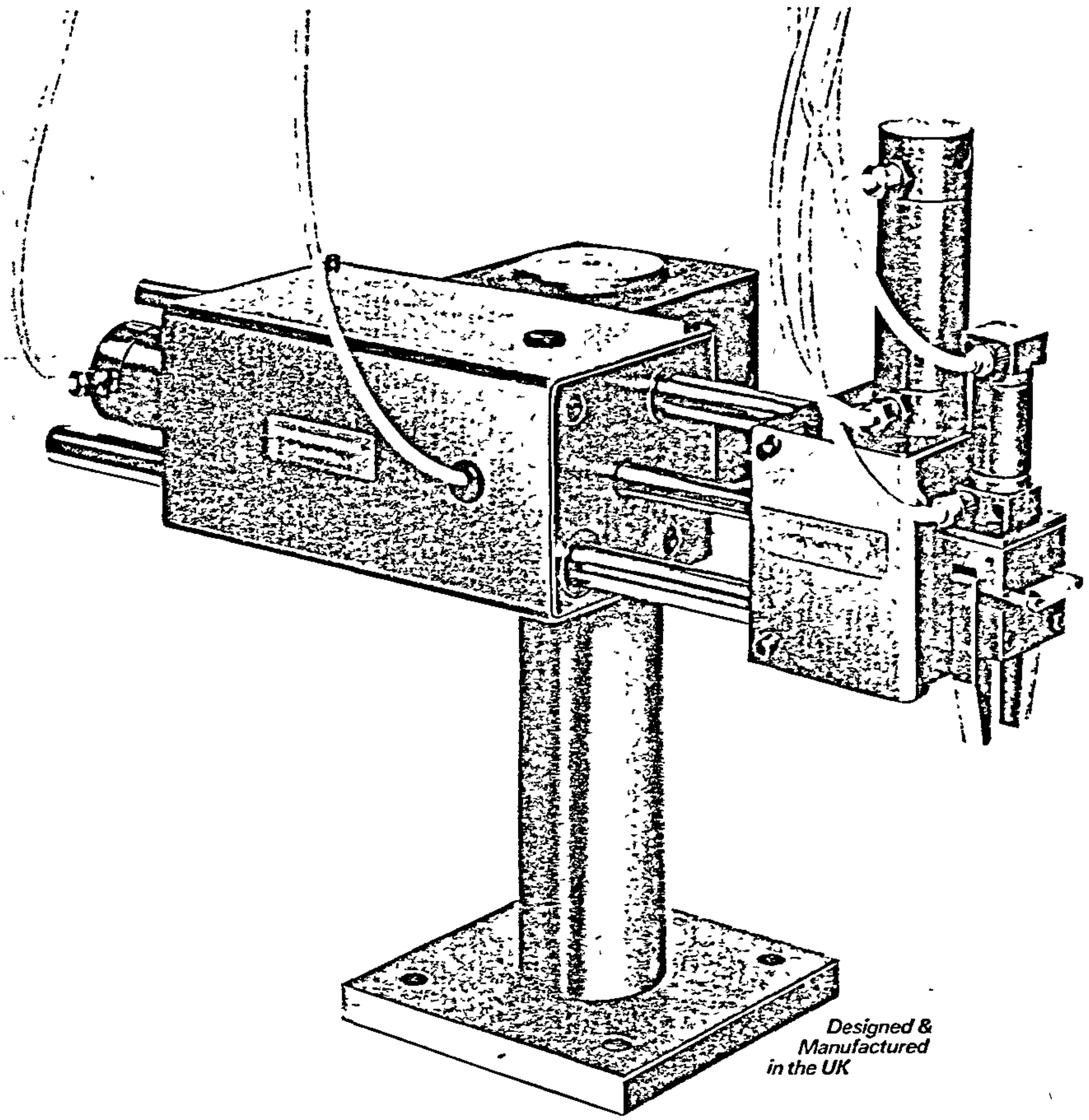
Handling robot	app. 1 200 kg
Hydraulic unit	app. 370 kg (including 150 dm^3 of hydraulic fluid)
Control cabinet	app. 100 kg


X
1650
(1850)
900
1000
(1200)
2180
2000
650
600
Z
535
530
670


200°
280°
increased
940
Control
cabinet
Hydraulic
unit
A
C
800
600
825
500
1220
470

Precision Pick & Placement Robot

Valley Automation Ltd.
Valley Road, Lye,
Stourbridge, West Midlands DY9 8JH, England
Telephone Lye (038-482) 2324/2419 Telegrams Lye 2
Telex 338212 CHAMCOM G Code VALLEY



Designed & Manufactured in the UK

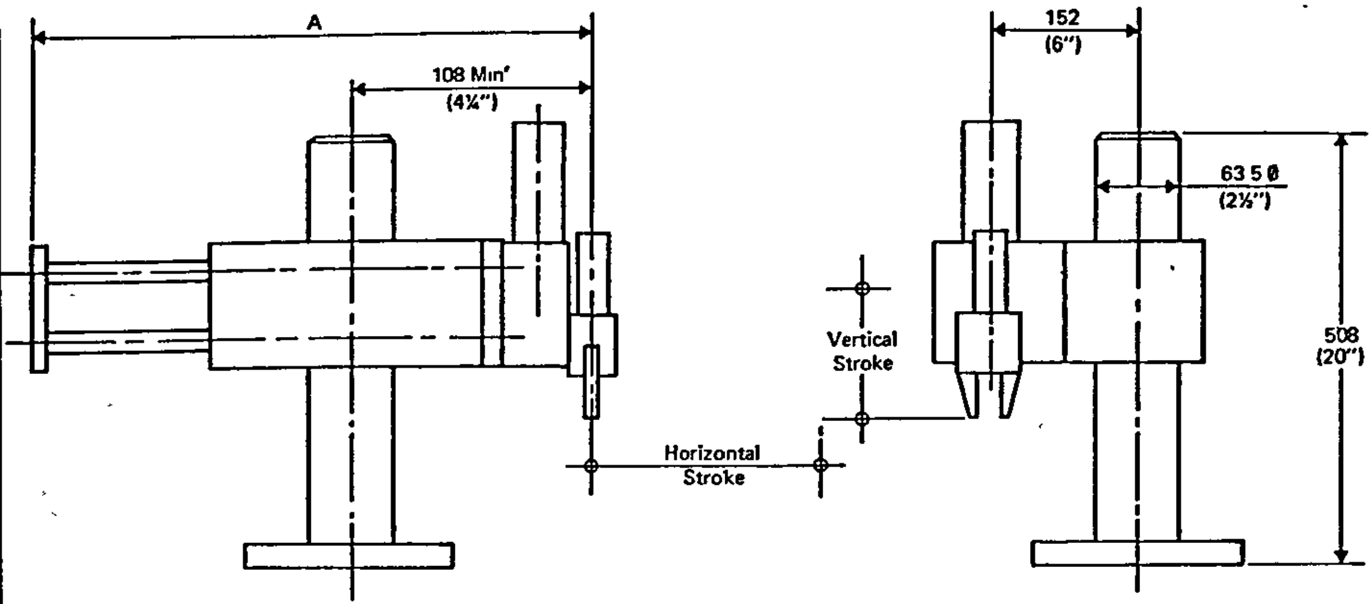
This pick and placement robot is pneumatically operated and provides an ideal solution to many component handling problems such as automatic assembly and machine loading.
It is designed and constructed to offer very high repeatable accuracy. Both the horizontal and vertical movements are carried out through precision linear bearings. There are a range of horizontal and vertical stroke lengths.

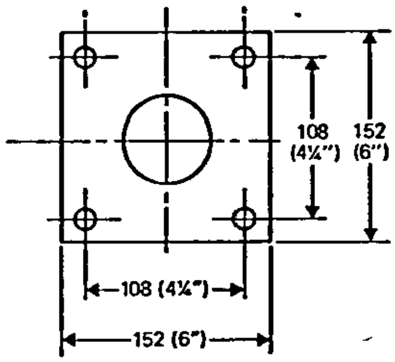
The standard pick-up heads can be either pneumatically operated jaws, electro-magnetic or vacuum heads. The pick-up head units are designed to suit the specific applications required.

The units can be either controlled by a cyclic cam timer which in turn operates a series of solonoid air valves or alternatively it can be controlled via a programmeable sequential controller. Whichever method is utilised, the unit is supplied complete with all necessary control equipment.

As this robot is adaptable to many applications, a complete technical advisory service is always available and it augments the already wide range of component handling and orientation equipment manufactured by Valley Automation.

General dimensions of the Precision Pick and Placement Robot.





Horizontal Stroke ins mm	A ins. mm
2 50	14 355
4 100	16 406
6 150	18 457

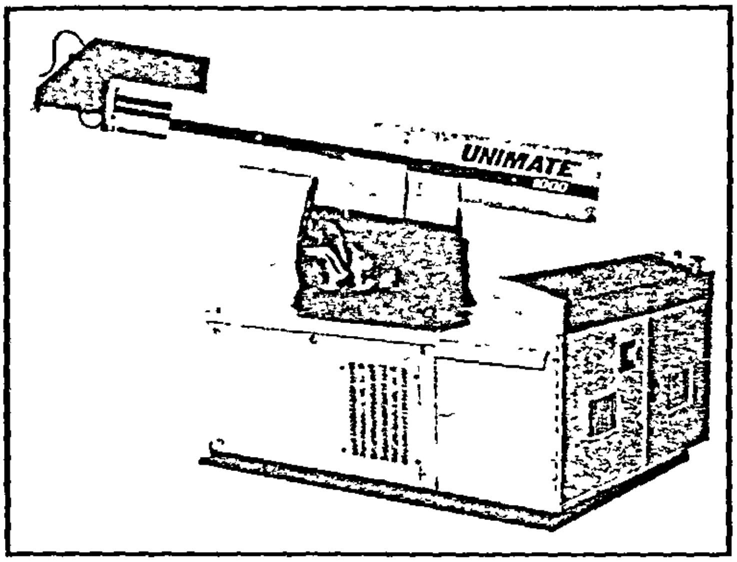
Vertical Stroke ins. mm
1 25
2 50

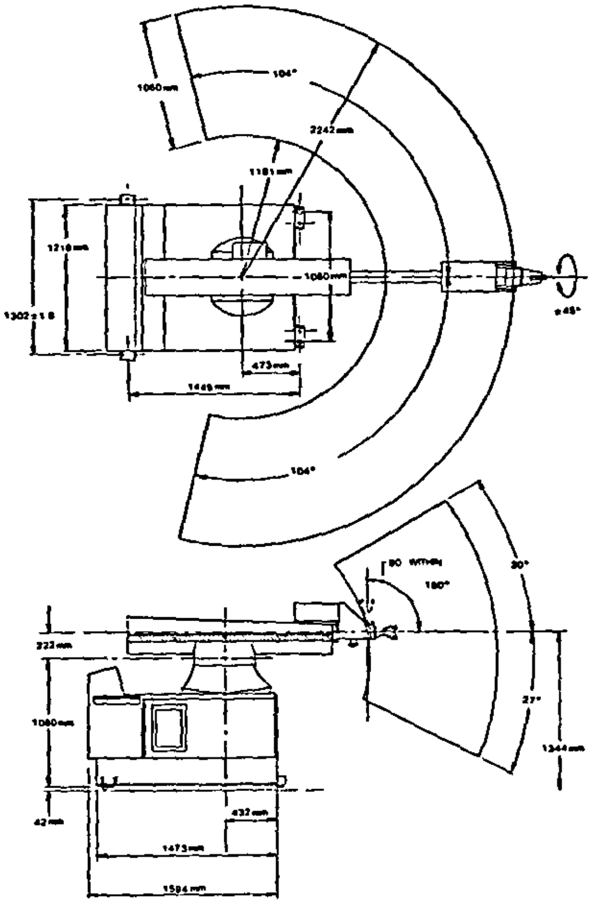
Dimensions are subject to change

Valley Automation Ltd.
Valley Road, Lye,
Stourbridge, West Midlands DY9 8JH, England
Telephone Lye (038-482) 2324/2419 Telegrams Lye 2324
Telex· 338212 CHAMCOM G Code VALLEY

1000

The low cost Series 1000 UNIMATE® offers superior performance for jobs that require only limited handling. It's the ideal tool for operations where lifting requirements are less than 22 kgs. The 1000 Series robots have five axes, three of them hydraulically powered. Gripper and wrist movements are pneumatically operated working between adjustable end stops. Fully extended the Series 1000 UNIMATE® robots have a reach of 2250mm. Programming is done through a plug -in teach control offering "lead-by-the-hand" simplicity.





Typical Applications

Materials handling, plastic injection moulding, machine loading, die casting, press loading and load/unload machine tools.

MODEL SPECIFICATION FOR UNIMATE 1000

Manipulator Wt	1200Kg
Hydraulic Supply Wt (with fluid)	Integral
Control Cabinet Wt	Integral
Mounting Position	Floor
No of Degrees of Freedom	3-5
Positioning Repeatability	1.27mm
Power Requirements	380/415/525, 3Ø
	$50H_3$, 10KVA
Point-to-point	Up to 256 Points
WRIST TORQUE	
Bend	$5.7Kgm^{-1}$
Yaw	$1.7Kgm^{-1}$
Swivel	N/A

SPECIFICATION: IRb 6 ASEA

ARM MOTIONS, STROKES AND SPEEDS:

Right - left	340°	95°/sec
Up - down	800mm	750mm/sec
Out - in	560mm	1100mm/sec
Traverse		

WRIST MOTIONS, STROKES AND SPEEDS:

Revolution	±180°	195°/sec
Swing (right-left)		
Bend (up-down)	±90°	115°/sec

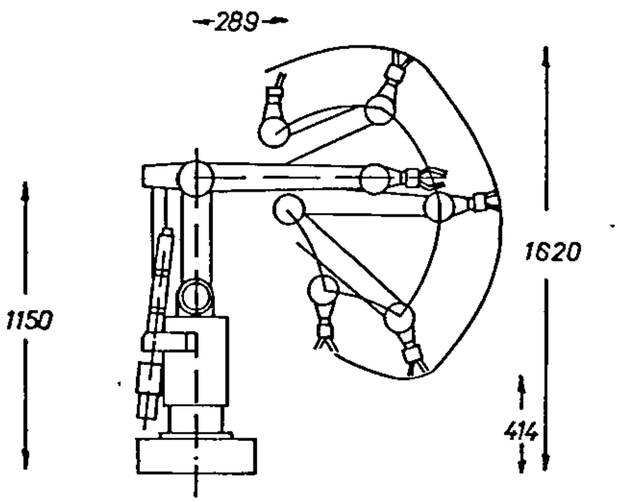
CONTROL FUNCTION:

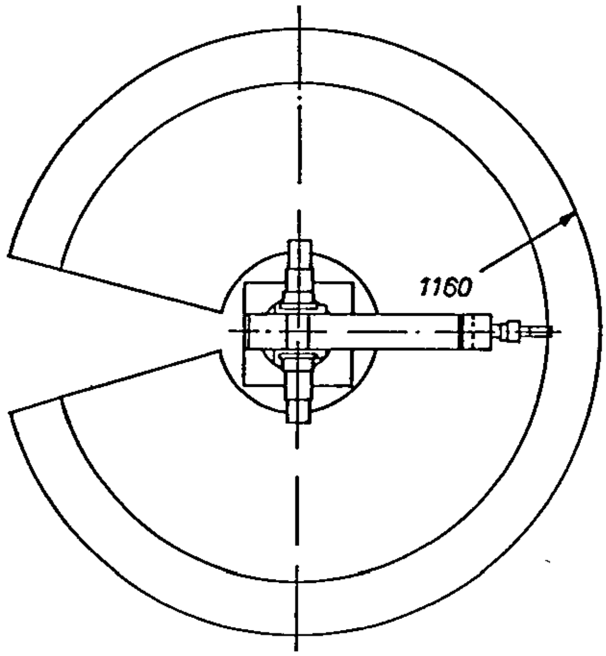
Motion control	CP by PTP teaching
Memory systems	Semi-conductor type plus magnetic tape
Memory capacity	250 points (basic)

POSITIONING ACCURACY: ±0.2mm

CONDITIONS FOR INSTALLATION:

Dimensions (length x width x height)	720 x 720 x 1620mm
Weight	300kg
Power requirements	2kVA
Temperature	40°C
Source of driving power	Electric





SPECIFICATION: BOC/HAL BOC

ARM MOTIONS, STROKES AND SPEEDS:

Right - left	85°	30°/sec
Up - down	70°	30°/sec
Out - in	914mm	150mm/sec
Traverse		

WRIST MOTIONS, STROKES AND SPEEDS:

Revolution		
Swing (right-left)	180°	90°/sec
Bend (up-down)	180°	90°/sec

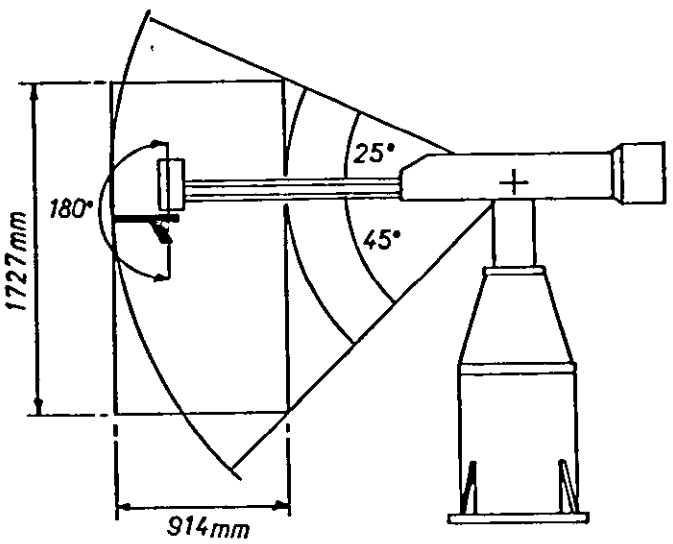
CONTROL FUNCTION:

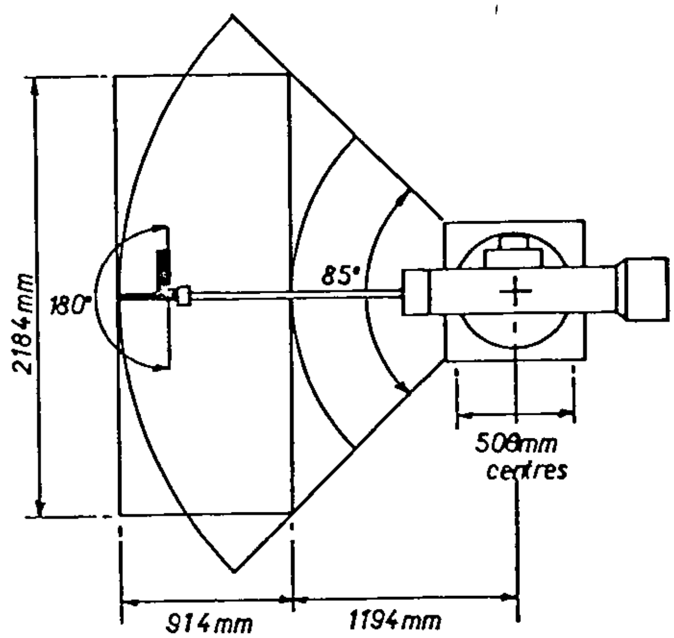
Motion control	CP
Memory systems	Solid state non-volatile
Memory capacity	10min/module (max 15 modules)

POSITIONING ACCURACY: ±1.5mm

CONDITIONS FOR INSTALLATION:

Dimensions (length x width x height)	610 x 610 x 2032mm
Weight	527kg
Power requirements	220/440V
Temperature	
Source of driving power	Hydraulic





SPECIFICATION: T3 CINCINNATI MILACRON

ARM MOTIONS, STROKES AND SPEEDS:

Right - left	240°
Up - down	3962mm
Out - in	1424mm
Traverse	

WRIST MOTIONS, STROKES AND SPEEDS:

Revolution	240°	1270mm/sec
Swing (right-left)	180°	For tool
Bend (up-down)	190°	Centre point

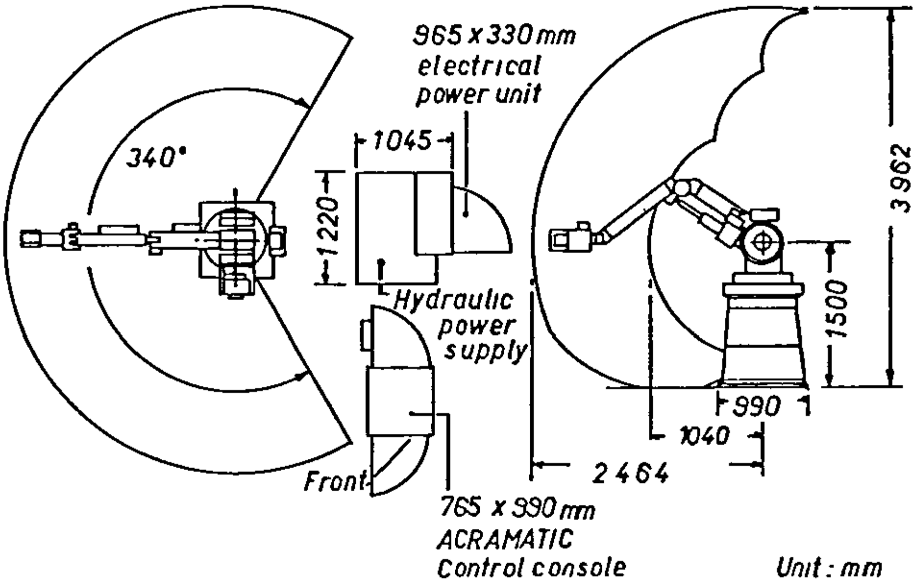
CONTROL FUNCTION:

Motion control	CP by PTP teaching
Memory systems	Acromatic computer plus magnetic tape
Memory capacity	700 points

POSITIONING ACCURACY: ± 1.27mm

CONDITIONS FOR INSTALLATION

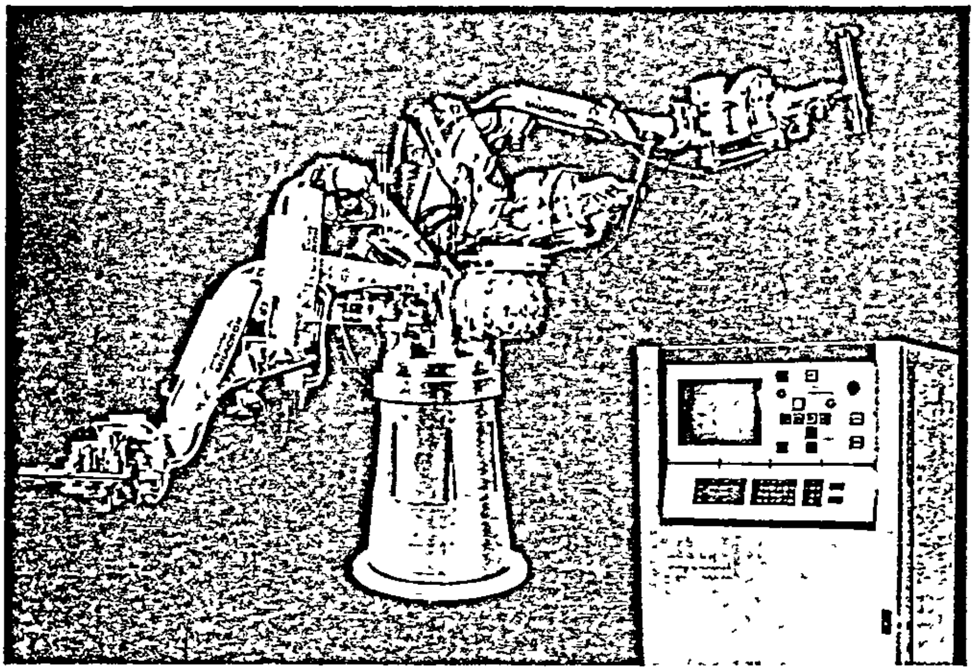
Dimensions (Length x width x height)	990 x 990 x 2000mm
Weight	2267kg
Power requirements	22kVA
Temperature	50°C
Source of driving power	Hydraulic



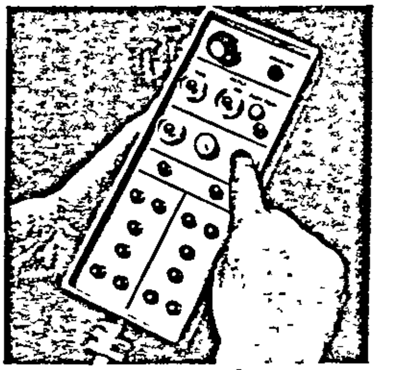
CINCINNA
MILACR

T³ Computer-Controlled Industrial Robot (Standard Model)

Offers the durability, reach, freedom of motion and strength to do the most grueling job around the clock no matter how hazardous the working conditions.



T³ Computer-Controlled Industrial Robot THE TOMORROW TOOL Today (Inset) Cincinnati Milacron ACRAMATIC Robot Control



Portable, light-weight, hand-held teach unit provides convenient means of programming the robot

The T³ is a simple, solidly built 6-axis computer-controlled industrial robot It combines a heavy base casting with strong shoulder, upper arm, and forearm fabrications for total structure ruggedness and stability

Unique Jointed-Arm Construction
Exclusive with T³, this unique 6-axis jointed-arm construction provides the added flexibility the robot needs in order to perform in difficult-to-reach places Duplicating the dexterity of the human arm/hand, T³'s jointed-arm is tougher by far, well able to withstand the most hostile industrial environment to get the job done ... day in, day out ... with astonishing reliability Sealed-for-life lubrication and rotary joints with large antifriction bearings result in minimal wear and virtually maintenance-free operation.

Powerful Direct Drives
Each of the six jointed-arm axes of the T³ is direct driven by its own powerful and independent electro-hydraulic servo system. Five of the axes use compact rotary actuators built-into each joint and one axis is driven by a pivoted cylinder This construction gives the robot a backlash-free system capable of the high torque, speed and flexibility needed to handle hefty payloads with up to 240° of movement Tests prove that T³ can easily lift 100-lb. loads three shifts a day at speeds up to 50 ips

Precise Position Feedback
Each axis also has its own position feedback device consisting of a resolver and tachometer to assure repeatable and precise arm positioning Accuracy to any programmed point is ±0 050".

Cost-Effective Straight-Line Motion
The powerful logic of the robot's reliable ACRAMATIC minicomputer-based control provides infinitely variable 6-axis positioning and controlled path (straight-line) motion between programmed points All of T³'s jointed-arm motion is referenced to the Tool Center Point (TCP) a discrete point at a selectable distance from the arm where the tool meets the work All TCP moves are made in a cost-effective straight-line

"Teaching" T³ Is Fast and Easy
No computer experience is needed, no calculations are involved — just knowledge of the physical job to be performed. A lightweight, hand-held unit lets the operator program the T³ from the best vantage point Optional offset branching further simplifies the teaching function in that a series of repetitive moves can be "taught" as a subroutine just once Jobs requiring len teaching sessions or recurring jobs can easily committed to the robot's semiconductor memory and stored on t optional tape cassette for future use

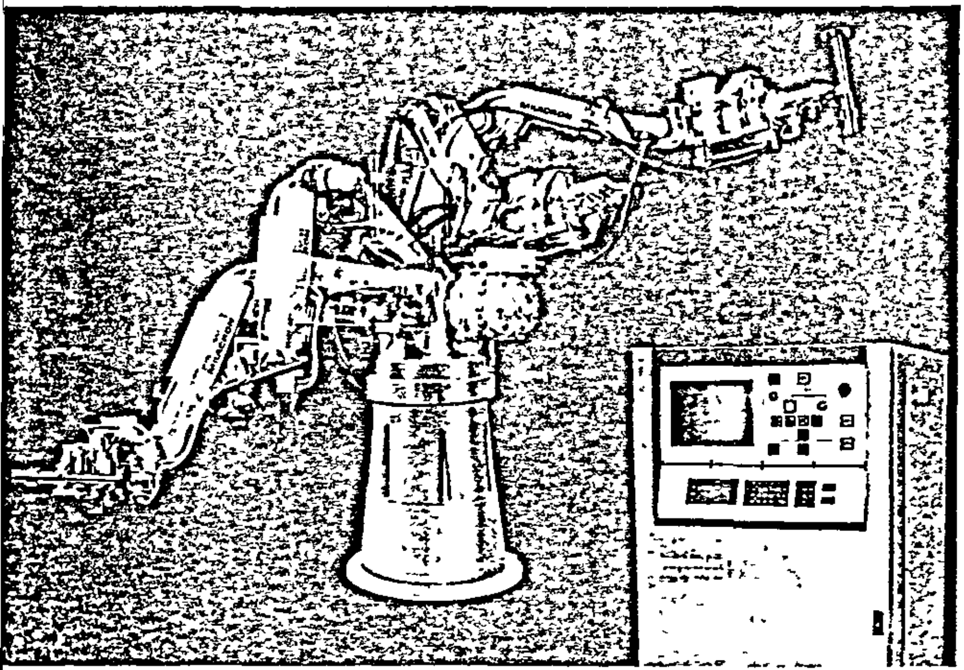
Wide Application Flexibility
Easy to program ... to tool ... to use wit pallet-oriented work .. the T³ can smoo track moving lines, and while tracking, p welds in precisely the right spots, or plu assembly out of a moving welding jig ar hang it high overhead on a moving conveyor, tracking the two continuously moving lines independently of one anot T³ can reach with ease into tight places multiple levels, with one hand or two, at virtually any angle, anywhere within 10(cu ft of volume — inside an auto chass under a hood, down deep into boxes, o straight out 97" to load parts onto one o more metalcutting or metalforming machines.

Industrial Robot Division,
Cincinnati Milacron Ltd ,
Caxton Road
Bedford MK 41 OHT, England.
Phone 0234-45221

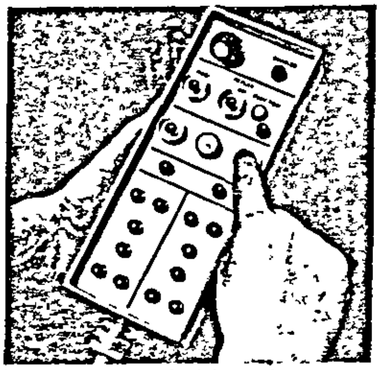
CINCINNATI
MILACRON

HT³ Computer-Controlled Industrial Robot (Heavy Duty Model)

Offers the durability, reach, freedom of motion and extra strength to do heavy-duty jobs around the clock no matter how hazardous the working conditions



HT³ Computer-Controlled Industrial Robot . THE TOMORROW TOOL Today (Inset) Cincinnati Milacron ACRAMATIC Robot Control



Portable, hand-held teach unit provides convenient means for the operator to program the robot

Additional Payload Capability
The HT³ is a heavy duty model computer-controlled industrial robot capable of additional load-carrying beyond the limits of the standard model T³ robot Ratings for the HT³ indicate a load capacity of 225 lbs at 10" from the tool mounting plate and a maximum velocity of 35 ips at full load

Powerful Direct Drives — Double the Torque
Each of the six axes of the HT³ is direct driven by its own powerful and independent electro-hydraulic servo system. Five of the axes use compact rotary actuators built-into each joint and one axis (the elbow) is driven by a pivoted cylinder This construction gives the robot a backlash-free system capable of the high torque, speed and flexibility needed to handle hefty payloads with up to 240° of movement Actuator displacement or torque for each of the six hydraulic components is double the value of those used in the standard T³ model. A dual vane is used in the shoulder actuator which permits 90° motion from near horizontal to slightly over vertical position

Precise Position Feedback
Each axis also has its own position feedback device consisting of a resolver and tachometer to assure repeatable and precise arm positioning Accuracy to any programmed point is ±0 050".

Unique Jointed-Arm Construction
Duplicating the dexterity of the human arm, HT³'s unique jointed-arm construction is tougher by far, well able to withstand the most hostile industrial environment to get the job done ... day in, day out ... with astonishing reliability Sealed-for-life lubrication and rotary joints with large antifriction bearings result in minimal wear and virtually maintenance-free operation.

Cost-Effective Straight-Line Motion
The powerful logic of the robot's reliable ACRAMATIC minicomputer-based control provides infinitely variable 6-axis positioning and controlled path (straight-line) motion between programmed points All of HT³'s jointed-arm motion is referenced to the Tool Center Point (TCP) .. a discrete point at a selectable distance from the arm where the tool meets the work All TCP moves are made in a cost-effective straight-line.

"Teaching" HT³ Is Fast and Easy
No computer experience is needed, no calculations are involved — just knowledge of the physical job to be performed A lightweight, hand-held unit connected to the control console by a 33' long flexible cord lets the opeator program the HT³ from the best vantage point Jobs requiring lengthy teaching sessions or recurring jobs can be easily committed to the robot's semiconductor memory and stored on the optional tape cassette for future use.

Wide Application Flexibility
Easy to program ... to tool ... to use with pallet-oriented heavy work .. the HT³ can for example, smoothly track moving lines, and while tracking, pluck an assembly weighing as much as 225 lbs. out of a mov ing welding jig and hang it high overhead c a moving conveyor, tracking the two con- tinuously moving lines independently of or another HT³ can reach with skillful accura into confined locations at multiple levels, with one hand or two, at virtually any angle inside an auto chassis, under a hood, dow deep into boxes, or straight out 97" to load large, heavy parts onto one or more metal cutting or metalforming machines.

Industrial Robot Division,
Cincinnati Milacron Ltd ,
Caxton Road
Bedford MK 41 OHT, England
Phone 0234-45221

... (HT³) Computer-Controlled Industrial Robot

›tra strong...smart...swift...spacesaving...reliable offers wide application flexibility

›ecs for HT³ Heavy Duty Model

ad capacity
ad 10" (254 mm) from tool mounting plate ..... 225 lb. (102 kg)*

sitioning accuracy, axis drive
curacy to any programmed point .... ±0 050 in (±1 27 mm)
ve for each of 6 axes .... direct, electrohydraulic

inted arm motions, range, velocity
ximum horizontal sweep .... 240°
ximum horizontal reach .... 102" (2591 mm) to tool mounting plate
n to max reach, floor to ceiling .... 0" to 154" (0 to 3911 mm)
ximum velocity of TCP .... 35 ips (890 mmps)
ch .... 180°
ll .... 240°
w .... 180°

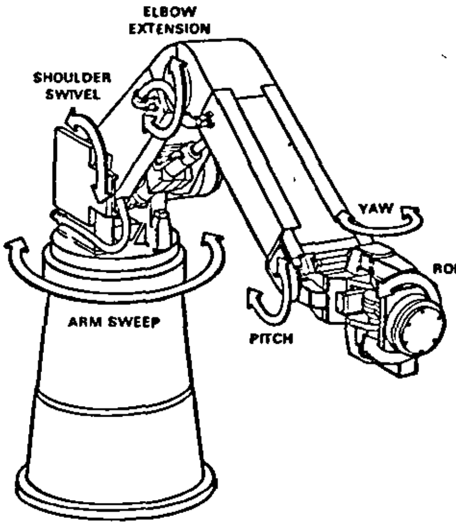
or space and approximate net weight
bot .... 9 sq ft (0 8 sqm); 5,000 lb (2267 kg)
draulic power supply. .... 17 sq ft (1 5 sqm), 1,200 lb (544 kg)
ctrical power unit .... 3.4 sq ft (0 3 sqm), 700 lb. (317 kg)
RAMATIC computer control .... 8 3 sq ft. (0 8 sqm), 800 lb (365 kg)

wer requirments .... 230/460 volts, 3 phase, 60 Hz, 32 KVA

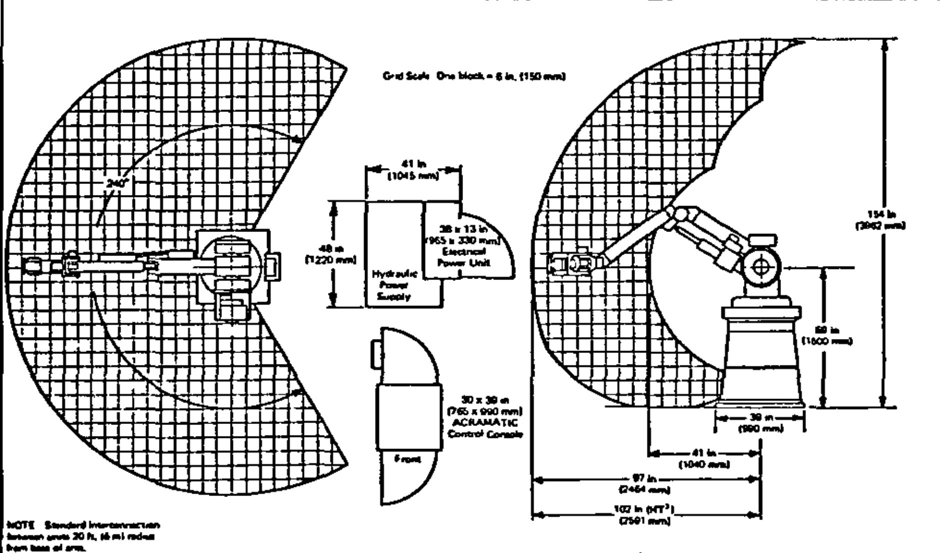
vironmental temperature .... 40 to 120°F (5 to 50°C)

NICOMPUTER memory capacity .... 700 points std

onsult factory for special applications



Maneuverability of the 6-axis jointed-arm increases productivity of all stationary-bas… line tracking operations



Basic range and floor space drawings

All illustrations and specifications containe… in this literature are based on the latest product information available at the time of publication The right is reserved to make changes at any time without notice in price… materials, equipment, specifications, and models, and to discontinue models In additions, all nominal dimensions are subject to an allowable variation of ±0 25-in. (6 mm), unless otherwise specified

WARNING. In order to clearly show details of this machine, some covers, shields, doors, and guards have either been removed or shown in an "open" position Furthermore, operators are shown ONLY t… indicate relative product size, they may be … positions which are NOT the normal or saf… operating positions Be sure that all protective devices are properly installed before operating this equipment

blication No. A-276

CINCINNATI MILACRON
Cincinnati Ohio 45209

Printed in U S A
5M 1079 OZ

SPECIFICATION: Mr Aros   HITACHI LTD

ARM MOTIONS, STROKES AND SPEEDS:

Right - left	+90°	70-700mm/min
Up - down	1300mm	70-700mm/min
Out - in	1100mm	70-700mm/min
Traverse	2000mm	70-700mm/min

WRIST MOTIONS, STROKES AND SPEEDS:

Revolution		
Swing (right-left)		
Bend (up-down)	-50° -50°	70-700mm/min

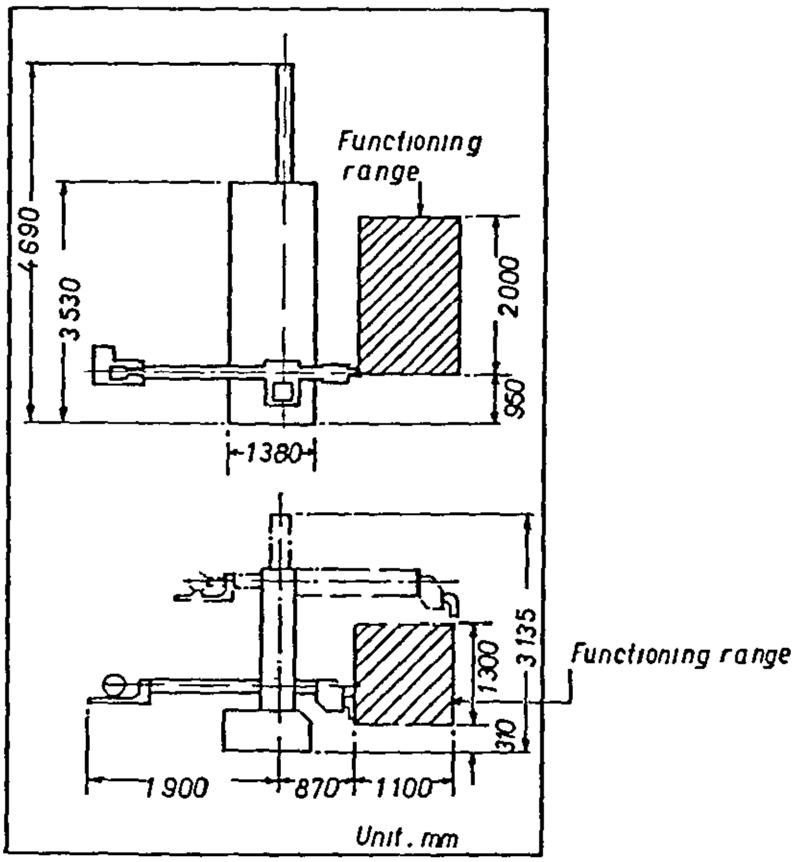
CONTROL FUNCTION:

Motion control	CP based on PTP teaching system
Memory systems	Computer plus sensing system
Memory capacity	512 steps

POSITIONING ACCURACY: ±1.0mm

CONDITIONS FOR INSTALLATION:

Dimensions (length x width x height)	1380 x 4690 x 3135mm
Weight	1500kg
Power requirements	2.5kVA
Temperature	0-50°C
Source of driving power	Electric/oil-hydraulic



SPECIFICATION: UNIMAN 4000 KUKA NACHI

ARM MOTIONS, STROKES AND SPEEDS:

Right - left	1200mm	250mm/sec
Up - down	760mm	250mm/sec
Out - in	760mm	250mm/sec
Traverse		

WRIST MOTIONS, STROKES AND SPEEDS:

Revolution	200°	90°/sec
Swing (right-left)	200°	90°/sec
Bend (up-down)		

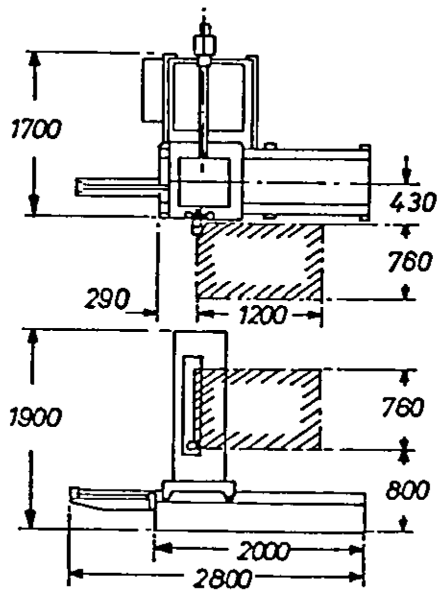
CONTROL FUNCTION:

Motion control	PTP
Memory systems	Magnetic disc
Memory capacity	3199 points

POSITIONING ACCURACY: ±0.5mm

CONDITIONS FOR INSTALLATION:

Dimensions (length x width x height)	1700 x 2800 x 1900mm
Weight	1350kg
Power requirements	5kVA
Temperature	45°C
Source of driving power	Hydraulic



SPECIFICATION: R50 LANGUEPIN

ARM MOTIONS, STROKES AND SPEEDS:

Right - left	1200, 1600, 2000	500mm/sec
Up - down	800	500mm/sec
Out - in	1200	500mm/sec
Traverse		

WRIST MOTIONS, STROKES AND SPEEDS:

Revolution	400°	150°/sec
Swing (right-left)	210°	150°/sec
Bend (up-down)	400°	150°/sec

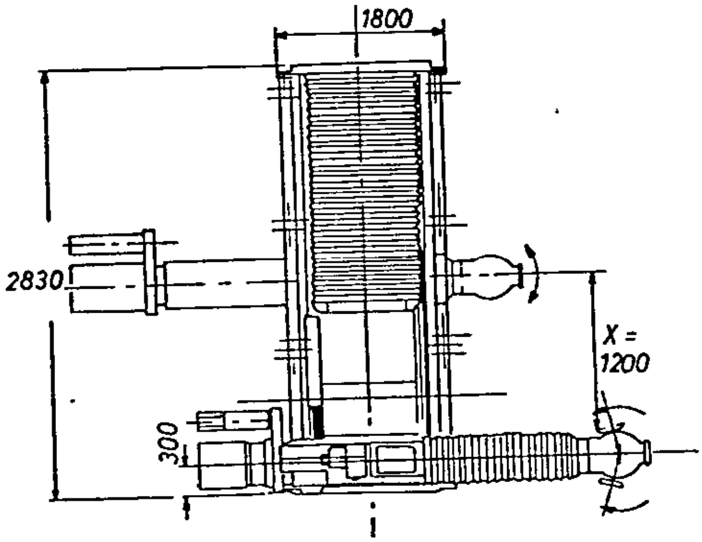
CONTROL FUNCTION:

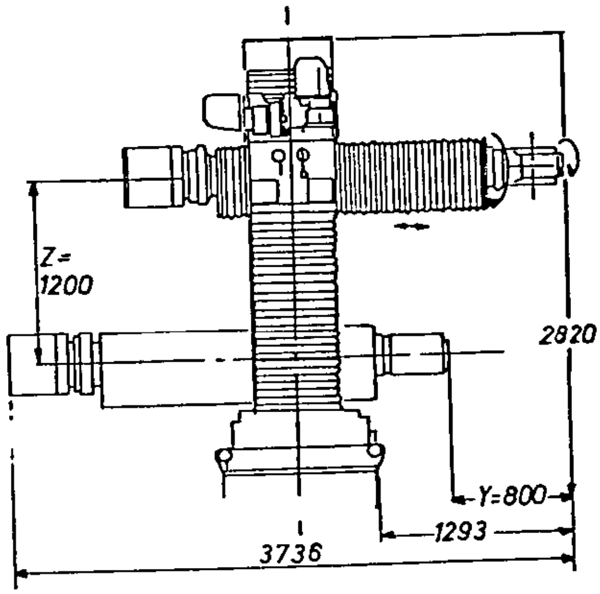
Motion control	CP based on PTP teaching
Memory systems	Ferrite core
Memory capacity	

POSITIONING ACCURACY: ±0.5mm

CONDITIONS FOR INSTALLATION:

Dimensions (length x width x height)	2830 x 1800 x 2820mm
Weight	2000kg
Power requirements	12kVA
Temperature	45°C
Source of driving power	Electric





SPECIFICATION: PW 751 SHIN MEIWA

ARM MOTIONS, STROKES AND SPEEDS:

Right - left	750mm	75mm/sec
Up - down	750mm	in 16 increments
Out - in	750mm	
Traverse		

WRIST MOTIONS, STROKES AND SPEEDS:

Revolution	560°	
Swing (right-left)		
Bend (up-down)	400°	28°/sec

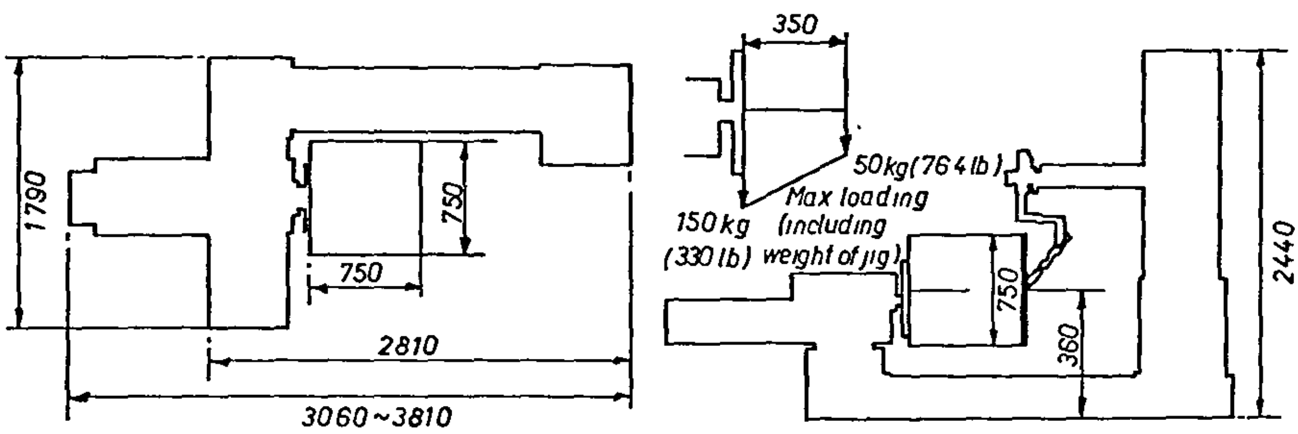
CONTROL FUNCTION:

Motion control	PTP interpolation
Memory systems	Core memory plus recorder
Memory capacity	470 steps

POSITIONING ACCURACY: ±0.5mm

CONDITIONS FOR INSTALLATION

Dimensions (length x width x height)	3810 x 1790 x 2440mm
Weight	2000kg
Power requirements	1.0kW
Temperature	40°C
Source of driving power	Electric servo motors



SPECIFICATION: TOSMAN TOKYO SHIBAURA ELECTRIC CO LTD

ARM MOTIONS, STROKES AND SPEEDS:

Right - left	220°	90°/sec
Up - down	60°	30°/sec
Out - in	700mm	700mm/sec
Traverse		

WRIST MOTIONS, STROKES AND SPEEDS:

Revolution	220°	90°/sec
Swing (right-left)		
Bend (up-down)	220°	90°/sec

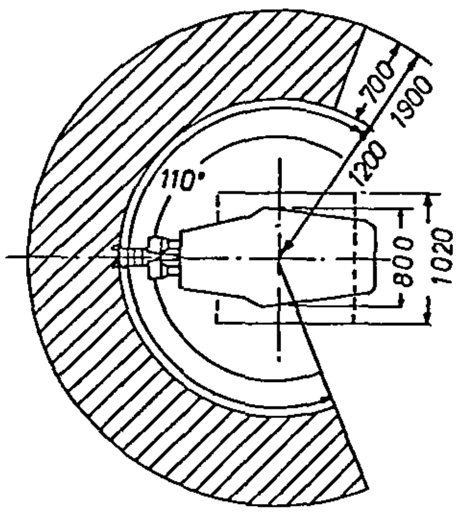
CONTROL FUNCTION:

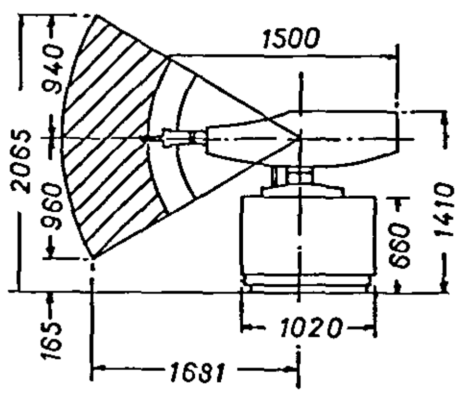
Motion control	PTP
Memory systems	Wire memory
Memory capacity	512 steps

POSITIONING ACCURACY: ±1.0mm

CONDITIONS FOR INSTALLATION

Dimensions (length x width x height)	1020 x 1020 x 1410
Weight	600kg
Power requirements	200V
Temperature	$40^{\circ}C$
Source of driving power	Hydraulic





SPECIFICATION: TRALLFA TRALLFA

ARM MOTIONS, STROKES AND SPEEDS:

Right - left	93°	(3150mm)
Up - down	72°	(2040mm)
Out - in	75°	(975mm)
Traverse		

WRIST MOTIONS, STROKES AND SPEEDS:

Revolution	
Swing (right-left)	210°
Bend (up-down)	210°

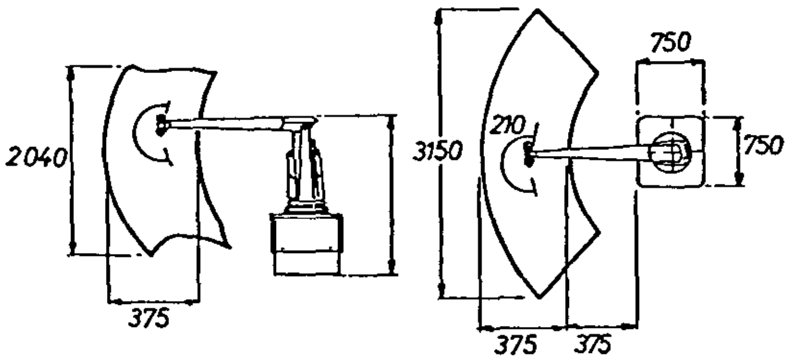
CONTROL FUNCTION:

Motion control	CP and PTP
Memory systems	Magnetic tape and Trallfa CRC
Memory capacity	up to 2hr

POSITIONING ACCURACY: ±2.0mm

CONDITIONS FOR INSTALLATION:

Dimensions (length x width x height)		1750 x 750 x 1600mm
Weight		450kg
Power requirements		7kVA
Temperature	40	40°C
Source of driving power		Hydraulic



SPECIFICATION: 2040 UNIMATE

ARM MOTIONS, STROKES AND SPEEDS:

Right - left	220^{0}	110^{0}/sec
Up - down	57^{0}	30^{0}/sec
Out - in	1041mm	762mm/sec
Traverse		

WRIST MOTIONS, STROKES AND SPEEDS:

Revolution	360^{0}	110^{0}/sec
Swing (right-left)		
Bend (up-down)	220^{0}	110^{0}/sec

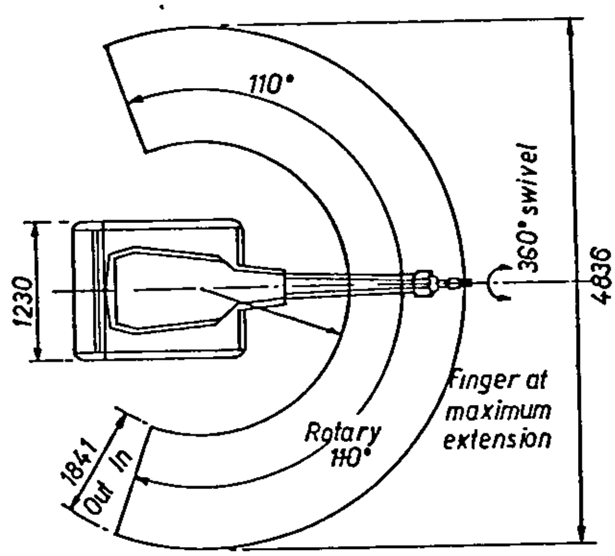
CONTROL FUNCTION:

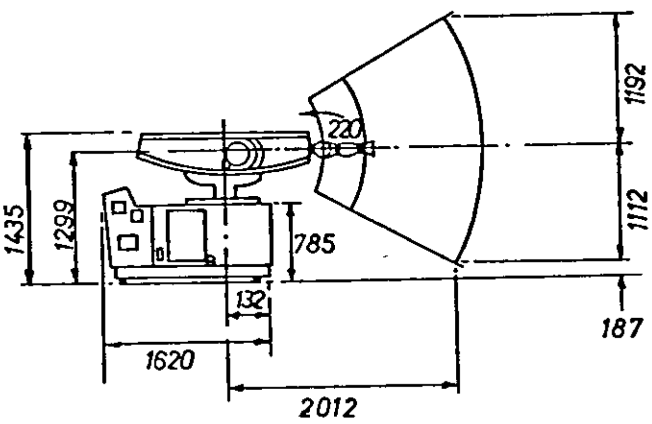
Motion control	CP by PTP teaching
Memory systems	Wire memory plus magnetic tape storage
Memory capacity	512 steps

POSITIONING ACCURACY: ±1.0mm

CONDITIONS FOR INSTALLATION:

Dimensions (length x width x height)	1260 x 1230 x 1435
Weight	1500kg
Power requirements	440V
Temperature	50^{0}C
Source of driving power	Hydraulic





SPECIFICATION: UNIMATE APPRENTICE - UNIMATE

ARM MOTIONS, STROKES AND SPEEDS:

Right - left	890mm	500mm/sec
Up - down	90°	
Out - in	50°	
Traverse		

WRIST MOTIONS, STROKES AND SPEEDS:

Revolution	180°
Swing (right-left)	
Bend (up-down)	175°

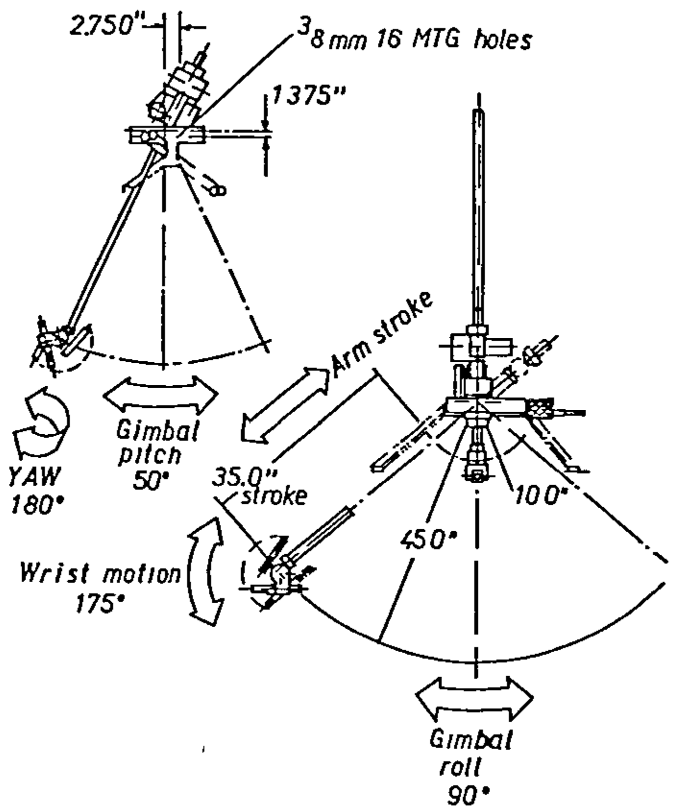
CONTROL FUNCTION:

Motion control	CP
Memory systems	
Memory capacity	

POSITIONING ACCURACY: ±1.0mm

CONDITIONS FOR INSTALLATION:

Dimensions (length x width x height)	880 x 500 x 2300mm
Weight	34kg + controller at 80kg
Power requirements	1.0kVA
Temperature	
Source of driving power	Electric stepping motor



SPECIFICATION: K15 VOLKSWAGEN

ARM MOTIONS, STROKES AND SPEEDS:

Right - left	320°	80°/sec
Up - down	65°	30°/sec
Out - in	100°	50°/sec
Traverse		

WRIST MOTIONS, STROKES AND SPEEDS:

Revolution	350°	120°/sec
Swing (right-left)		
Bend (up-down)	270°	120°/sec

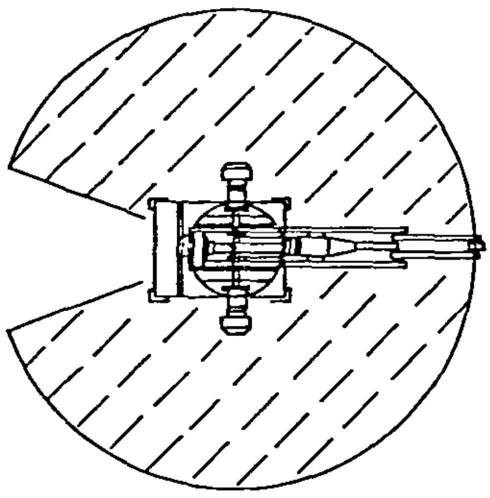
CONTROL FUNCTION:

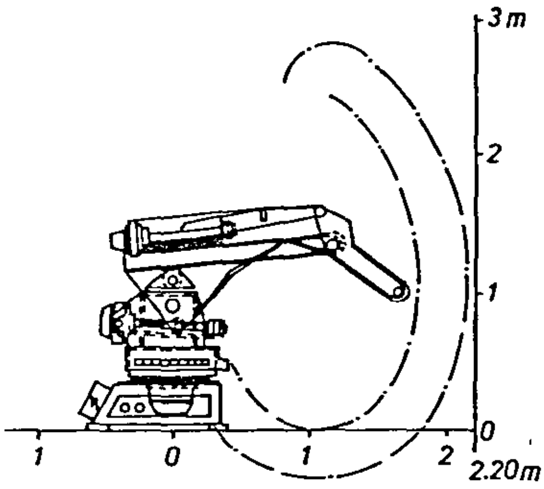
Motion control	CP by PTP teaching
Memory systems	
Memory capacity	100 points plus magnetic or punched tape storage

POSITIONING ACCURACY: ±1.0mm

CONDITIONS FOR INSTALLATION:

Dimensions (length x width x height)	1000 x 1000 x 1200mm
Weight	760kg
Power requirements	80kW
Temperature	50°C
Source of driving power	DC servo motors





SPECIFICATION: MOTORMAN — LINCMAN YASKAWA

ARM MOTIONS, STROKES AND SPEEDS:

Right - left	240°	90°/sec
Up - down	±40°	800mm/sec
Out - in	+20°-40°	1100mm/sec
Traverse		

WRIST MOTIONS, STROKES AND SPEEDS:

Revolution	360°	150°/sec
Swing (right-left)		
Bend (up-down)	180°	100°/sec

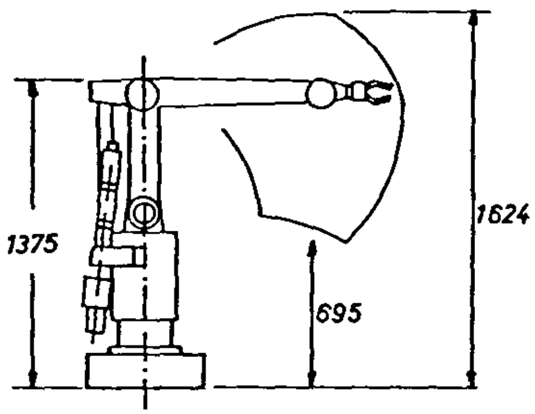
CONTROL FUNCTION:

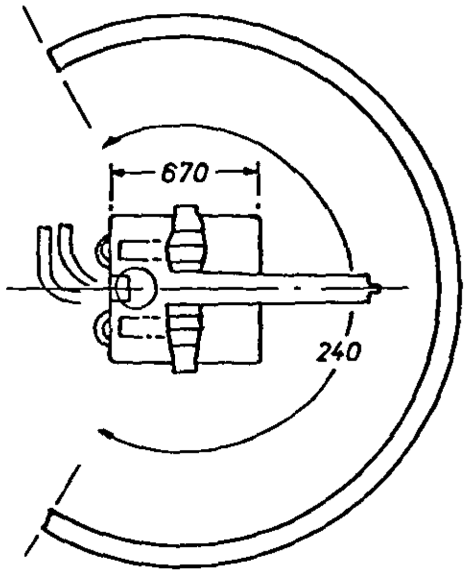
Motion control	PTP
Memory systems	Microcomputer plus magnetic tape storage
Memory capacity	250 (basic)

POSITIONING ACCURACY: ±0.3mm

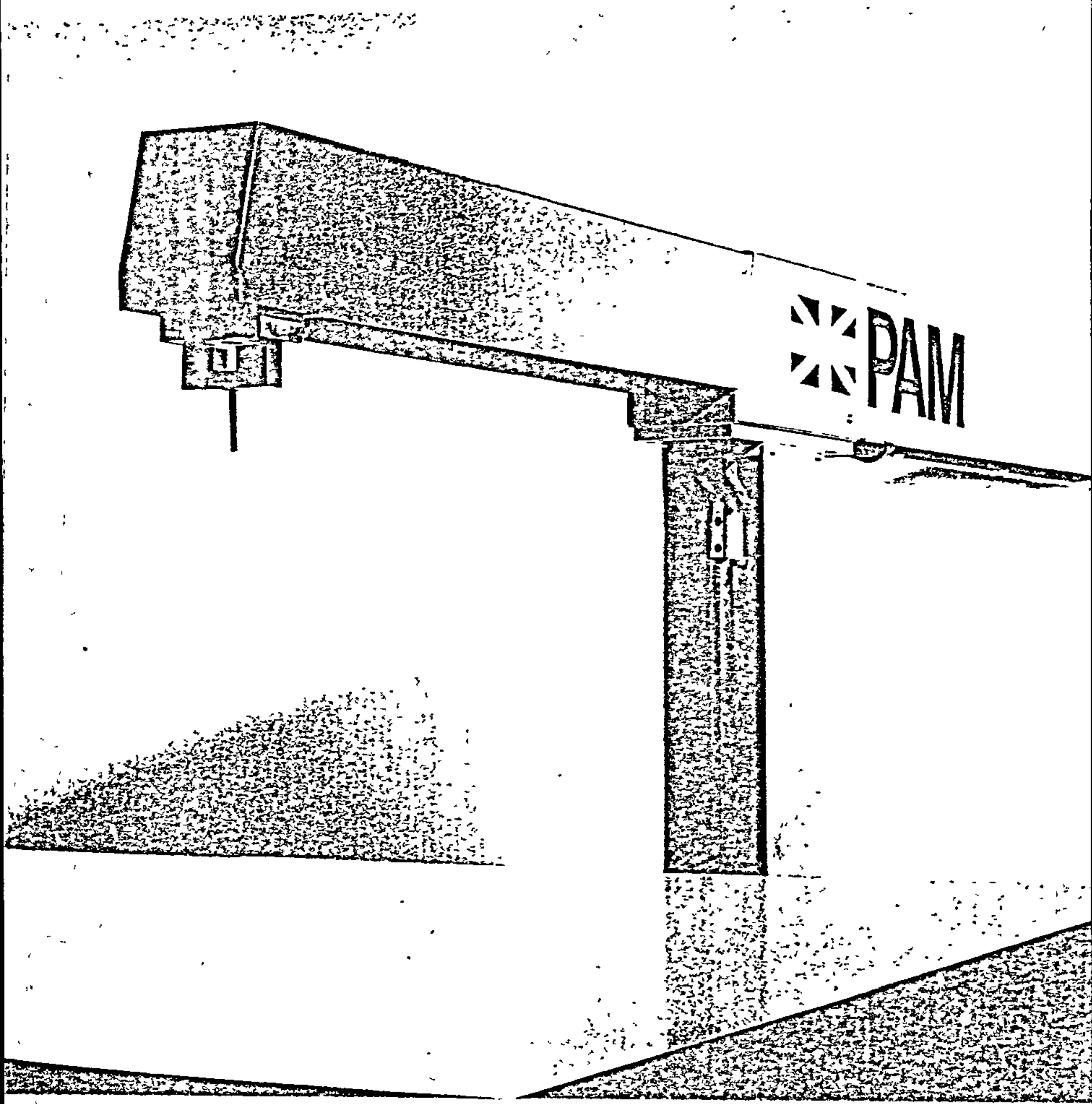
CONDITIONS FOR INSTALLATION:

Dimensions (length x width x height)	700 x 650 x 1600
Weight	350kg
Power requirements	200VAC
Temperature	45°C
Source of driving power	DC motor drives





PAM ROBOT



Technical specification

laximum loadıng on manipulator arm:
Kg (11lb).

laximum stroke/travel of X & Y axes:
10mm (24ın).

laximum stroke/travel of Z axis:
05mm (12ın). (610mm is avaılable as an
lternatıve optıon)

ositional accuracy (resolution) on all axes:
± 0 052mm (± 0 0025ın) (repeatabılıty·
% of resolutıon).

tored positions:
aximum of 1000 stored posıtıons are
ailable – only 200 are normally of practıcal
se, but this can be increased by addıng
rther memory capacıty

ositional speed of manipulator arm:
inimum 0.3 metre/sec (1 0 fps).

oint-to-point transfer time:
seconds maxımum.

hree-axis transfer speed:
inimum 0.52 metre/sec (1.7 fps).

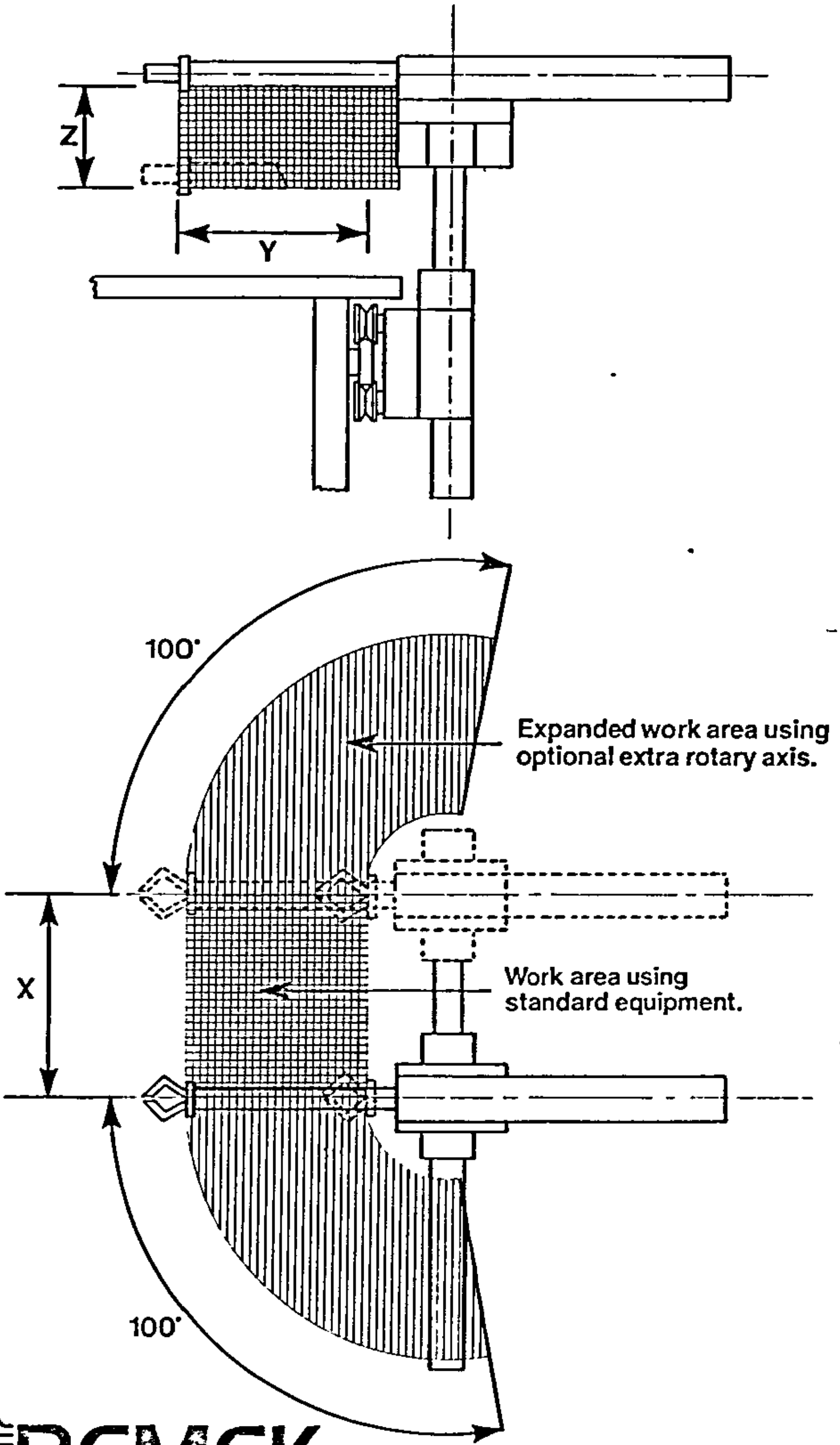
aximum Z-axis downward force:
1N (70 lbf).

nvironmental requirements:
rm work-table mountıng

ectrical supply:
20/240 VAC Sıngle Phase ınput and
tput signals 24V AC and DC (other
quırements can be met).

r supply:
bar (73 psi). Approx 56 lıtre/min
ft³/mın).

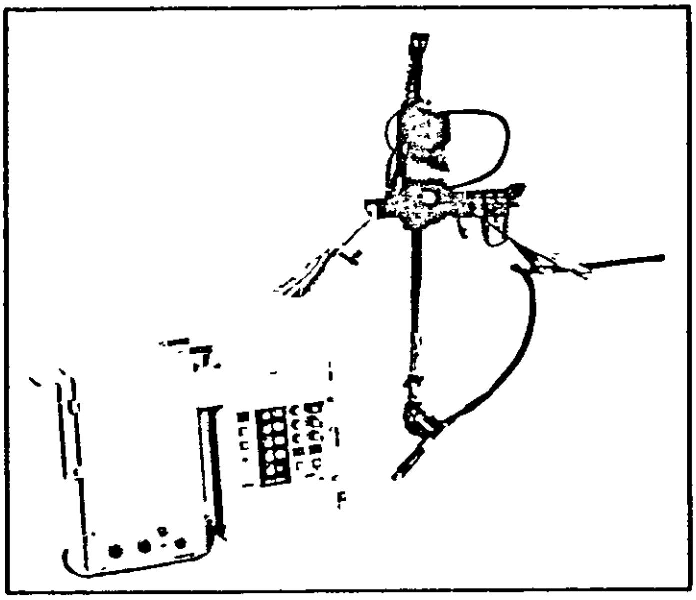
mensions:
axis – 1168mm (3ft 10ın) collapsed,
07mm (6ft 7ın) extended.
axıs – 1041mm (3ft 5ın) telescoped,
53mm (5ft 9ın) extended.
axıs – 978mm (3ft 2½ın) collapsed,
83mm (4ft 2½ın) extended.

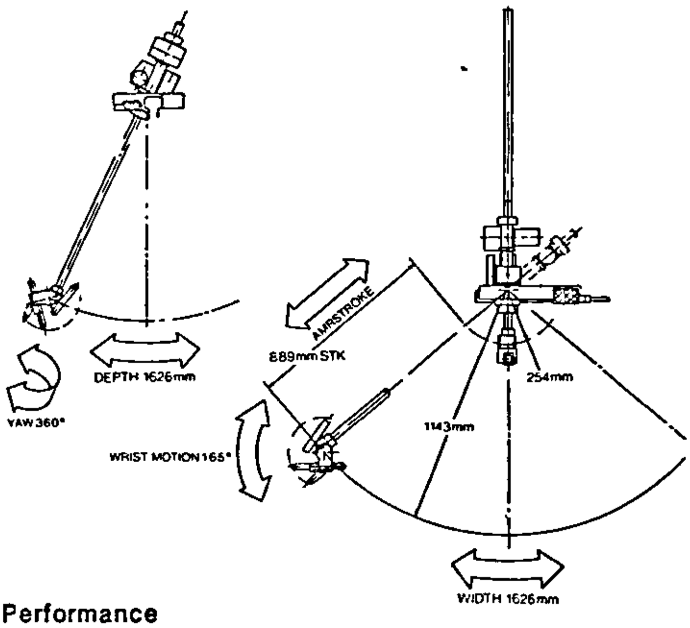


REMEK
Remek Automatıon Lımıted
Barton Road, Water Eaton Industrıal Estate, Bletchley, Mılton Keynes MK2 3H
Telephone Mılton Keynes (0908) 71828

The Unimate® Apprentice™

The Apprentice© is an arc welding robot that ensures top quality work and consistency even under hazardous or monotonous conditions. The portability of the robot makes it particularly useful when the workpiece to be welded can not be moved.





Performance

WELDING SPEED	10 to 200 mm per minute	WEAVING AMPLITUDE	2 to 20 mm peak to peak ± 1 mm max
NO OF WELDING SPEEDS	4	ACCURACY	deviation between taught welding path and the repeated path in automatic welding mode
NO OF PRESELECT WELDING CURRENTS	4		
TRANSFER SPEED	500 mm per second	ARM WEIGHT	34 Kg
WEAVE CHANNELS	2		
WEAVING FREQUENCY	0 1 per sec to 1 per sec	CABINET WEIGHT	79 Kg

Working Envelope

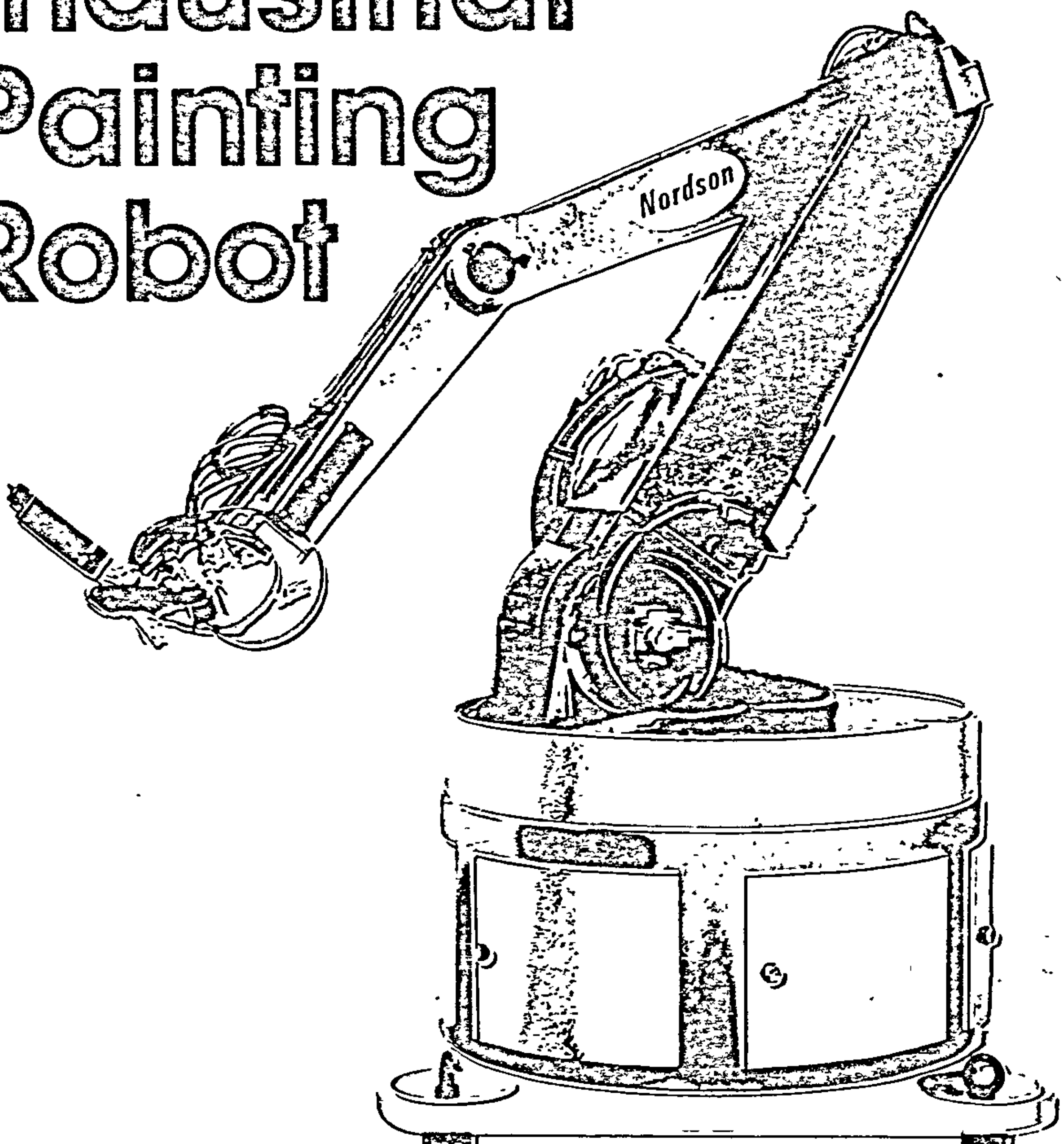
ARM STROKE	890 mm
GIMBAL, ROLL	90 degrees
GIMBAL, PITCH	50 degrees
YAW	180 degrees
WRIST MOTION	175 degrees

Cable Length: 10 mm

Power Requirement:
240/480 V, + 10%-15%
Single phase 50 Hz, 1 KVA
(other options available)

Nordson®

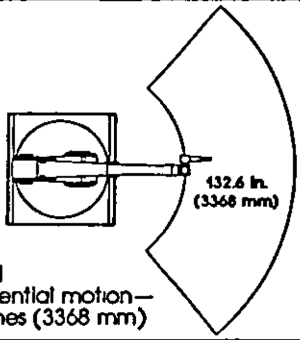
Industrial Painting Robot



The Nordson Robot Provides Six Axis Movemen for those "Hard-to-Reach" Areas

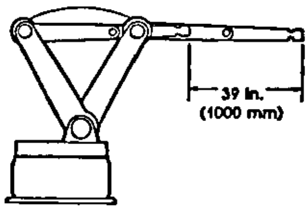
(Work Envelope dimensions can be increased depending on spray gun used.)

Manipulator Arm/Swivel Base Assembly



Horizontal circumferential motion— 132 6 inches (3368 mm)

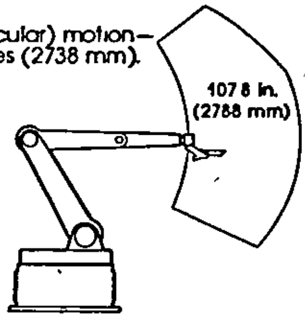
Manipulator Arm



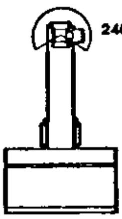
Forward and reverse motion— 39 37 inches (1000 mm)

Manipulator Arm

Vertical (perpendicular) motion— 107 8 inches (2738 mm).

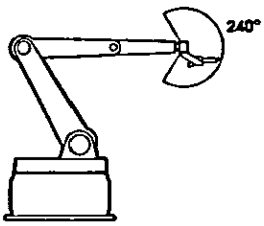


Manipulator Wrist



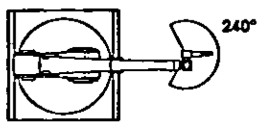
Rotational motion— 240 degrees.

Manipulator Wrist



Vertical (perpendicular) motion— 240 degrees.

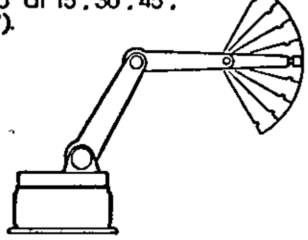
Manipulator Wrist



Horizontal, circumferential motion— 240 degrees.

Manipulator Arm

Manual pivotal adjustment— 15 degree increments (Up 60° at 15°, 30°, 45°, and 60°. Down 60° at 15°, 30°, 45°, and 60°).

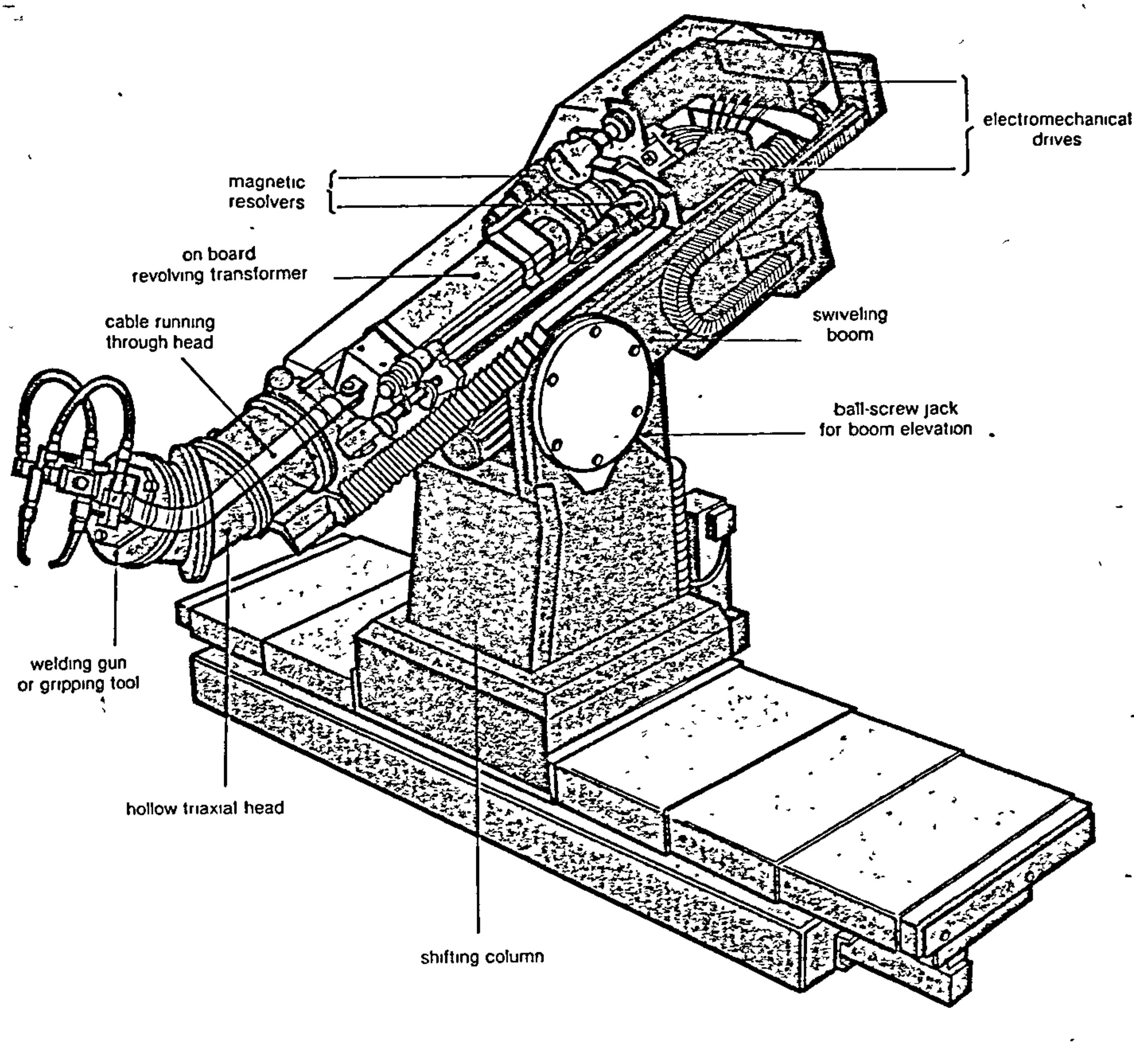


The six axis movement, illustrated above, is an important feature of the Nordson Robot. It means that the manipulator arm has complete flexibility in the most diversified applications. The Nordson Robot can duplicate and maintain the movements of the most skilled painter.

SPECIFICATIONS

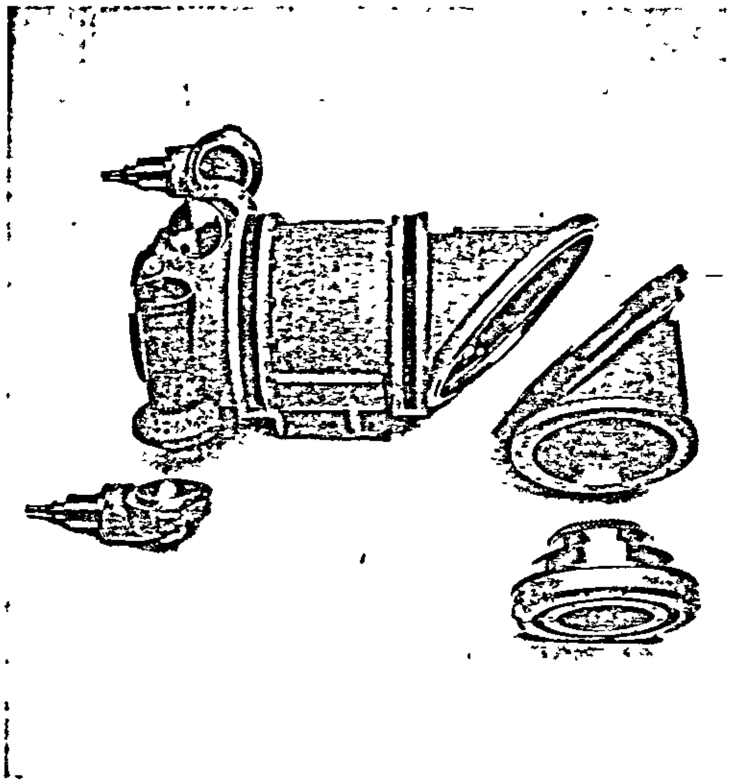
	U.S.A.	Metric
Manipulator		
Dimensions		
(Base)	38 6 in. dia.	980 mm d
Weight	1300 lbs.	590 kg
Speed of movement	82 in./sec.	2 m/sec.
Electronic Control		
Height	76 5 in.	1943 mm
Width	22 in.	559 mm
Depth	33 in.	838 mm
Weight	350 lbs.	159 kg
Hydraulic Power Pack		
Height	78 in.	1981 mm
Width	28 in.	711 mm
Depth	43 in.	1092 mm
Weight	1380 lbs.	625 kg
Electrical	460V, 60 Hz, 3 Phase	
Power	10 HP, 6 kW	

FATA-BISIACH &CARRU
JOLLY 80 ROBOT



CHARACTERISTICS

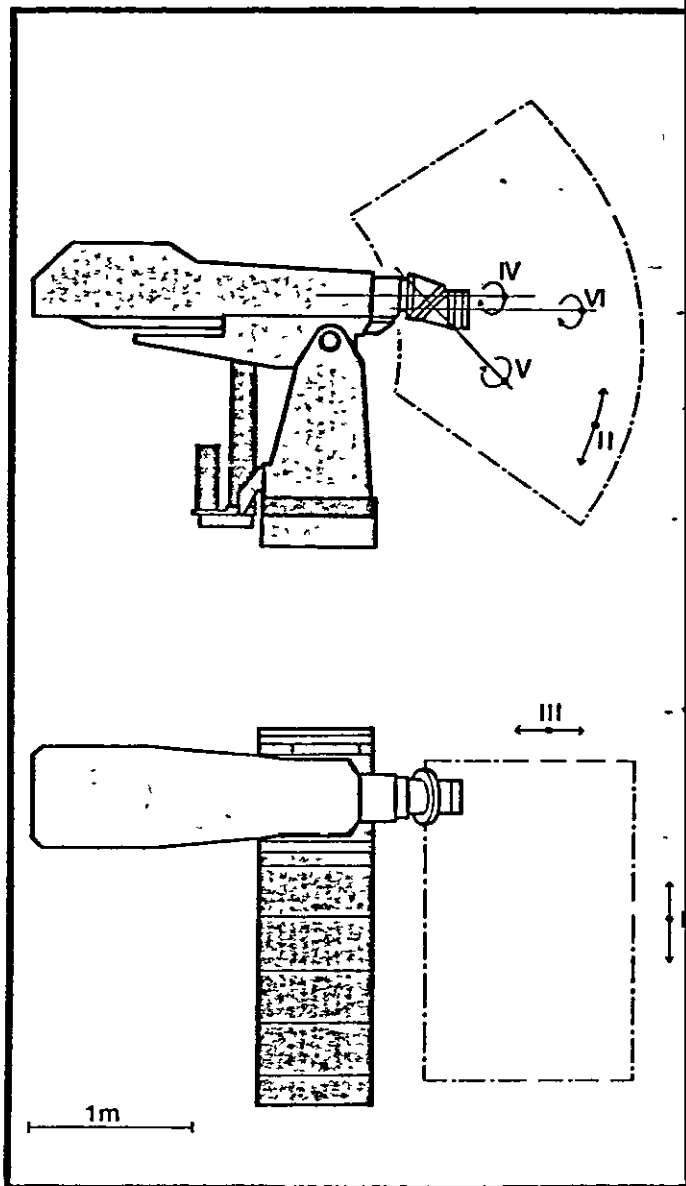
Electromechanical drive on all axes, with three 6-N.m and three 10-N.m. d.c. motors
Ball-screw drives on main axes
Air braking of boom descent
Semi-absolute position sensing by magnetic resolvers
Speed sensing by tachometers
Welding control with two or four programmes
Kinematically integrated head with three degrees of freedom
Safety stop through shock sensor on the gunholder
On board revolving transformer with 70 kVA rating
Coaxial supply cable between transformer and gun running through head; cross-section 300 mm², length 800 mm.
Payload on the head 70 kg (100 kg at reduced speed)
Overall weight 1000 kg
Repeat accuracy 0.3 mm



Ranges and speeds:

Axis	Range	Max. Spee
I	2000 mm	0.5 m/se
II	70°	25°/se
III	1100 mm	0.4 m/se
IV	400°	60°/se
V	400°	60°/se
VI	400°	100°/se

Linar interpolation between points
Handheld keypad for field teaching
Programme with 512 steps extendible to 2048 steps
ON/OFF signals available for driving external devices
Alphanumeric keyboard and display.



SPECIFICATIONS

Robot

GKN Linc-Man Model			L10
Number of Axes			5
Axis	S	Base rotation 240°	90°/sec
	L	Lower arm ± 40°	800mm/sec
	U	Upper arm + 20° - 40°	1100mm/sec
	T	Wrist Turn 360°	150°/sec
	B	Wrist Bend 180°	100°/sec
		A sixth external axis is available as an option.	—
Accuracy (wrist centre)			±0 2mm
Load capacity			10kg
Weight			405kg

Welding Set

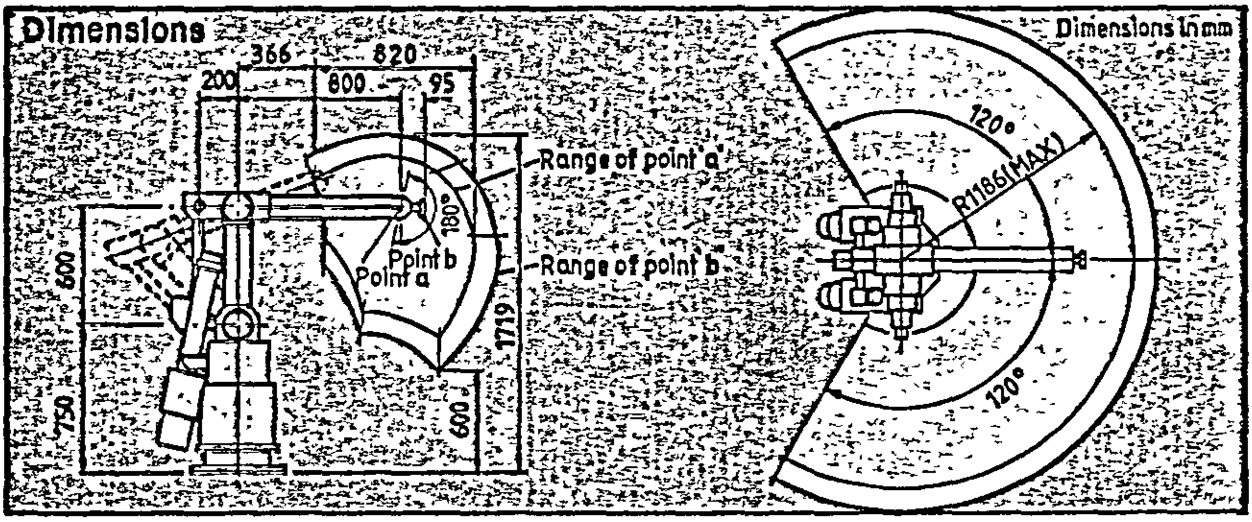
DYNA-AUTO CPM SERIES			
Power Source	Current (A)	Voltage (V)	Duty Cycle
CPM300M	300	15 - 32	50%
CPM350M	350	15 - 36	50%
CPM500M	500	15 - 42	60%
Wire Feed Unit		Type CM231	
Wire Speed		1.5 - 15m/min	
Wire size (solid)		0 6 - 1 6mm	
Wire size (f c w)		1 6 - 2 0mm	
Weld Gun	Standard type is CWG300 (300A at 100% duty cycle, 400A at 60% duty cycle).		
Options	Seam following, air-blast nozzle cleaner, water cooling, fume extraction and weld-check systems are available as optional extras.		

Controller

Dimensions		6000 RG/10
Size	Height	1600mm
	Width	650mm
	Depth	700mm
Weight		350kg
Ambient temperature		0 - 45°C
Power required		5kVA
Enclosure	Totally enclosed, dust-proof	

Control Functions

Teaching Method	Direct by control box including step forward and step back function
Path Control Method	PTP (point to point) with motor interpolation (Also linear interpolation — see below)
Linear Interpolation.	Weld gun or tool point traces true straight line with 5 axis movement.
Position Sensor.	Incremental rotary encoder.
Memory Details:	Type — I C memory for sequence control and system programme Capacity — 4 programmes — 99 jobs — 1000 points — 600 instructions
Interface Section	— 16 input channels, — 16 output channels, — 4 analogue channels for welder control, plus welder on/off control functions
Speed Selection	8 teaching speeds Play back speed set by 'TRT' function or by direct entry of absolute torch/tool point speed
Editing	Taught data may be altered as follows. (a) Single point add, erase, shift. (b) Instruction insert, delete (c) Speed of operation.
Display	Used for taught data review and on-line diagnostics Current status is shown during playback.
Time Delay	0 - 25 5 seconds in 0 1 second increments.
Branches and Sub-routines	Control inputs and internal counters can be used to call alternative jobs, part-jobs, sub-routines and other control functions
Diagnostic Functions.	Error and alarm codes indicate incorrect data entry/operation and results of self-diagnostic checks
Tape Recording	A built-in tape interface permits off-line data storage



GKN LINCOLN ELECTRIC LIMITED
Black Fan Road, Welwyn Garden City, Herts. AL7 1QA, England
Telephone: Welwyn Garden 24581. Telex: 268412

APPENDIX THREE

MOBILE ROBOTS

It is neither necessary nor desirable to create mobile robots in the image of man. Mobile robots need not be so flexible overall but could have senses that humans do not possess, such as infra-red vision, a much greater depth of vision field and immunity to extremes of temperatures.

In most industrial applications static robots serving manufacturing machines of different types and being connected via conveyors would be better than mobile robots with their attendant limitations. The most important of these are the need for some type of self-adaptive steering mechanism under control of image recognition units, sonar or radar, and a reliable low-loss tractor mechanism. A mobile robot by its very nature, must contain its own power source and must therefore be capable of checking its own "energy state" at intervals and then guiding itself to some central location or 'plugging in' to the nearest power source should its energy capacity fall below some pre-programmed level. For example before beginning a particular task it must compute if it has enough motive power left in its batteries to complete the task or whether it has to be recharged first. However, are mobile robots really necessary? Given its limitations there could be important roles for mobile robots in areas such as plant security, firefighting, warehousing and the monitoring of hazardous environments.

The domestic robot is many years away and although single task machines could be available for some domestic duties within a few years, their high cost would make them prohibitive.

The near disaster at the Three Mile Island nuclear power plant in early 1979 has pointed out the need for mobile robots which could enter a radiation-contaminated area, observe the damage through optical sensors, monitor the radiation level and have the manual dexterity to manipulate control valves, or to be able to remove wreckage which is posing a melt down threat. Nuclear radiation is not the only hazardous environment where emergencies take place. All too frequently we hear of mine disasters where deadly gases prevent rescuers from entering the area where survivors may be trapped. Fire earthquakes and tornados also impose obstacles to human efforts towards rescue.

The technology is now available to develop a disaster control robot which would venture into high radiation areas, into intense fires or, in times of natural disaster, could go into the area and not only observe but overcome debris and the dangers of downed high voltage wires to safely effect rescues.

An observing mobile robot(15) (operated by a battery) could be equipped with "eyes" in the form of a television camera, (supplemented by a powerful lighting system), "ears" in the form of directional microphones, and special senses tuned to atomic radiation levels and the temperature of the robot's position. The robot would have an on-board microcomputer to control its movement, eyes, ears and senses. There must be a human element to analyze the visual, audio and sensor data transmitted to the control area. Transmission could be achieved with a fibre optic tether cable which employs light instead of an electric current to transmit video, audio and digital data so to overcome the problems of radioaction and underwater operations. A damage control robot (DCR) would have "hands" which could, for example, operate valves and could be operated pneumatically, hydraulically, magnetic, gaseous or electrical.

Modern day robots range from the tiny microcomputer-controlled "Turtle" to the giant mechanical workhorses of industry.

The Turtle which is manufactured by Terrapin Inc is capable of guided movement, forward or reverse, at a speed of approximately 20 feet per minute. Its range is limited by the length of umbilical cord which connects the Turtle to the microcomputer.

This type of robot could also be utilised in the exploration of Space.

APPENDIX FOUR

VISION SYSTEMS

MACHINE INTELLIGENCE CORPORATION

VS-100 MACHINE VISION SYSTEM

A commercial development of the SRI 'eye'

Binary threshold picture at variable resolution

Accepts inputs from a variety of cameras

Operator interacts with light pen on text/graphic display

AUTOMATIX

AUTOVISION I

New company with large financial backing

Similar to SRI eye

Not fully developed

AUTOVISION II will be grey scale based

Programs written in PASCAL

Accepts different cameras

BROWN BOVERI and CIE

OMS (Optical Measurement System)

Systems sold working with ASEA, PUMA, VW, and KUKA robots

Hardware Orientated

Fast recognition (200ms)

IITB

SAM (Sensor System for Automation and Measurement)

Now being sold by BOSCH

Hardware orientated IR Strobe and Vidicon camera

Fast recognition (150ms)

AUTOPLACE

OPTOSENSE

Very simple grey level bit matching techniques but useful and easy to apply

Cannot check for orientation

Not fully developed

Hardware overkill

Microprocessor Scorecard®

Processor	Classification	Technology	Parts Family: Clock Driver	Parts Family: I/O Interface	Parts Family: UART/USRT	Parts Family: RAM	Parts Family: ROM/PROM	Parts Family: Interface	Features: Interrupts	Features: One-chip CPU	Features: Microprogrammed	Features: Accessible Stack	Features: DMA Ability	Features: BCD Arithmetic	Features: <0, >70° C. Avail.	Word Size (Data/Instruction)	Address Capacity (Program Words)
AMD 2900	4-bit Slice	TTL		●		●	●	●			●				●	4N/9	
DATA GENERAL mN601	16-bit CPU	NMOS		●		●		●					●			16/16	32K
ELECTRONIC ARRAYS 9002	8-bit CPU																
ESSEX SX200	4-bit CPU	PMOS	●							●				●		4/8	1K
FAIRCHILD F8	8-bit CPU	NMOS	●	●	●	●	●	●	●	●		●	●	●		8/8	64K
FAIRCHILD MACROLOGIC	4-bit Slice	TTL				●	●	●			●					4N/6	
GENERAL INSTRUMENTS CP-1600	16-bit CPU	NMOS		●	●	●	●		●	●	●	●	●			8/8	64K
INTEL 3000	2-bit Slice	TTL				●	●	●	●		●		●			2N/18+	512
INTEL 4004	4-bit CPU	PMOS	●	●		●	●	●		●				●	●	4/8	4K
INTEL 4040	4-bit CPU	PMOS	●	●		●	●	●	●	●				●		4/8	8K
INTEL 8008 1	8-bit CPU	PMOS	●	●	●	●	●	●	●	●					●	8/8	16K
INTEL 8080A 1	8-bit CPU	NMOS	●	●	●	●	●	●	●	●		●	●	●	●	8/8	64K
INTERSIL 6100	12-bit CPU	CMOS	●			●			●	●			●		●	12/12	4K
MONOLITHIC MEMORIES 6701	4-bit Slice	TTL				●	●				●				●	4/17	
MOS TECHNOLOGY 6502A	8-bit CPU	NMOS	●	●		●	●	●	●	●		●		●		8/8	64K
MOSTEK 5065	8-bit CPU	PMOS				●	●		●	●		●	●			8/8	32K
MOTOROLA 6800	8-bit CPU	NMOS		●	●	●	●		●	●		●	●	●		8/8	64K
MOTOROLA 10800	4-bit Slice	ECL		●		●	●	●			●			●		4N/16	
NATIONAL SC/MP	8-bit CPU		●	●		●	●		●	●			●	●		8/8	64K
NATIONAL GPC/P	4-bit Slice	PMOS									●	●	●			4N/23	100
NATIONAL IMP-4	4-bit CPU	PMOS				●	●	●	●		●			●		4/4	4096
NATIONAL IMP-8	8-bit CPU	PMOS				●	●		●		●	●	●			8/8	64K
NATIONAL IMP-16	16-bit CPU	PMOS				●	●		●		●	●	●			16/16	64K
NATIONAL PACE	16-bit CPU	PMOS	●			●	●	●	●	●	●	●	●	●		16/16	64K
NATIONAL TCS	4-bit CPU	PMOS	●	●		●	●			●				●		4/8	8K
NEC μCOM-8	8-bit CPU	NMOS		●	●	●	●	●	●	●		●	●	●		8/8	64K
RCA COSMAC	8-bit CPU	CMOS	●			●			●	●		●	●		●	8/8	64K
ROCKWELL PPS-4	4-bit CPU	PMOS	●	●	●	●	●			●		●		●		4/8	8K
ROCKWELL PPS-4/1	4-bit CPU																
ROCKWELL PPS-8	8-bit CPU	PMOS	●	●	●	●	●		●	●		●	●	●		8/8	16K
SIGNETICS 2650	8-bit CPU	NMOS				●	●		●	●			●	●		8/8	32K
TEXAS INSTRUMENTS SBP0400	4-bit Slice	IIL				●	●				●				●	4N/9+	
TEXAS INSTRUMENTS TMS1000	4-bit CPU	PMOS								●	●			●		4/8	2K
TEXAS INSTRUMENTS TMS9900	16-bit CPU	NMOS		●	●	●	●	●	●	●		●	●			16/16	32K
TOSHIBA TLCS 12	12-bit CPU	NMOS		●		●	●		●	●	●					12/12	4K
TRANSITRON 1601	4-bit Slice	TTL							●		●	●	●			16/16	32K
WESTERN DIGITAL 1600	16-bit CPU	NMOS			●	●	●		●		●	●	●			16/16	64K
ZILOG Z80	8-bit CPU	NMOS		●	●				●		●	●	●	●		8/8	64K

Processor	Clock (kHz/Phases)	Register Add Time (μ sec per Data Word)	Number of CPU Registers: ALU	Number of CPU Registers: XR	Number of CPU Registers: GP	Return Stack Size (Nr x Bits)	Voltages Required	Power Dissipation	Package Sizes (DIP Pins)	Status: First Samples	Status: First Deliveries	Remarks
AMD 2900	8300/1	12	1		17	5	.9		16,20,24 40	3Q75	4Q75	Second Source Motorola
DATA GENERAL mN601		2 4	1		4	(RAM)			40	3Q76	4Q76	
ELECTRONIC ARRAYS 9002												
ESSEX SX200	500/0	16	1		64	1x10	10 ~ 18	15	28			Clock,Ram ROM on Chip
FAIRCHILD F8	2000/0	2	65			(RAM)	+5,+12	6	40	1Q75	2Q75	Second Source Mostek
FAIRCHILD MACROLOGIC	13333/1	.075	8				5	5	14,16,24	3Q75	1Q76	
GENERAL INSTRUMENTS CP-1600	5000/2	2.4	6			(RAM)	−3 +5,+12	6	40	3Q75	3Q75	
INTEL 3000	6061/1	165	2		10	(None)	5	.8	28,40	3Q74	3Q74	Second Source Signetics
INTEL 4004	740/2	10 8	1		16	3x12	15 or (−10,+5)	1.0	16	2Q71	4Q71	Second Source National
INTEL 4040	740/2	10.8	1		24	7x12	15 or (−10,+5)	1.0	16,24	4Q74	4Q74	
INTEL 8008 1	800/2	12 5	1		6	7x14	−9,+5	1.0	18	4Q71	1Q72	
INTEL 8080A 1	3000/2	1 33	1		6	(RAM)	−5 +5,+12	1.0	40	4Q73	2Q74	Second Source AMD,T I
INTERSIL 6100	4000/0	5	1		1	Modifies Program	5	01	28,40	2Q75	3Q75	PDP 8 Code
MONOLITHIC MEMORIES 6701	6666/1	.2	3		16	(None)	5	1 0	40	1Q74	2Q74	
MOS TECHNOLOGY 6502A	2000/0	1	1	2		(RAM)	5	25	40	3Q75	4Q75	Second Source Synertek
MOSTEK 5065	1400/3	10	3			(RAM)	−12,−5,+5	7	40	1Q74	3Q74	
MOTOROLA 6800	1000/2	2	2	1		(RAM)	5	.25	24,40	2Q74	4Q74	Second Source AMI
MOTOROLA 10800		055	1				−5 2,−2	1 3	48	3Q76	3Q76	
NATIONAL SC/MP	1000/0	38	1	3	1	(None)	12 or (−7,+5)	.9	40	4Q75	1Q76	
NATIONAL GPC/P	715/4	1 4	8			16x4N	−12,+5	7	22,24	1Q73	3Q73	1 < N < 6
NATIONAL IMP-4	500/4	12	4			7x12	−12,+5	1.0	24,40	3Q74	4Q74	
NATIONAL IMP-8	715/4	4 6	3	1		16x8	−12,+5	1 0	22,24	4Q73	1Q74	
NATIONAL IMP-16	715/4	4 6	2	2		16x16	−12 +5	1.2	22,24	1Q73	3Q73	
NATIONAL PACE	2000/2	8	2	2		10x16	−12,+5,+8	7	40	1Q75	2Q75	
NATIONAL TCS	400/1	10	1		161	2x11						
NEC μCOM-8	2000/2	2	1		6	(RAM)	−5,+5,+12	.8	42	4Q74	2Q75	
RCA COSMAC			1		p	rx16	5 ~ 12	.01	40	1Q76	2Q76	(p + 2r) ≤ 30
ROCKWELL PPS-4	200/2	5	1		1	2x12	17	225	42	1Q72	3Q72	
ROCKWELL PPS-4/1												
ROCKWELL PPS-8	256/2	4	1	1	2	(RAM)	17	3	42	4Q74	1Q75	
SIGNETICS 2650	1200/1	4.8	7			8x15	5	5	40	1Q75	2Q75	
TEXAS INSTRUMENTS SBP0400	1000/1	1	10				> 85	13	40	4Q74	3Q75	
TEXAS INSTRUMENTS TMS1000	500/1	12	1		1	1x6	15	1	28,40	1Q75	2Q75	Clock, RAM,ROM on Chip
TEXAS INSTRUMENTS TMS9900						(RAM)			64	1Q76	2Q76	
TOSHIBA TLCS 12	1000/3	13	4		6	(RAM)	−5,+5	.8	16,24,28 42	2Q74	3Q74	
TRANSITRON 1601		.4	u	v		(RAM)			40	2Q76	3Q76	(u + v) ≤ 8
WESTERN DIGITAL 1600	3300/4	.6	x	y		(RAM)	−5,+5,+12	1.2	40	3Q75	3Q75	(x + y) ≈ 16
ZILOG Z80	2500/1	1.6	1	2	13	(RAM)	5	1.0	40	1Q76	2Q76	8080A Compatible

APPENDIX SIX

STANDARD DAC & DAC AND SAMPLED DATA THEORY

The RTI-1241[56] is shipped from the factory with jumpers installed as required to produce the configuration shown in Table A6.1. The only tailoring required to get the board fully operational is the selection of a base address, which can be selected by installing a wire jumper across the relevant pins.

The relationship between analog voltage and digital value is given in table A6.2.

The RTI;1241 appears to the controlling microcomputer as a block of eight continuous memory locations in the microcomputer's address space. On the RTI-1241[57] board and 8 DAC's one of which is used for the control of this robot, again this is memory mapped.

All control and data transfer operations are accomplished by writing into, or reading from, one or another of the eight words for the RTI-1241 exactly as would be done with read/write memory. Each word has a pre-assigned function as in Table A6.3. In the tabulation below the functions of all the bits in each word of the memory map are described.

DAC 2 DATA(BASE + 0): Data written into this word is converted into an analog signal output by one of the analog output channels (DAC 2). The 12-bit DAC data is right-justified in the 16-bit microcomputer word; the four most significant bits of the computer are ignored, and can therefore have any value. This is a write-only address.

DAC 1 DATA(BASE + 2): This word functions in exactly the same way as DAC 2 DATA, but produces analog output on the DAC 1 output channel.

SETUP (BASE + 4): The three active bits in the SETUP word enable and disable control functions which may be used during data acquisition opertions.

EOC INT: 1-Enables End-of-Conversion Interrupts
0-Disables End-of-Conversion Interrupts

AUTO SCAN: 1-Causes Musc Address to be automatically

Table A6.1 Shipped Configuration of the RTI-1241 Board

Function	Factory Wiring
Analog Inputs	
Mux logic	Single Ended
Instrumentation Amplifier Inputs	Single Ended
Ground Sensing	On Board Analog Common
IA Gain	IV/V
ADC Input Range	±10V Biploar
ADC Output Code	Two's Complement
Analog Outputs	
DAC Input Code	Two's Complement
DAC Output Range	±10V Bipolar
Reference	Internal +10V
Interface	
Base Address	FFFØ (HEX)
Operating Mode	Polled ADC Status
Interrupt Line	Name Selected
System Reset	Both DAC's and Digital Output Drivers
Analog Common Digital Ground	Connected

Table A6.2

Relation of Analog Voltage to Digital Value

Analog Voltage	Hex Data
9.995	07FF
7.500	0600
5.000	0400
2.500	0200
1.250	0100
0.625	0080
0	0000
-0.625	FF80
-1.250	FF00
-2.500	FE00
-5.000	FC00
-7.500	FA00
-10.000	F800

MEMORY MAP

Data Word Format

Word Address	A_{12}	A_{13}	A_{14}	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	Word Name & Operation	
Addr.)	0	0	0	Ø ←			→ Ø	M	D_2	D_3	D_4	D_5	D_6	D_7	D_8	D_9	D_{10}	D_{11}	D	DAC 2 DATA	WRITE
+2)	0	0	1	Ø ←			→ Ø	M	D_2	D_3	D_4	D_5	D_6	D_7	D_8	D_9	D_{10}	D_{11}	D_L	DAC 1 DATA	WRITE
+4)	0	1	0	EOC INT	AUTO SCAN	EXT CC	Ø ←												→ Ø	SETUP	READ, WRITE
+6)	0	1	1	Ø ←												→	0	G_1	G_0	GAIN	READ, WRITE
+8)	1	0	0	Ø ←							→ Ø	A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0	MUX ADDRESS	READ, WRITE
+A)	1	0	1	Ø ←															→ Ø	CONV. COMM.	WRITE
+C)	1	1	0	EOC	U/R	Ø	Ø	OPTIONAL - SEE NOTE												STATUS	READ
+E)	1	1	1	S	S	S	S	M	B_2	B_3	B_4	B_5	B_6	B_7	B_8	B_9	B_{10}	B_{11}	L	ADC DATA	READ

1. The symbol Ø indicates a bit that is ignored during a write and has an arbitrary value when read.
2. The three active bits in the setup word enable or disable control functions on the RTI-1240/1241.
   EOC INT: 1 - Enables end-of-conversion interrupts.
   0 - Disables end-of conversion interrupts.
   AUTO SCAN: 1 - Causes MUX address to increment automatically as each conversion is performed. Incrementation takes place just after the sample-hold circuit holds the input value for the current conversion.
   0 - Disables the Auto SCAN feature.
   EXT CC: 1 - Enables external convert commands (from P2-18).
   0 - Disables external convert commands.
3. Gain codes (units with software-programmable gain only):

   00 - Gain = 1
   01 - Gain = 2
   10 - Gain = 4
   11 - Gain = 8
4. Convert command will occur on any write to Base +A. The data written is ignored.
5. Two bits in the status word are control functions:
   EOC: 1 - Indicates end of conversion (Data Ready).
   0 - Conversion not complete. When interrupts are used, EOC indicates the presence of an interrupt. Reading the status word clears EOC (if set) and the associated interrupt, if any.
   U/R: 1 - Indicates an underrange condition, that is:
   a) the signal just converted is small enough to use a higher gain, and
   b) a higher gain is available.
   0 - Indicates no further gain ranging can be done. (The U/R bit is present only on models with software-programmable gain.)

Option Status Word Formats (Jumper Option - See Chart)

EOC	U/R	Ø	Ø	0	0	G_1	G_0	A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0	Gain & MUX Setting in Status Word.
EOC	U/R	Ø	0	M	B_2	B_3	B_4	B_5	B_6	B_7	B_8	B_9	B_{10}	B_{11}	L	ADC Data In Status Word.

6. In the ADC Data Word, S indicates a sign fill bit equal to 0 for unipolar coding and MSB for 2's complement coding.
7. Bus reset clears the control bits in the setup word and the EOC bit in the Status Word.

Table A6.3 Memory Map of RTI-1241 Board

incremented as each conversion is performed in channel-scanning applications

0-Disables automatic scanning

EXT CC: 1-Enables external convert commands

0-Disables external convert commands

MUX ADDRESS(BASE + 8): The eight least significant bits of this word select the analog input channel during data acquisition operations. The MUX ADDRESS word is read/write, so that unrestricted use may be made of the full microcomputer instruction set for selecting and modifying channel addresses. The read function is also useful, particularly during Auto Scan operation, for determining the current input channel

STATUS (BASE + C): The STATUS word contains information about the conversion currently in progress or just completed. The two most significant bits have the following significance.

EOC: 1-Indicates End of Conversion (data ready)

0-Conversion not complete

When interrupts are used, EOC indicates the presence of an interrupt. Reading the STATUS word clears EOC (if set) and the associated interrupt (if any).

U/R 1-Indicates an underrange condition, that is;

a) The signal just converted is small enough to use a higher gain, <u>and</u>

b) A higher gain is available

The STATUS word is read-only. The EOC bit in this word is cleared by a system reset.

ADC DATA (BASE + E): The results of A to D conversions are available in this word. Data will be valid until a new conversion is begun. The 12-bit ADC output is right-justified in the 16 bit microcomputer word. The four highest-order bits (0-3) in the computer word are filled with zeros when unipolar or offset binary coding is used, and have a value equal to the MSB of the ADC data when two's complement code is selected. This sign fill is necessary for correct operation of the microcomputer's arithmetic instruction. The ACD DATA word is read-only.

GAIN (BASE + 6): The two least significant bits of this word set the gain of the instrumentation amplifier. The GAIN word is read/write.

CONV COMM (BASE + A): This triggers the A to D converter. The data sent by the write operation is not used, and therefore can have any value. A MOV instruction could be used, but SETO or CLR is preferable, since these instructions take considerably less time than a MOV. The CONV COMM word is write-only.

Analogue signal input-output

Analogue signals are continuously variable in amplitude. This contrasts with the digital representation of quantities inside a computer where a finite number of bits to a word means that only discrete values of amplitude can be represented. Hence, if analogue signals are to be passed to or from a digital computer, some kind of signal converter is required, as shown conceptually in figure A6.1 (55).

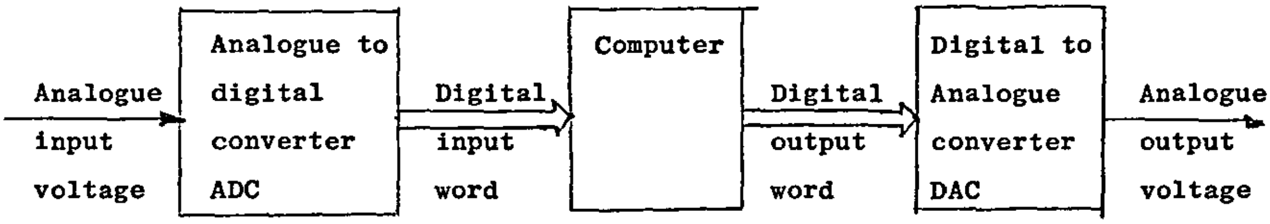


Figure A6.1

The terms data acquisition or data conversion are applied to the process on the analogue input side and data distribution on the output side. Digital processing of analogue signals by a computer offers several advantages including accuracy, flexibility, repeatability and the ability to perform complex operations. One consequence of using a computer is that data can be input at discrete points in time. Hence only sampled values of an anlogue signal can be taken, as shown in figure A6.2, and not the true signal itself. An obvious

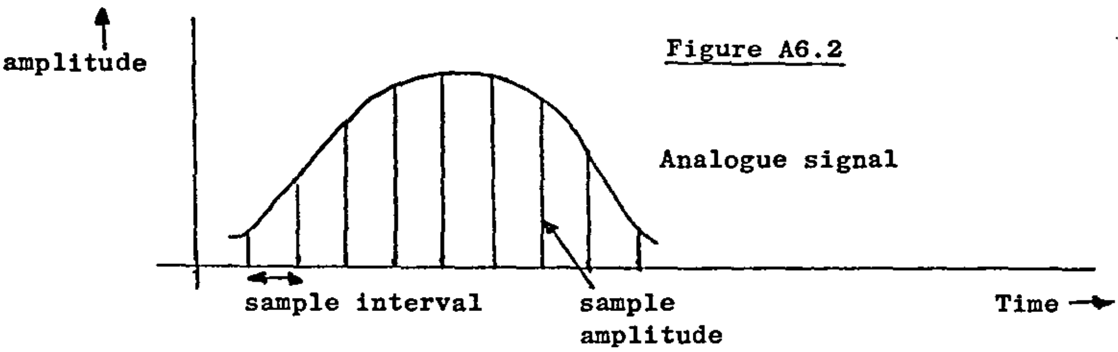
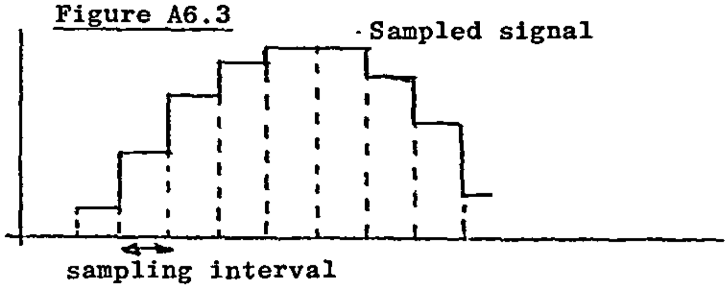


Figure A6.2

Figure A6.3



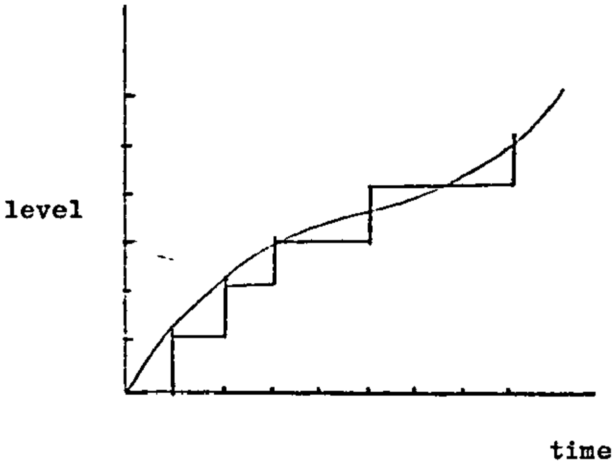
requirement is that the signal should not change significantly between samples otherwise information is lost, and a high enough sample rate must be used. The upper limit of sampling rate is of the order of 10^5 samples per second. However, it should be kept in mind that the higher the sampling rate, the less time there is available between samples for the computer to do useful processing of the data. The computer likewise, can only output data at discrete points in time which will be of the same form as in figure A6.3. The horizontal portions of this waveform occur while the next update of output amplitude is awaited. In many cases the 'staircase' effect is not noticeable because the analogue signal is slowly varying. A second consequence of the computer is that the data is represented by words having a finite number of bits. For example, a three-bit data word can assume any of 2^3 (that is 8) different codes: 000, 001, 010 .... 110, 111. Each code is made to correspond to a fixed level of analogue signal and consequently the signal may not be resolved into a sufficient number of discrete elements ot maintain the required accuracy.

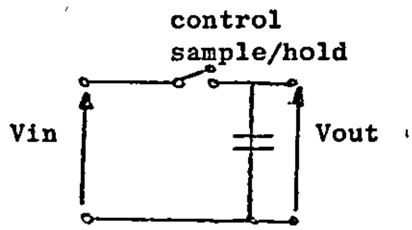
Apart from the sampling circuit in the analogue to digital converter, it requires a temporary storage or 'holding' device to maintain the value of the sampled input until the conversion process is complete. Analogue to digital converters use comparators to compare the input signal to the required digital output. The comparison takes a finite time, so the input voltage has to be maintained otherwise erroneous digital output can result. To do this, holding circuitry is required which is often achieved by using capacitors, however, leakage from these capacitors can cause problems by producing a slight droop in the voltage.

The conversion process involves the quantitising of the sampled input. The continuous input signal is converted into a set of discrete levels and any sample with a value between the discrete levels possible is converted to the level nearest to the actual value. This process is known as amplitude quantitisation and is illustrated in figure A6.4.

The difference between the analogue signal and the digital representation is dependent on the quatitising step as well as the sampling rate.

Figure A6.4 Amplitude Quantisation





A twelve bit analogue to digital converter, which gives a small quantitising step compared to an eight bit ADC, will give a closer representation of the analogue signal. The performance of an actual S/H differs from the ideal shown in figure A6.5. However, these differences

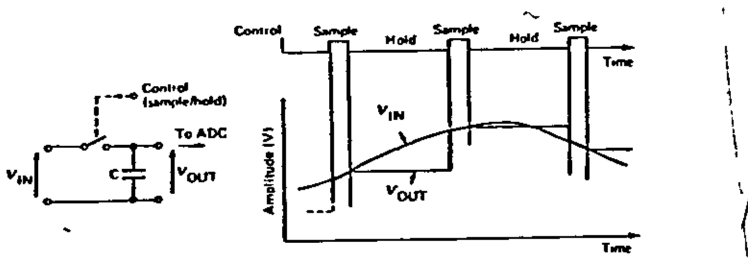


Figure A6.5
Ideal S/H

contribute to the overall accuracy of the system and can be significant. The most important effects are shown in figure A6.6 and are listed below.

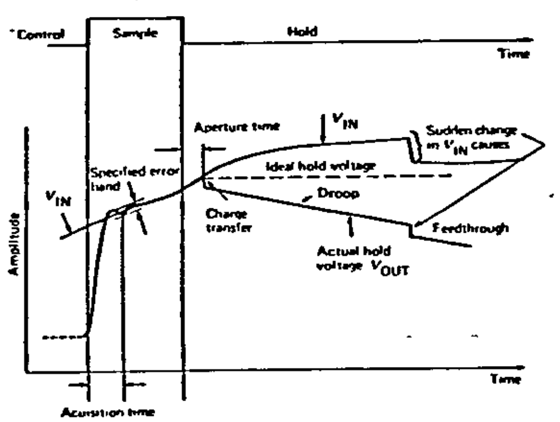


Figure A6.6
Actual S/H

i) Acquistion time (typically 1-10 s). This is the time taken from the start of the SAMPLE condition for the output voltage to equal the input voltage to within a specified band of error. A large component of acquisition time is due to the charging time of the capacitor. A low capacitor value should therefore be used.

ii) Aperture time (typically 0.01-0.2 s). This is the time between the HOLD instruction being given and the actual time the switch is opened.

iii) Aperture uncertainty or jitter (typically 2% to 10% of aperture time)

iv) Droop (typically 0.1-100mV/s). Ideally the output voltage of the S/H in the HOLD condition should stay constant. However, in practice Vout drifts from this value with time. This is called droop and is caused by discharge of the S/H capacitor due to (a) leakage current of the open switch, (b) self-discharge of the capacitor through its own dielectric. Droop is specified.as the maximum rate of change of output voltage and is undesirable since the reason for using the S/H is to obtain a constant sample amplitude. Droop can be reduced by using a large capacitor value. However, this conflicts with the requirements to minimize acquisition time and so an adequate compromise must be obtained.

v) Feedthrough and charge transfer. Feedthrough occurs during the HOLD condition when a change in input voltage causes a small unwanted change in output voltage even though the S/H switch is open. Charge transfer can take place when the switch is opened and a small charge is dumped in the storage capacitor which results in an offset in the output hold voltage. Both these effects contribute small errors and are only significant in high-accuracy analogue-input channels.

Consider the problem of inputting 64 analogue channels to a computer using the circuit which has been described. A separate chain ADC and S/H would be required for each channel, consequently the solution would be expensive. A multiplexer (MUX) allows a single S/H and ADC to be 'time-shared' over several analogue channels. The operation of the MUX can be understood from the system shown in figure A6.7.

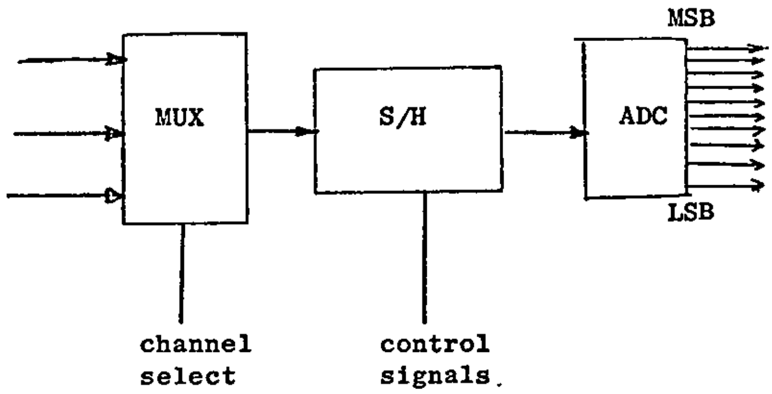


Figure .A6.7 Operation of a Multiplexer

The complete analogue to digital conversion process is illustrated by a flow chart in figure A6.8

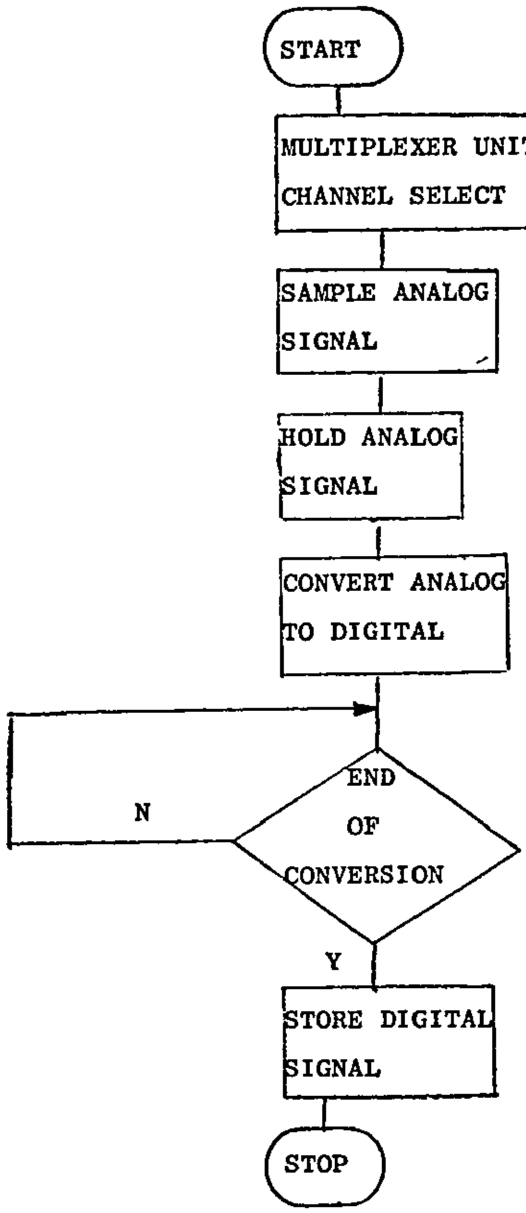


Figure A6.8 Analog to Digital Conversion Process

```
                    IDT  'TKY1'
     0000           RORG
                *
                *ONE AXIS ONE POSITION
                *
     0000       SPACE   BSS  32
          EFFE  ADCDAT EQU  >EFFE            MEM.FOR DIGITAL INPUT
          EFFC  STATUS EQU  >EFFC
          EFF0  DAC2   EQU  >EFF0             MEM. FOR DIGITAL OUTPUT
          EFF6  GAIN   EQU  >EFF6
          EFF8  MUXADR EQU  >EFF8            MEM. FOR SELECT CHANNEL ON ADC
          EFFA  CONV   EQU  >EFFA            MEM.TO START CONV
                *
     0020 02E0         LWPI SPACE
     0022 0000'
     0024 04E0         CLR  @MUXADR           CHANNEL 0 ON ADC
     0026 EFF8
     0028 04E0         CLR  @GAIN             GAIN = 1
     002A EFF6
                *
     002C 0204         LI   R4,>180           MAX +VE VAL.FOR OUTPUT ON DAC
     002E 0180
     0030 0205         LI   R5,>FE00          MAX -VE VAL FOR OUTPUT ON DAC
     0032 FE00
     0034 0201         LI   R1,>3FF           COMMANDED POSITION (CMD)
     0036 03FF
                *START MOVING ROBOT
     0038 0720  SAM    SETO @CONV             START CONVERSION
     003A EFFA
     003C 0560  CHK    INV  @STATUS           CHECK TO SEE IF DATA READY
     003E EFFC
     0040 11FD         JLT  CHK
     0042 C0A0         MOV  @ADCDAT,R2        ACTUAL POS
     0044 EFFE
     0046 6081         S    R1,R2             ERROR ACT-CMD
     0048 0282         CI   R2,>180           SEE IF ABOVE MAX +VE VEL
     004A 0180
     004C 1103         JLT  LAB1
     004E C804         MOV  R4,@DAC2          DAC 2 OUTPUT MAX +VE VEL
     0050 EFF0
     0052 10F2         JMP  SAM               GO AND START CONV.AGAIN
     0054 0283  LAB1   CI R3,>FE00            SEE IF ABOVE MAX -VE VEL
     0056 FE00
     0058 1503         JGT  LAB2
     005A C805         MOV  R5,@DAC2          DAC 2 OUTPUT MAX -VE VEL
     005C EFF0
     005E 10EC         JMP  SAM               GO AND START CONV.AGAIN
     0060 C803  LAB2   MOV  R3,@DAC2          DAC 2 OUTPUT ACTUAL VALUE
     0062 EFF0
     0064 10E9         JMP  SAM               GO AND START CONV.AGAIN
                       END
```

```
TKY1          TXMIRA        2.3.0  78.244 00:06:55    01/01/00     PAGE 0002

    ADCDAT   EFFE     ' CHK      003C     CONV     EFFA      DAC2     EFF0
    GAIN     EFF6     ' LAB1     0054   ' LAB2     0060      MUXADR   EFF8
    R0       0000       R1       0001     R10      000A      R11      000B
    R12      000C       R13      000D     R14      000E      R15      000F
    R2       0002       R3       0003     R4       0004      R5       0005
    R6       0006       R7       0007     R8       0008      R9       0009
  ' SAM      0038     ' SPACE    0000     STATUS   EFFC

 0000 ERRORS
```

```
 TXXREF        2.3.0  78.244    00:07.14    01/01/00    PAGE 0001

ADCDAT   0007       0025
CHK      0023       0024
CONV     0012       0022
DAC2     0009       0029  0033  0035
GAIN     0010       0016
LAB1     0031       0028
LAB2     0035       0032
MUXADR   0011       0015
R1                  0020  0026
R2                  0025  0026  0027
R3                  0031  0035
R4                  0018  0029
R5                  0019  0033
SAM      0022       0030  0034  0036
SPACE    0006       0014
STATUS   0008       0023

THERE ARE 0016 SYMBOLS
```

```
0001                     IDT  'TRY2'
0002              *
0003              *PROGRAM TO INPUT MORE THAN ONE POS
0004              *POS ASKED FOR WHEN REACHED PREVIOUS
0005              *
0006 F800                AORG >F800
0007 F800         SPACE BSS 32
0008      EFFE    ADCDAT EQU >EFFE
0009      EFFC    STATUS EQU >EFFC
0010      EFF2    DAC1 EQU >EFF2
0011      EFF0    DAC2 EQU >EFF0
0012      EFF6    GAIN EQU >EFF6
0013      EFF8    MUXADR EQU >EFF8
0014      EFFA    CONV EQU >EFFA
0015              *
0016 F820 04E0           CLR  @GAIN            GAIN=1
     F822 EFF6
0017 F824 04E0           CLR  @MUXADR          CHANNEL 0 ON ADC
     F826 EFF8
0018              *
0019 F828 0204           LI   R4,>180          MAX +VE VAL.FOR OUT.ON DAC
     F82A 0180
0020 F82C 0205           LI   R5,>FE80         MAX -VE VAL.FOR OUT.ON DAC
     F82E FE80
0021              *
0022 F830 2FA0    NEXT   XOP  @MESS1,14        INPUT POSITION
     F832 F092
0023 F834 2E41    NULL   XOP  R1,9             READ REQ'D POS INTO R1
0024 F836 F834           DATA NULL
0025 F838 F07A           DATA ERROR
0026 F83A 0281           CI   R1,>7FF          SEE IF VALUE WITHIN LIMITS
     F83C 07FF
0027 F83E 152F           JGT  ERR
0028 F840 0281           CI   R1,>0
     F842 0000
0029 F844 112C           JLT  ERR
0030              *
0031 F846 0720    SAM    SETO @CONV            START CONVERSION
     F848 EFFA
0032 F84A 0560    CHK    INV  @STATUS          CHECK TO SEE IF IN POS
     F84C EFFC
0033 F84E 10FD           JMP  CHK
0034 F850 C0A0           MOV  @ADCDAT,R2       GET ACTUAL POS
     F852 EFFE
0035 F854 6081           S    R1,R2            ERROR,ACT-CMD,ANS IN R2
0036 F856 0282           CI   R2,0             CHECK TO SEE IF IN POS.
     F858 0000
0037 F85A 13EA           JEQ  NEXT             IF IN POS.ASK FOR NEXT POS.
0038 F85C 0282           CI   R2,>100          COMPARE WITH MAX VALUE
     F85E 0100
0039 F860 1103           JLT  LAB1
0040 F862 C804           MOV  R4,@DAC2         OUTPUT +1.8V
     F864 EFF0
0041 F866 10EF           JMP  SAM              GO AND START CONV.AGAIN
```

```
  0042 F868 0283  LAD1    CI    R3,>FE80            COMPARE WITH MAX -VE VALUE
       F86A FE80
  0043 F86C 1503          JGT   LAD2
  0044 F86E C805          MOV   R5,@DAC2            OUTPUT -1.8V
       F870 EFF0
  0045 F872 10E9          JMP   SAM                 GO AND START CONV AGAIN
  0046 F874 C803  LAD2    MOV   R3,@DAC2            OUTPUT ACTUAL VALUE
       F876 EFF0
  0047 F878 10E6          JMP   SAM                 START CONVERSION AGAIN
  0048 F87A   59  ERROR   TEXT 'YOU HAVE MADE A MISTAKE'
       F87B   4F
       F87C   55
       F87D   20
       F87E   48
       F87F   41
       F880   56
       F881   45
       F882   20
       F883   4D
       F884   41
       F885   44
       F886   45
       F887   20
       F888   41
       F889   20
       F88A   4D
       F88B   49
       F88C   53
       F88D   54
       F88E   41
       F88F   4B
       F890   45
  0049 F891   00          BYTE 0
  0050 F892 0D0A  MESS1   DATA >0D0A
  0051 F894   4E          TEXT 'NEXT POS'
       F895   45
       F896   58
       F897   54
       F898   20
       F899   50
       F89A   4F
       F89B   53
  0052 F89C   00          BYTE 0
  0053 F89E 0D0A  ERR     DATA >0D0A
  0054 F8A0   52          TEXT 'REPEAT POSITION'
       F8A1   45
       F8A2   50
       F8A3   45
       F8A4   41
       F8A5   54
       F8A6   20
       F8A7   50
       F8A8   4F
       F8A9   53
```

```
      F8AA   49
      F8AB   54
      F8AC   49
      F8AD   4F
      F8AE   4E
 0055 F8C0 10CF          JMP  NEXT
 0056                    END
```


ADCDAT	EFFE	CHK	F84A	CONV	EFFA	DAC1	EFF2
DAC2	EFF0	ERR	F89E	ERROR	F87A	GAIN	EFF6
LAB1	F860	LAB2	F874	MESS1	F892	MUXADR	EFF8
NEXT	F830	NULL	F834	R0	0000	R1	0001
R10	000A	R11	000B	R12	000C	R13	000D
R14	000E	R15	000F	R2	0002	R3	0003
R4	0004	R5	0005	R6	0006	R7	0007
R8	0008	R9	0009	SAM	F846	SPACE	F800
STATUS	EFFC						

0000 ERRORS

```
ADCDAT  0008        0034
CHK     0032        0033
CONV    0014        0031
DAC1    0010
DAC2    0011        0040  0044  0046
ERR     0053        0027  0029
ERROR   0048        0025
GAIN    0012        0016
LAB1    0042        0039
LAB2    0046        0043
MESS1   0050        0022
MUXADR  0013        0017
NEXT    0022        0037  0055
NULL    0023        0024
R1                  0023  0026  0028  0035
R2                  0034  0035  0036  0038
R3                  0042  0046
R4                  0019  0040
R5                  0020  0044
SAM     0031        0041  0045  0047
SPACE   0007
STATUS  0009        0032

THERE ARE 0022 SYMBOLS
```

```
                         IDT  'TRY3'
                  *
                  *PROGRAM TO INPUT MORE THAN ONE POS
                  *POS ASKED FOR WHEN REACHED PREVIOUS
                  *
     F800                AORG >F800
     F800         SPACE BSS 32
          EFFE    ADCDAT EQU >EFFE
          EFFC    STATUS EQU >EFFC
          EFF2    DAC1 EQU >EFF2
          EFF0    DAC2 EQU >EFF0
          EFF6    GAIN EQU >EFF6
          EFF8    MUXADR EQU >EFF8
          EFFA    CONV EQU >EFFA
                  *
     F820 04E0           CLR  @GAIN              GAIN=1
     F822 EFF6
     F824 04E0           CLR  @MUXADR            CHANNEL 0 ON ADC
     F826 EFF8
                  *
     F828 0204           LI   R4,>180            MAX +VE VALUE TO BE OUTPUT ON
     F82A 0180
     F82C 0205           LI   R5,>FE80           MAX -VE VALUE TO BE OUTPUT ON
     F82E FE80
                  *
     F830 2FA0    NEXT   XOP  @MESS1,14          INPUT POSITION
     F832 F892
     F834 2E41    NULL   XOP  R1,9               READ REQ'D POS INTO R1
     F836 F834           DATA NULL
     F838 F87A           DATA ERROR
     F83A 0281           CI   R1,>7FF            CHECK TO SEE IF VALUE WITHIN L
     F83C 07FF
     F83E 152F           JGT  ERR
     F840 0281           CI   R1,>0
     F842 0000
     F844 112C           JLT  ERR
                  *
     F846 0720    SAM    SETO @CONV              START CONVERSION
     F848 EFFA
     F84A 0560    CHK    INV  @STATUS            CHECK TO SEE IF IN POSITION
     F84C EFFC
     F84E 10FD           JMP  CHK
     F850 C0A0           MOV  @ADCDAT,R2         GET ACTUAL POS
     F852 EFFE
     F854 6081           S    R1,R2              ERROR,ACT-CMD,ANS IN R2
     F856 0282           CI   R2,0               CHECK TO SEE IF IN POSITION
     F858 0000
     F85A 13EA           JEQ  NEXT               IF IN POSITION ,ASK FOR NEXT P
     F85C 0282           CI   R2,>180            COMPARE WITH MAX VALUE
     F85E 0180
     F860 1103           JLT  LAP1
     F862 C804           MOV  R4,@DAC2           OUTPUT +1.8V
     F864 EFF0
     F866 10EF           JMP  SAM                GO AND START CONVERSION AGAIN
```

```
 0042 F868 0283  LAB1   CI    R3,>FE80            COMPARE WITH MAX -VE VALUE
      F86A FC00
 0043 F86C 1503         JGT   LAB2
 0044 F86E C805         MOV   R5,@DAC2            OUTPUT -1.8V
      F870 EFF0
 0045 F872 10E9         JMP   SAM                 GO AND START CONVERSION AGAIN
 0046 F874 C803  LAB2   MOV   R3,@DAC2            OUTPUT ACTUAL VALUE
      F876 EFF0
 0047 F878 10E6         JMP   SAM                 GO AND START CONVERSION AGAIN
 0048 F87A   59  ERROR  TEXT 'YOU HAVE MADE A MISTAKE'
      F87B   4F
      F87C   55
      F87D   20
      F87E   48
      F87F   41
      F880   56
      F881   45
      F882   20
      F883   4D
      F884   41
      F885   44
      F886   45
      F887   20
      F888   41
      F889   20
      F88A   4D
      F88B   49
      F88C   53
      F88D   54
      F88E   41
      F88F   4B
      F890   45
 0049 F891   00         BYTE 0
 0050 F892 0D0A  MESS1  DATA >0D0A
 0051 F894   4E         TEXT 'NEXT POS'
      F895   45
      F896   58
      F897   54
      F898   20
      F899   50
      F89A   4F
      F89B   53
 0052 F89C   00         BYTE 0
 0053 F89E 0D0A  ERR    DATA >0D0A
 0054 F8A0   52         TEXT 'REPEAT POSITION'
      F8A1   45
      F8A2   50
      F8A3   45
      F8A4   41
      F8A5   54
      F8A6   20
      F8A7   50
      F8A8   4F
      F8A9   53
```

```
     F8AA   49
     F8AB   54
     F8AC   49
     F8AD   4F
     F8AE   4E
0055 F8E0 108F        JMP  NEXT
0056                  END
```


ADCDAT	EFFE	CHK	F84A	CONV	EFFA	DAC1	EFF2
DAC2	EFF0	ERR	F89E	ERROR	F87A	GAIN	EFF6
LAB1	F868	LAB2	F874	MESS1	F892	MUXADR	EFF8
NEXT	F830	NULL	F834	R0	0000	R1	0001
R10	000A	R11	000B	R12	000C	R13	000D
R14	000E	R15	000F	R2	0002	R3	0003
R4	0004	R5	0005	R6	0006	R7	0007
R8	0008	R9	0009	SAM	F846	SPACE	F800
STATUS	EFFC						

0000 ERRORS

```
ADCDAT  0008    0034
CHK     0032    0033
CONV    0014    0031
DAC1    0010
DAC2    0011    0040  0044  0046
ERR     0053    0027  0029
ERROR   0048    0025
GAIN    0012    0016
LAB1    0042    0039
LAB2    0046    0043
MESS1   0050    0022
MUXADR  0013    0017
NEXT    0022    0037  0055
NULL    0023    0024
R1              0023  0026  0028  0035
R2              0034  0035  0036  0038
R3              0042  0046
R4              0019  0040
R5              0020  0044
SAM     0031    0041  0045  0047
SPACE   0007
STATUS  0009    0032

THERE ARE 0022 SYMBOLS
```

VERSATRAN CONTROL PROGRAM

```
0001                     IDT  'VERTHREE'
0003               *
0004               *PROGRAM TO READ COMMANDS LOADED INTO
0005               *MEMORY IN THE FORM OF OP CODES
0006               *PROGRAMS BEGIN AT >FE00 & END AT >FCFE
0007               *PROGRAM INSTRUCTIONS INCLUDE CODES FOR
0008               *MOVE,STOP,CONTINUE,JUMP OR DELAY
0009               *
0010 F800                AORG >F800
0011               *
0012      EFF6     GAIN   EQU >EFF6
0013               *
0014 F800          SPACE  BSS  32
0015      EFFE     ADCDAT EQU  >EFFE
0016      EFFC     STATUS EQU  >EFFC
0017      EFF2     DAC1   EQU  >EFF2
0018      EFF0     DAC2   EQU  >EFF0
0019      DEFE     DAC3   EQU >DEFE
0020      EFF8     MUXADR EQU  >EFF8
0021      EFFA     CONV   EQU  >EFFA
0022               *
0023      FE00     MEMEQU EQU  >FE00            START OF USER MEMORY
0024               *
0025 F820 0201     BEGIN  LI   R1,MEMEQU        LOAD FIRST INSTRUCTION
     F822 FE00
0026               *
0027 F824 0209            LI   R9,1             NOS.TO SELECT ADC CHANNEL
     F826 0001
0028 F828 020B            LI   R11,>2
     F82A 0002
0029               *
0030 F82C 0208            LI   R8,>FE80         CONSTANTS FOR OUTPUT ON DAC
     F82E FE80
0031 F830 0207            LI   R7,>100
     F832 0100
0032 F834 0206            LI   R6,>0000
     F836 0000
0033               *
0034 F838 C0B1     START  MOV  *R1+,R2          READ INSTRUCTION
0035 F83A C0C2            MOV  R2,R3            SAVE
0036 F83C 0242            ANDI R2,>F000         SEPARATE OP CODE
     F83E F000
0037 F840 0243            ANDI R3,>FFF          SEPARATE INSTRUCTION
     F842 0FFF
0038               *
0039 F844 0282            CI   R2,>1000
     F846 1000
0040 F848 134D            JEQ MOVEVT            IS IT MOVE VERTICAL?
0041 F84A 0282            CI   R2,>2000
     F84C 2000
0042 F84E 1351            JEQ  MOVEHZ           IS IT MOVE HORIZONTAL?
0043 F850 0282            CI   R2,>3000
     F852 3000
0044 F854 1355            JEQ  MOVESW           IS IT MOVE IN SWING?
```

```
F856 0282          CI    R2,>4000
F858 4000
F85A 1602          JNE   NDELAY
F85C 0460          B     @DELAY            IS IT A DELAY?
F85E F9AA
F860 0282   NDELAY CI    R2,>5000
F862 5000
F864 1602          JNE   NJUMP
F866 0460          B     @JUMP             IS IT A JUMP?
F868 F932
F86A 0282   NJUMP  CI    R2,>6000
F86C 6000
F86E 1602          JNE   NWH
F870 0460          B     @WRISTH           IS IT TURN WRIST HOR?
F872 F950
F874 0282   NWH    CI    R2,>7000
F876 7000
F878 1602          JNE   NWV
F87A 0460          B     @WRISTV           IS IT TURN WRIST VER?
F87C F946
F87E 0282   NWV    CI    R2,>8000
F880 8000
F882 1602          JNE   NSTOP
F884 0460          B     @STOP             IS IT A STOP?
F886 F92A
F888 0282   NSTOP  CI    R2,>9000
F88A 9000
F88C 1602          JNE   NCON
F88E 0460          B     @CONTIN           IS IT CONTINUE?
F890 F90E
F892 0282   NCON   CI    R2,>A000
F894 A000
F896 1602          JNE   NOPEN
F898 0460          B     @OPEN             IS IT GRIP OPEN?
F89A F95A
F89C 0282   NOPEN  CI    R2,>B000
F89E B000
F8A0 16CB          JNE   START
F8A2 0460          B     @CLOSE            IS IT GRIP CLOSE?
F8A4 F964
            *
            *
            *......CONVERT ROUTINE.......*
            *
            *
F8A6 0720   CONVRT SETO  @CONV             START CONVERSION
F8A8 EFFA
F8AA 0560   CHK    INV   @STATUS           SEE IF DATA READY
F8AC EFFC
F8AE 11FD          JLT   CHK
F8B0 C120          MOV   @ADCDAT,R4        ACTUAL POSITION IN R4
F8B2 EFFE
F8B4 6103          S     R3,R4             ERROR, ACT-CMD
F8B6 C144          MOV   R4,R5             SAVE ERROR
```

```
0080 F8B8 0745          ABS  R5                FIND MODULUS OF ERROR
0081 F8BA 0285          CI   R5,>0020          SEE IF NEARLY IN POSITION
     F8BC 0020
0082 F8BE 110F          JLT  LAB3              IF NEARLY THERE JUMP
0083 F8C0 0284          CI   R4,>100           SEE IF ERR.> MAX +VE VALUE
     F8C2 0100
0084 F8C4 1103          JLT  LAB1
0085 F8C6 CAB7          MOV  R7,@DAC2(R10)     OUTPUT MAX +VE VEL.
     F8C8 EFF0
0086 F8CA 10ED          JMP  CONVRT            START CONVERSION AGAIN
0087 F8CC 0284   LAB1   CI   R4,>FE00          SEE IF ERR.< MAX -VE VALUE
     F8CE FE00
0088 F8D0 1503          JGT  LAB2
0089 F8D2 CA88          MOV  R8,@DAC2(R10)     OUTPUT MAX -VE VEL.
     F8D4 EFF0
0090 F8D6 10E7          JMP  CONVRT            START CONVERSION AGAIN
0091 F8D8 CA84   LAB2   MOV  R4,@DAC2(R10)     OUTPUT ACTUAL VELOCITY
     F8DA EFF0
0092 F8DC 10E4          JMP  CONVRT            START CONVERSION AGAIN
0093 F8DE CA86   LAB3   MOV  R6,@DAC2(R10)     OUTPUT ZERO VELOCITY
     F8E0 EFF0
0094 F8E2 10AA          JMP  START             NEXT INSTRUCTION
0095             *
0096             *.......MOVE VERTICAL ROUTINE.......*
0097 F8E4 04E0   MOVEVT CLR  @GAIN
     F8E6 EFF6
0098 F8E8 04E0          CLR  @MUXADR           CHANNEL ONE
     F8EA EFF8
0099 F8EC 020A          LI   R10,0             LOAD DISPLACEMENT VECTOR
     F8EE 0000
0100 F8F0 10DA          JMP  CONVRT            GO TO CONVERT
0101             *.......MOVE HORIZONTAL ROUTINE.......*
0102 F8F2 04E0   MOVEHZ CLR  @GAIN
     F8F4 EFF6
0103 F8F6 C809          MOV  R9,@MUXADR        CHANNEL TWO
     F8F8 EFF8
0104 F8FA 020A          LI   R10,2             LOAD DISPLACEMENT VECTOR
     F8FC 0002
0105 F8FE 10D3          JMP  CONVRT            JUMP TO CONVERT
0106             *
0107             *
0108             *.......MOVE IN SWING ROUTINE.......*
0109 F900 04E0   MOVESW CLR  @GAIN
     F902 EFF6
0110 F904 C80B          MOV  R11,@MUXADR
     F906 EFF8
0111 F908 020A          LI   R10,>EF0E
     F90A EF0E
0112 F90C 10CC          JMP  CONVRT            JUMP TO CONVERT
0113             *
0114             *
0115             *.......CONTINUE ROUTINE.......*
0116 F90E 0283   CONTIN CI   R3,>0000          IF SO CONT.UNTIL STOPPED
     F910 0000
```

```
0117 F912 1602          JNE   NEB
0118 F914 0460          B     @BEGIN            CONTINUE UNTIL STOPPED
     F916 F820
0119 F918 0283  NEB     CI    R3,>0001          IS IT LAST CYCLE?
     F91A 0001
0120 F91C 1306          JEQ   STOP              IF SO STOP
0121 F91E 0603          DEC R3                  COUNY DOWN NO OF CYCLES
0122 F920 A0C2          A     R2,R3
0123 F922 0641          DECT R1                 RESET INST.POINTER
0124 F924 C443          MOV   R3,*R1            REPLACE MODIFIED INST.
0125 F926 0460          B     @BEGIN
     F928 F820
0126            *
0127            *..........STOP ROUTINE........*
0128 F92A 2FA0  STOP    XOP   @ST,14            YOUR PROGRAM IS COMPLETE
     F92C F97A
0129 F92E 0460          B     @>80
     F930 0080
0130            *
0131            *...........JUMP ROUTINE...........*
0132 F932 0243  JUMP    ANDI R3,>FFF
     F934 0FFF
0133 F936 1602          JNE   JMPF
0134 F938 0223          AI    R3,>F000          MAKE -VE FOR JUMP BACK
     F93A F000
0135 F93C 0A13  JMPF    SLA   R3,1
0136 F93E 0641          DECT R1
0137 F940 A0C1          A     R1,R3             POINT TO JMP INST.& ADD JMP
0138 F942 0460          B     @START
     F944 F838
0139            *
0140            *............WRIST VERTICAL & HORIZONTAL ROUTINE........*
0141 F946 020C  WRISTV LI     R12,>100          BASE ADD CRU
     F948 0100
0142 F94A 1E02          SBZ   2                 SEND SIGNAL
0143 F94C 0460          B     @RTD
     F94E F96E
0144            *
0145 F950 020C  WRISTH LI     R12,>100          BASE ADD CRU
     F952 0100
0146 F954 1D02          SBO   2
0147 F956 0460          B     @RTD
     F958 F96E
0148            *
0149            *.........GRIPPER OPEN & CLOSE ROUTINE..........*
0150            *
0151 F95A 020C  OPEN    LI    R12,>100
     F95C 0100
0152 F95E 1D04          SBO   4                 SEND SIGNAL
0153 F960 0460          B     @RTD
     F962 F96E
0154            *
0155 F964 020C  CLOSE   LI    R12,>100
     F966 0100
```

```
  0156 F968 1E04          SBZ  4                  SEND SIGNAL
  0157 F96A 0460          B    @RTD
       F96C F96E
  0158               *
  0159               *.........REAL TIME DELAY(0.2 SEC'S).........*
  0160               *
  0161 F96E 0206    RTD   LI   R6,>7000           APROX EQU 0.2 SEC'S
       F970 7000
  0162 F972 0606    DEC   DEC  R6
  0163 F974 16FE          JNE  DEC
  0164 F976 0460          B    @START
       F978 F838
  0165               *
  0166               *.................MESSAGES.................*
  0167 F97A 0D0A    ST    DATA >0D0A
  0168 F97C   59          TEXT 'YOUR PROGRAM IS COMPLETE,RETURN TO MONITOR'
       F97D   4F
       F97E   55
       F97F   52
       F980   20
       F981   50
       F982   52
       F983   4F
       F984   47
       F985   52
       F986   41
       F987   4D
       F988   20
       F989   49
       F98A   53
       F98B   20
       F98C   43
       F98D   4F
       F98E   4D
       F98F   50
       F990   4C
       F991   45
       F992   54
       F993   45
       F994   2C
       F995   52
       F996   45
       F997   54
       F998   55
       F999   52
       F99A   4E
       F99B   20
       F99C   54
       F99D   4F
       F99E   20
       F99F   4D
       F9A0   4F
       F9A1   4E
       F9A2   49
```

```
     F9A3  54
     F9A4  4F
     F9A5  52
0169 F9A6 0D0A         DATA )0D0A,)0000
     F9A8 0000
0170              *.......TIME DELAY ROUTINE.......*
0171 F9AA 020C   DELAY  LI   R12,)100
     F9AC 0100
0172 F9AE 04E0          CLR  @)FF8A           CLEAR REG 0 OF T.S.R.
     F9B0 FF8A
0173 F9B2 1E00          SBZ  0                INT MODE
0174 F9B4 1D03          SBO  3                ENABLE 9901 INT3
0175 F9B6 0300          LIMI 3                ENABLE 9900 INT3
     F9B8 0003
0176 F9BA C803          MOV  R3,@)FF8C        PUT NO OF QRT SEC'S IN FF8C
     F9BC FF8C
0177 F9BE 0202          LI   R2,3             COUNT ONE
     F9C0 0003
0178 F9C2 33C2          LDCR R2,15            LOAD TIMER
0179 F9C4 10FF   SELF   JMP  SELF             WAIT FOR INT3
0180 F9C6 0460          B    @START
     F9C8 F838
0181              *
0182              *........TIMER SERVICE ROUTINE(T.S.R).......*
0183 FA40               AORG )FA40            WP=FF68,PC=FA40
0184 FA40 8040   TSR    C    R0,R1            IS THE DELAY FINISHED?
0185 FA42 130B          JEQ  NODLAY           DELAY IS FINISHED
0186 FA44 0580          INC  R0               COUNT FOR NO. QRT SEC'S REQD
0187 FA46 020C          LI   R12,)100
     FA48 0100
0188 FA4A 0202          LI   R2,)FE9F         1/4 SEC'S IN UNITS OF 21.33mic
     FA4C FE9F
0189 FA4E 33C2          LDCR R2,15            LOAD TIMER
0190 FA50 1E00          SBZ  0
0191 FA52 1D03          SBO  3
0192 FA54 0300          LIMI 3
     FA56 0003
0193 FA58 0380          RTWP
0194 FA5A 04C0   NODLAY CLR  R0               RESET REG 0
0195 FA5C 04CF          CLR  R15              CLEAR STATUS IN OLD WP
0196 FA5E 05CE          INCT R14              RESET PC TO NEXT PLACE
0197 FA60 0380          RTWP
0198              *
0199              *.................T.S.R MEMORY SPACE...............*
0200 FFAA               AORG )FFAA            WP=FF8A,PC=FFAA
0201 FFAA 0460          B    @TSR
     FFAC FA40
0202              *
0203              *.................................................*
0204 FFAE 0340          IDLE
0205                    END
```

VERSATRAN CONTROL PROGRAM

ADCDAT	EFFE	BEGIN	F820	CHK	F8AA	CLOSE	F964
CONTIN	F90E	CONV	EFFA	CONVRT	F8A6	DAC1	EFF2
DAC2	EFF0	DAC3	DEFE	DEC	F972	DELAY	F9AA
GAIN	EFF6	JMPF	F93C	JUMP	F932	LAB1	F8CC
LAB2	F8D8	LAB3	F8DE	MEMEQU	F800	MOVEHZ	F8F2
MOVESW	F900	MOVEVT	F8E4	MUXADR	EFF8	NEB	F918
NCON	F892	NDELAY	F860	NJUMP	F86A	NODLAY	FA5A
NOPEN	F89C	NSTOP	F888	NWH	F874	NWV	F87E
OPEN	F95A	R0	0000	R1	0001	R10	000A
R11	000B	R12	000C	R13	000D	R14	000E
R15	000F	R2	0002	R3	0003	R4	0004
R5	0005	R6	0006	R7	0007	R8	0008
R9	0009	RTD	F96E	SELF	F9C4	SPACE	F800
ST	F97A	START	F838	STATUS	EFFC	STOP	F92A
TSR	FA40	WRISTH	F950	WRISTV	F946		

0000 ERRORS

```
ADCDAT  0015    0077
BEGIN   0025    0118  0125
CHK     0075    0076
CLOSE   0155    0068
CONTIN  0116    0062
CONV    0021    0074
CONVRT  0074    0086  0090  0092  0100  0105  0112
DAC1    0017
DAC2    0018    0085  0089  0091  0093
DAC3    0019
DEC     0162    0163
DELAY   0171    0047
GAIN    0012    0097  0102  0109
JMPF    0135    0133
JUMP    0132    0050
LAB1    0087    0084
LAB2    0091    0088
LAB3    0093    0082
MEMEQU  0023    0025
MOVEHZ  0102    0042
MOVESW  0109    0044
MOVEVT  0097    0040
MUXADR  0020    0098  0103  0110
NEB     0119    0117
NCON    0063    0061
NDELAY  0048    0046
NJUMP   0051    0049
NODLAY  0194    0185
NOPEN   0066    0064
NSTOP   0060    0058
NWH     0054    0052
NWV     0057    0055
OPEN    0151    0065
R0              0184  0186  0194
R1              0025  0034  0123  0124  0136  0137  0184
R10             0085  0089  0091  0093  0099  0104  0111
R11             0028  0110
R12             0141  0145  0151  0155  0171  0187
R14             0196
R15             0195
R2              0034  0035  0036  0039  0041  0043  0045  0048  0051
                0054  0057  0060  0063  0066  0122  0177  0178  0188
                0189
R3              0035  0037  0078  0116  0119  0121  0122  0124  0132
                0134  0135  0137  0176
R4              0077  0078  0079  0083  0087  0091
R5              0079  0080  0081
R6              0032  0093  0161  0162
R7              0031  0085
R8              0030  0089
R9              0027  0103
RTD     0161    0143  0147  0153  0157
SELF    0179    0179
SPACE   0014
ST      0167    0120
```

```
TXXREF          2.3.0  78.244    00:01:49    01/01/00    PAGE 0002

START   0034        0067  0094  0138  0164  0180
STATUS  0016        0075
STOP    0128        0059  0120
TSR     0184        0201
WRISTH  0145        0053
WRISTV  0141        0056

THERE ARE 0058 SYMBOLS
```

```
                       IDT 'INT1'
              *PROGRAM TO MOVE 3 AXES
              *USING INTERRUPTS
              *TIME  DELAY
              *
              *
     F800              AORG >F800
     F800     SPACE    BSS  32
              *
          EFFE ADCDAT  EQU  >EFFE
          EFFC STATUS  EQU  >EFFC
          EFF2 DAC1    EQU  >EFF2
          EFF0 DAC2    EQU  >EFF0
          DEFE DAC3    EQU  >DEFE
          EFF8 MUXADR  EQU  >EFF8
          EFFA CONV    EQU  >EFFA
          EFF6 GAIN    EQU  >EFF6
              *
     F820 02E0         LWPI SPACE
     F822 F800
              *CONSTANTS FOR VELOCITIES
     F824 0203         LI   R3,>180                 MAX +VE VALUE
     F826 0180
     F828 C803         MOV  R3,@>FD0C
     F82A FD0C
     F82C 0203         LI   R3,>FE80                MAX -VE VALUE
     F82E FE80
     F830 C803         MOV  R3,@>FD0E
     F832 FD0E
     F834 0203         LI   R3,0
     F836 0000
     F838 C803         MOV  R3,@>FD10        ZERO VEOCITY
     F83A FD10
              *
              *INPUT POSITIONS
     F83C 2FA0         XOP  @MESS1,14        HOW MANY POS'S?
     F83E F969
     F840 2E42 NULL1   XOP  R2,9             NO OF POSITIONS IN R2
     F842 F840         DATA NULL1
     F844 F840         DATA NULL1
     F846 0208         LI   R8,>FD20         BASE ADD FOR TAB.OF DELAYS
     F848 FD20
     F84A 0201         LI   R1,>FB00         BASE ADD FOR POS TABLE
     F84C FB00
     F84E C802         MOV  R2,@>FD12        SAVE NO OF POSITIONS
     F850 FD12
     F852 2FA0 MES     XOP  @MESS2,14        POSITION OF AXIS ONE
     F854 F97B
     F856 2E71 NULL2   XOP  *R1+,9           READ POS INTO ADDRESS IN R1,
     F858 F856         DATA NULL2            INCREMENT R1 BY TWO
     F85A F856         DATA NULL2
     F85C 2FA0         XOP  @MESS3,14        POSITION OF AXIS TWO
     F85E F991
     F860 2E71 NULL3   XOP  *R1+,9
```

```
0042 F862 F860          DATA NULL3
0043 F864 F860          DATA NULL3
0044 F866 2FA0          XOP  @MESS4,14         POSITION OF AXIS THREE
     F868 F9A7
0045 F86A 2E71   NULL4  XOP  *R1+,9
0046 F86C F86A          DATA NULL4
0047 F86E F86A          DATA NULL4
0048 F870 2FA0   NULL5  XOP  @MESS5,14         TIMES GO ROUND DELAY LOOP
     F872 F9BD
0049 F874 2C78          XOP  *R8+,9
0050 F876 F870          DATA NULL5
0051 F878 F870          DATA NULL5
0052 F87A 0602          DEC  R2                DEC NO OF POS'S LEFT
0053 F87C 0282          CI   R2,0
     F87E 0000
0054 F880 16E8          JNE  MES
0055 F882 04C9          CLR  R9                REG.FOR MEM FOR DELAYS
0056 F884 04E0          CLR  @>FD00            MEM.FOR ACTUAL POS'S
     F886 FD00
0057 F888 04E0          CLR  @>FD02
     F88A FD02
0058 F88C 04E0          CLR  @>FD04
     F88E FD04
0059 F890 04E0          CLR  @>FD06            MEM.  FOR REQ'D POS'S
     F892 FD06
0060 F894 04E0          CLR  @>FD08
     F896 FD08
0061 F898 04E0          CLR  @>FD0A
     F89A FD0A
0062             *
0063 F89C 0208          LI   R8,>FD20          MEM.FOR START OF DELAYS
     F89E FD20
0064 F8A0 0204          LI   R4,>FE00          MEM.FOR START OF POSITIONS
     F8A2 FE00
0065 F8A4 0205          LI   R5,>FD06          MEM.FOR REQ'D POSITIONS
     F8A6 FD06
0066 F8A8 CD74          MOV  *R4+,*R5+         MOVE REQ'D POS FROM FE00
0067 F8AA CD74          MOV  *R4+,*R5+         ONWARDS TO FD06,FD08,FD0A
0068 F8AC CD74          MOV  *R4+,*R5+
0069 F8AE 020C          LI   R12,>100          BASE ADDRESS
     F8B0 0100
0070 F8B2 1E00          SBZ  0                 INTERRUPT MODE
0071 F8B4 1D03          SBO  3                 ENABLE INTERRUPT ON 9901
0072 F8B6 0300          LIMI 3                 ENABLE INTERRUPT ON 9900
     F8B8 0003
0073 F8BA 0200          LI   R0,3              COUNT=1 CLOCK MODE
     F8BC 0003
0074 F8BE 33C0          LDCR R0,15             START COUNT
0075             *RETURN TO START IF NO DELAY
0076             *
0077 F8C0 1E00   START  SBZ  0                 INTERRUPT EVERY 38 MS
0078 F8C2 1D03          SBO  3
0079 F8C4 0300          LIMI 3
     F8C6 0003
```

```
F8C8 0200          LI   R0,>300F
F8CA 300F
F8CC 33C0          LDCR R0,15
F8CE C1A0          MOV  @>FD08,R6          SAVE REQ'D POS
F8D0 FD08
F8D2 61A0          S    @>FD02,R6          CHECK TO SEE IF IN POS
F8D4 FD02
F8D6 0746          ABS  R6                 ABSOLUTE ERROR
F8D8 0286          CI   R6,>28             ALLOW FOR SLIGHT ERROR
F8DA 0028
F8DC 1101          JLT  NEX                IF NEARLY THERE NEXT AXIS
F8DE 1011          JMP  SELF               OTHERWISE AWAIT INTERRUPT
F8E0 C0A0    NEX   MOV  @>FD06,R2          REPEAT FOR ALL OTHER AXES
F8E2 FD06
F8E4 60A0          S    @>FD00,R2
F8E6 FD00
F8E8 0742          ABS  R2
F8EA 0282          CI   R2,>28
F8EC 0028
F8EE 1101          JLT  NEX2
F8F0 1008          JMP  SELF
F8F2 C1E0    NEX2  MOV  @>FD0A,R7
F8F4 FD0A
F8F6 61E0          S    @>FD04,R7
F8F8 FD04
F8FA 0747          ABS  R7
F8FC 0287          CI   R7,>28
F8FE 0028
F900 1101          JLT  DEL                IF NEARLY THERE NEXT INST.
F902 10FF    SELF  JMP  SELF               ELSE AWAIT INTERRUPT
             *CHECK TO SEE IF LAST POSITION
F904 0620    DEL   DEC  @>FD12             DEC NO OF POS'S LEFT
F906 FD12
F908 C1A0          MOV  @>FD12,R6          SAVE
F90A FD12
F90C 0286          CI   R6,0               SEE IF LAST POSITION
F90E 0000
F910 1312          JEQ  STOP               IF LAST THEN STOP ROUTINE
F912 C278          MOV  *R8+,R9            OTHERWISE NEXT DELAY
F914 10FF    SELF2 JMP  SELF2              AWAIT INTERRUPT
             *RETURN TO CONT FROM INTERRUPT IF THERE IS A DELAY
F916 1E00    CONT  SBZ  0                  INTERRUPT EVERY 30mS
F918 1D03          SBO  3
F91A 0300          LIMI 3
F91C 0003
F91E 0200          LI   R0,>300F
F920 300F
F922 33C0          LDCR R0,15
F924 0289          CI   R9,0               SEE IF DELAY FIN.
F926 0000
F928 1605          JNE  SELF3              IF NOT AWAIT INTERRUPT
F92A 0205          LI   R5,>FD06           LOCATION FOR CMD
F92C FD06
F92E CD74          MOV  *R4+,*R5+          MOVE CMD TO >FD06, >FD08, >FD0A
```

```
 0117 F930 CD74          MOV  *R4+,*R5+
 0118 F932 CD74          MOV  *R4+,*R5+
 0119 F934 10FF  SELF3   JMP  SELF3             AWAIT INTERRUPT
 0120            *STOP ROUTINE
 0121 F936 0300  STOP    LIMI 0                 DISENABLE ALL INTERRUPTS
      F938 0000
 0122 F93A 2FA0          XOP  @MESS,14          PROGRAM FINISHED
      F93C F944
 0123 F93E 0460          B    @>80              RETURN TO MONITOR
      F940 0080
 0124 F942 0340          IDLE
 0125 F944   50  MESS    TEXT 'PROGRAM FINISHED RETURN TO MONITOR'
      F945   52
      F946   4F
      F947   47
      F948   52
      F949   41
      F94A   4D
      F94B   20
      F94C   46
      F94D   49
      F94E   4E
      F94F   49
      F950   53
      F951   48
      F952   45
      F953   44
      F954   20
      F955   52
      F956   45
      F957   54
      F958   55
      F959   52
      F95A   4E
      F95B   20
      F95C   54
      F95D   4F
      F95E   20
      F95F   4D
      F960   4F
      F961   4E
      F962   49
      F963   54
      F964   4F
      F965   52
 0126 F966 0D0A          DATA >0D0A
 0127 F968   00          BYTE 0
 0128 F969   4E  MESS1   TEXT 'NO OF POSITIONS'
      F96A   4F
      F96B   20
      F96C   4F
      F96D   46
      F96E   20
      F96F   50
```

```
     F970  4F
     F971  53
     F972  49
     F973  54
     F974  49
     F975  4F
     F976  4E
     F977  53
0129 F978 0D0A          DATA >0D0A
0130 F97A  00           BYTE 0
0131 F97B  50   MESS2   TEXT 'POSITION OF AXIS 1'
     F97C  4F
     F97D  53
     F97E  49
     F97F  54
     F980  49
     F981  4F
     F982  4E
     F983  20
     F984  4F
     F985  46
     F986  20
     F987  41
     F988  58
     F989  49
     F98A  53
     F98B  20
     F98C  31
0132 F98E 0D0A          DATA >0D0A
0133 F990  00           BYTE 0
0134 F991  50   MESS3   TEXT 'POSITION OF AXIS 2'
     F992  4F
     F993  53
     F994  49
     F995  54
     F996  49
     F997  4F
     F998  4E
     F999  20
     F99A  4F
     F99B  46
     F99C  20
     F99D  41
     F99E  58
     F99F  49
     F9A0  53
     F9A1  20
     F9A2  32
0135 F9A4 0D0A          DATA >0D0A
0136 F9A6  00           BYTE 0
0137 F9A7  50   MESS4   TEXT 'POSITION OF AXIS 3'
     F9A8  4F
     F9A9  53
     F9AA  49
```

```
     F9AB   54
     F9AC   49
     F9AD   4F
     F9AE   4E
     F9AF   20
     F9B0   4F
     F9B1   46
     F9B2   20
     F9B3   41
     F9B4   58
     F9B5   49
     F9B6   53
     F9B7   20
     F9B8   33
0138 F9BA 0D0A          DATA >0D0A
0139 F9BC   00          BYTE 0
0140 F9BD   44   MESS5  TEXT 'DELAY'
     F9BE   45
     F9BF   4C
     F9C0   41
     F9C1   59
0141 F9C2 0D0A          DATA >0D0A
0142 F9C4   00          BYTE 0
0143             *INTERRUPT ROUTINE
0144             *
0145 FA00               AORG >FA00            ORIGIN
0146 FA00 FAC0          DATA >FAC0            NEW WP
0147 FA02 FA04          DATA >FA04            NEW PC
0148 FA04 0300          LIMI 0                DISABLE ALL INTERRUPTS
     FA06 0000
0149 FA08 0200          LI   R0,3             NO OF AXES
     FA0A 0003
0150 FA0C 0208          LI   R8,1             NO.TO SELECT ADC CHANNEL
     FA0E 0001
0151 FA10 0209          LI   R9,2             NO .TO SELECT ADC CHANNEL
     FA12 0002
0152 FA14 04E0          CLR  @GAIN            GAIN = 1
     FA16 EFF6
0153             *
0154 FA18 04E0   AXIS1  CLR  @MUXADR          CHANNEL 0 ON ADC
     FA1A EFF8
0155 FA1C 020A          LI   R10,0            DISPLACEMENT FOR DAC
     FA1E 0000
0156 FA20 100A          JMP  CON
0157 FA22 C808   AXIS2  MOV  R8,@MUXADR       CHANNEL 1 ON ADC
     FA24 EFF8
0158 FA26 020A          LI   R10,2            DISPLACEMENT FOR DAC
     FA28 0002
0159 FA2A 1009          JMP  CONVRT
0160 FA2C C809   AXIS3  MOV  R9,@MUXADR       CHANNEL 2 ON ADC
     FA2E EFF8
0161 FA30 020A          LI   R10,>EF0E        DISPLACKMENT FOR DAC
     FA32 EF0E
0162 FA34 1004          JMP  CONVRT
```

```
 0163 FA36 0203  CON     LI    R3,>FD00           MEM.FOR ACTUAL POS'S
      FA38 FD00
 0164 FA3A 0205          LI    R5,>FD06           MEM.FOR REQ'D POS'S
      FA3C FD06
 0165            *
 0166 FA3E 0720  CONVRT SETO @CONV                START CONVERSION
      FA40 CFFA
 0167 FA42 0560  CHK     INV   @STATUS            CHECK DATA READY
      FA44 EFFC
 0168 FA46 11FD          JLT   CHK
 0169 FA48 C060          MOV   @ADCDAT,R1         ACTUAL POS IN R1
      FA4A EFFE
 0170 FA4C CCC1          MOV   R1,*R3+            MOVE TO MEMLOC
 0171 FA4E 6075          S     *R5+,R1            ACTUAL-CMD
 0172 FA50 C081          MOV   R1,R2              SAVE
 0173 FA52 0742          ABS   R2                 ABSOLUTE ERROR
 0174 FA54 0282          CI    R2,>28             SEE IF NEARLY IN POS.
      FA56 0028
 0175 FA58 1111          JLT   LAB3
 0176 FA5A 0281          CI    R1,>180            SEE IF ERR. > MAX + VALUE
      FA5C 0180
 0177 FA5E 1104          JLT   LAB1
 0178 FA60 CAA0          MOV   @>FD0C,@DAC2(R10) OUTPUT +1.8V
      FA62 FD0C
      FA64 EFF0
 0179 FA66 100D          JMP   TEST               SEE IF ANY MORE AXES
 0180 FA68 0281  LAB1    CI    R1,>FE80           SEE IF ERR.< MAX -VE VALUE
      FA6A FE80
 0181 FA6C 1504          JGT   LAB2
 0182 FA6E CAA0          MOV   @>FD0E,@DAC2(R10) OUTPUT-1.8V
      FA70 FD0E
      FA72 EFF0
 0183 FA74 1006          JMP   TEST
 0184 FA76 CA81  LAB2    MOV   R1,@DAC2(R10)      OUTPUT ACTUAL VELOCITY
      FA78 EFF0
 0185 FA7A 1003          JMP   TEST
 0186 FA7C CAA0  LAB3    MOV   @>FD10,@DAC2(R10) OUTPUT 0V
      FA7E FD10
      FA80 EFF0
 0187 FA82 0600  TEST    DEC   R0                 DEC. COUNT FOR NO OF AXES
 0188 FA84 0280          CI    R0,2
      FA86 0002
 0189 FA88 13CC          JEQ   AXIS2              SERVICE AXIS TWO
 0190 FA8A 0280          CI    R0,1
      FA8C 0001
 0191 FA8E 13CE          JEQ   AXIS3              SERVICE AXIS THREE
 0192 FA90 C1A0          MOV   @>F812,R6          SAVE DEL.COUNT FROM OLD R9
      FA92 F812
 0193 FA94 0286          CI    R6,0
      FA96 0000
 0194 FA98 1606          JNE   CONTD              IF <>0 GO TO CONTD ROUTINE
 0195 FA9A 020D          LI    R13,>F800          WP FOR MAIN PROGRAM
      FA9C F800
 0196 FA9E 020E          LI    R14,START          PC FOR RETURN
```

```
     FAA0 F8C0
     FAA2 04CF          CLR   R15             CLEAR STATUS
     FAA4 0380          RTWP                  RETURN TO MAIN PROGRAM
                  *
     FAA6 0620  CONTD   DEC   @>F812          DEC.COUNT IN OLD R9,DELAY
     FAA8 F812
     FAAA 020D          LI    R13,>F800       WP OF MAIN PROGRAM.
     FAAC F800
     FAAE 020E          LI    R14,CONT        PC FOR RETURN
     FAB0 F916
     FAB2 04CF          CLR   R15             CLEAR STATUS
     FAB4 0380          RTWP                  RETURN TO MAIN PROGRAM
                  *
                  *
                  *
                  *
                  *TSR MEM SPACE
                  *
     FFAA               AORG  >FFAA
     FFAA 0420          BLWP  @>FA00          GO TO >FA00 FOR NEW PC&WP
     FFAC FA00
     FFAE 0380          RTWP
                        END
```


ADCDAT	EFFE	AXIS1	FA18	AXIS2	FA22	AXIS3	FA2C
CHK	FA42	CON	FA36	CONT	F916	CONTD	FAA6
CONV	EFFA	CONVRT	FA3E	DAC1	EFF2	DAC2	EFF0
DAC3	DEFE	DEL	F904	GAIN	EFF6	LAB1	FA68
LAB2	FA76	LAB3	FA7C	MES	F852	MESS	F944
MESS1	F969	MESS2	F97B	MESS3	F991	MESS4	F9A7
MESS5	F9ED	MUXADR	EFF8	NEX	F8E0	NEX2	F8F2
NULL1	F840	NULL2	F856	NULL3	F860	NULL4	F86A
NULL5	F870	R0	0000	R1	0001	R10	000A
R11	000B	R12	000C	R13	000D	R14	000E
R15	000F	R2	0002	R3	0003	R4	0004
R5	0005	R6	0006	R7	0007	R8	0008
R9	0009	SELF	F902	SELF2	F914	SELF3	F934
SPACE	F800	START	F8C0	STATUS	EFFC	STOP	F936
TEST	FA82						

0000 ERRORS

```
ADLDAT  0010     0169
AXIS1   0154
AXIS2   0157     0189
AXIS3   0160     0191
CHK     0167     0168
CON     0163     0156
CONT    0108     0202
CONTD   0200     0194
CONV    0016     0166
CONVRT  0166     0159  0162
DAC1    0012
DAC2    0013     0178  0182  0184  0186
DAC3    0014
DEL     0101     0098
GAIN    0017     0152
LAB1    0180     0177
LAB2    0184     0181
LAB3    0186     0175
MES     0036     0054
MESS    0125     0122
MESS1   0128     0029
MESS2   0131     0036
MESS3   0134     0040
MESS4   0137     0044
MESS5   0140     0048
MUXADR  0015     0154  0157  0160
NEX     0088     0086
NEX2    0094     0092
NULL1   0030     0031  0032
NULL2   0037     0038  0039
NULL3   0041     0042  0043
NULL4   0045     0046  0047
NULL5   0048     0050  0051
R0               0073  0074  0080  0081  0111  0112  0149  0187  0188
                 0190
R1               0034  0037  0041  0045  0169  0170  0171  0172  0176
                 0180  0184
R10              0155  0158  0161  0178  0182  0184  0186
R12              0069
R13              0195  0201
R14              0196  0202
R15              0197  0203
R2               0030  0035  0052  0053  0088  0089  0090  0091  0172
                 0173  0174
R3               0021  0022  0023  0024  0025  0026  0163  0170
R4               0064  0066  0067  0068  0116  0117  0118
R5               0065  0066  0067  0068  0115  0116  0117  0118  0164
                 0171
R6               0082  0083  0084  0085  0102  0103  0192  0193
R7               0094  0095  0096  0097
R8               0033  0049  0063  0105  0150  0157
R9               0055  0105  0113  0151  0160
SELF    0099     0087  0093  0099
SELF2   0106     0106
SELF3   0119     0114  0119
```

```
TXXREF          2.3.0  78.244    00:01:49    01/01/00    PAGE 0002

START   0034        0067  0094  0138  0164  0180
STATUS  0016        0075
STOP    0128        0059  0120
TSR     0184        0201
WRISTH  0145        0053
WRISTV  0141        0056

THERE ARE 0058 SYMBOLS
```

```
0001                   IDT 'INT3'
0002             *PROGRAM TO MOVE 3 AXES
0003             *USING INTERRUPTS
0004             *TIME  DELAY
0005             *MOVE WRIST
0006             *DATA PROGRAM SEPARATE
0007 F800              AORG >F800
0008 F800        SPACE  BSS 32
0009             *
0010      EFFE   ADCDAT EQU >EFFE
0011      EFFC   STATUS EQU >EFFC
0012      EFF2   DAC1   EQU >EFF2
0013      EFF0   DAC2   EQU >EFF0
0014      DEFE   DAC3   EQU >DEFE
0015      EFF8   MUXADR EQU >EFF8
0016      EFFA   CONV   EQU >EFFA
0017      EFF6   GAIN   EQU >EFF6
0018             *
0019 F820 02E0          LWPI SPACE
     F822 F800
0020             *CONSTANTS FOR VELOCITIES
0021 F824 0203          LI   R3,>180              MAX +VE VALUE
     F826 0180
0022 F828 C803          MOV  R3,@>FD0C
     F82A FD0C
0023 F82C 0203          LI   R3,>FE80             MAX -VE VALUE
     F82E FE80
0024 F830 C803          MOV  R3,@>FD0E
     F832 FD0E
0025 F834 0203          LI   R3,0
     F836 0000
0026 F838 C803          MOV  R3,@>FD10       ZERO VEDCITY
     F83A FD10
0027             *
0028 F83C 04C9          CLR  R9              REG FOR MEM FOR DELAYS
0029 F83E 04E0          CLR  @>FD00          MEMLOC FOR ACTUAL POS
     F840 FD00
0030 F842 04E0          CLR  @>FD02
     F844 FD02
0031 F846 04E0          CLR  @>FD04
     F848 FD04
0032 F84A 04E0          CLR  @>FD06          MEMLOC. FOR REQUIRED POS
     F84C FD06
0033 F84E 04E0          CLR  @>FD08
     F850 FD08
0034 F852 04E0          CLR  @>FD0A
     F854 FD0A
0035             *
0036 F856 020A          LI   R10,>FDA0       MEMLOC FOR CLOSE/OPEN JAWS
     F858 FDA0
0037 F85A 0208          LI   R8,>FD20        MEMLOC.FOR START OF DELAYS
     F85C FD20
0038 F85E 0204          LI   R4,>FE00        MEMLOC.FOR START OF POSITIONS
     F860 FE00
```

```
0039 F862 0205          LI   R5,>FD06          MEMLOC FOR REQ'D POSITIONS
     F864 FD06
0040 F866 020C          LI   R12,>120          PORT AREA ON CRU
     F868 0120
0041 F86A 1E0C          SBZ  12                OPEN JAWS
0042 F86C CD74          MOV  *R4+,*R5+         MOVE REQ'D POS FROM F800
0043 F86E CD74          MOV  *R4+,*R5+         ONWARDS TO FD06,FD08,FD0A
0044 F870 CD74          MOV  *R4+,*R5+
0045 F872 020C          LI   R12,>100          BASE ADDRESS
     F874 0100
0046 F876 1E00          SBZ  0                 INTERRUPT MODE
0047 F878 1D03          SBO  3                 ENABLE INTERRUPT ON 9901
0048 F87A 0300          LIMI 3                 ENABLE INTERRUPT ON 9900
     F87C 0003
0049 F87E 0200          LI   R0,3              COUNT=1 CLOCK MODE
     F880 0003
0050 F882 33C0          LDCR R0,15             START COUNT
0051               *RETURN TO START IF NO DELAY
0052               *
0053 F884 1E00    START SBZ  0                 INTERRUPT EVERY 38 MS
0054 F886 1D03          SBO  3
0055 F888 0300          LIMI 3
     F88A 0003
0056 F88C 0200          LI   R0,>300F
     F88E 300F
0057 F890 33C0          LDCR R0,15
0058 F892 C1A0          MOV @>FD08,R6          SAVE REQ'D POS
     F894 FD08
0059 F896 61A0          S    @>FD02,R6         CHECK TO SEE IF IN POS
     F898 FD02
0060 F89A 0746          ABS  R6                ABSOLUTE ERROR
0061 F89C 0286          CI   R6,>28            ALLOW FOR SLIGHT ERROR
     F89E 0028
0062 F8A0 1101          JLT  NEX               IF NEARLY THERE NEXT AXIS
0063 F8A2 1011          JMP  SELF              OTHERWISE AWAIT INTERRUPT
0064 F8A4 C0A0    NEX   MOV  @>FD06,R2         REPEAT THIS SEQ. FOR ALL AXES
     F8A6 FD06
0065 F8A8 60A0          S    @>FD00,R2
     F8AA FD00
0066 F8AC 0742          ABS  R2
0067 F8AE 0282          CI   R2,>28
     F8B0 0028
0068 F8B2 1101          JLT  NEX2
0069 F8B4 1008          JMP  SELF
0070 F8B6 C1E0    NEX2  MOV  @>FD0A,R7
     F8B8 FD0A
0071 F8BA 61E0          S    @>FD04,R7
     F8BC FD04
0072 F8BE 0747          ABS  R7
0073 F8C0 0287          CI   R7,>28
     F8C2 0028
0074 F8C4 1101          JLT  DEL               IF NEARLY THERE NEXT INSTRUCTI
0075 F8C6 10FF    SELF  JMP  SELF              OTHERWISE AWAIT INTERRUPT
0076               *CHECK TO SEE IF LAST POSITION
```

```
F8C8 0620  DEL    DEC  @>FD12          DECREASE NO OF POSITIONS LEFT
F8CA FD12
F8CC C1A0         MOV  @>FD12,R6       SAVE
F8CE FD12
F8D0 0286         CI   R6,0            SEE IF LAST POSITION
F8D2 0000
F8D4 1321         JEQ  STOP            IF LAST THEN STOP ROUTINE
F8D6 C278         MOV  *R8+,R9         OTHERWISE NEXT DELAY
                 *OPEN/CLOSE JAWS
F8D8 805A         C    *R10,R1         COMPARE JAW INST WITH 1
F8DA 1307         JEQ  OPEN            IF EQUALS ONE THEN OPEN
F8DC 05CA         INCT R10             INCREASE R10 BY TWO
F8DE 020C         LI   R12,>120        PORT AREA ON CRU
F8E0 0120
F8E2 1D0C         SBO  12              CLOSE JAWS
F8E4 020C         LI   R12,>100        INTERRUPT AREA ON CRU
F8E6 0100
F8E8 1006         JMP  SELF2           AWAIT INTERRUPT
F8EA 05CA  OPEN   INCT R10             INCREASE BY TWO R10
F8EC 020C         LI   R12,>120        PORT AREA ON CRU
F8EE 0120
F8F0 1E0C         SBZ  12              OPEN JAWS
F8F2 020C         LI   R12,>100
F8F4 0100
F8F6 10FF  SELF2  JMP  SELF2           AWAIT INTERRUPT
                 *RETURN TO CONT FROM INTERRUPT IF THERE IS A DELAY
F8F8 1E00  CONT   SBZ  0               INTERRUPT EVERY 38mS
F8FA 1D03         SBO  3
F8FC 0300         LIMI 3
F8FE 0003
F900 0200         LI   R0,>300F
F902 300F
F904 33C0         LDCR R0,15
F906 0289         CI   R9,0            CHECK TO SEE IF DELAY FIN
F908 0000
F90A 1605         JNE  SELF3           IF NOT AWAIT INTERRUPT
F90C 0205         LI   R5,>FD06        LOCATION FOR CMD
F90E FD06
F910 CD74         MOV  *R4+,*R5+       MOVE CMD TO >FD06,>FD08,>FD0A
F912 CD74         MOV  *R4+,*R5+
F914 CD74         MOV  *R4+,*R5+
F916 10FF  SELF3  JMP  SELF3           AWAIT INTERRUPT
                 *STOP ROUTINE
F918 0300  STOP   LIMI 0               DISENABLE ALL INTERRUPTS
F91A 0000
F91C 2FA0         XOP  @MESS,14        PROGRAM FINISHED
F91E F926
F920 0460         B    @>80            RETURN TO MONITOR
F922 0080
F924 0340         IDLE
F926   50  MESS   TEXT 'PROGRAM FINISHED RETURN TO MONITOR'
F927   52
F928   4F
F929   47
```

```
      F92A   52
      F92B   41
      F92C   4D
      F92D   20
      F92E   46
      F92F   49
      F930   4C
      F931   49
      F932   53
      F933   48
      F934   45
      F935   44
      F936   20
      F937   52
      F938   45
      F939   54
      F93A   55
      F93B   52
      F93C   4E
      F93D   20
      F93E   54
      F93F   4F
      F940   20
      F941   4D
      F942   4F
      F943   4E
      F944   49
      F945   54
      F946   4F
      F947   52
 0114 F948 0D0A          DATA >0D0A
 0115 F94A   00          BYTE 0
 0116              *INTERRUPT ROUTINE
 0117              *
 0118 FA00               AORG >FA00              ORIGIN
 0119 FA00 FAC0          DATA >FAC0              NEW WP
 0120 FA02 FA04          DATA >FA04              NEW PC
 0121 FA04 0300          LIMI 0                  DISENABLE ALL INTERRUPTS
      FA06 0000
 0122 FA08 0200          LI   R0,3               NO OF AXES
      FA0A 0003
 0123 FA0C 0208          LI   R8,1               NO.TO SELECT ADC CHANNEL
      FA0C 0001
 0124 FA10 0209          LI   R9,2               NO .TO SELECT ADC CHANNEL
      FA12 0002
 0125 FA14 04E0          CLR  @GAIN              GAIN = 1
      FA16 EFF6
 0126              *
 0127 FA18 04E0   AXIS1  CLR  @MUXADR            CHANNEL 0 ON ADC
      FA1A EFF8
 0128 FA1C 020A          LI   R10,0              DISPLACEMENT FOR DAC
      FA1E 0000
 0129 FA20 100A          JMP  CON
 0130 FA22 C808   AXIS2  MOV  R8,@MUXADR         CHANNEL 1 ON ADC
```

```
     FA24 EFF8
0131 FA26 020A          LI   R10,2              DISPLACEMENT FOR DAC
     FA28 0002
0132 FA2A 1009          JMP  CONVRT
0133 FA2C C809   AXIS3  MOV  R9,@MUXADR         CHANNEL 2 ON ADC
     FA2E EFFB
0134 FA30 020A          LI   R10,>EF0E          DISPLACKMENT FOR DAC
     FA32 EF0E
0135 FA34 1004          JMP  CONVRT
0136 FA36 0203   CON    LI   R3,>FD00           MEMLOC FOR ACTUAL POSITIONS
     FA38 FD00
0137 FA3A 0205          LI   R5,>FD06           MEMLOC FOR REQ'D POSITIOS
     FA3C FD06
0138             *
0139 FA3E 0720   CONVRT SETO @CONV              START CONVERSION
     FA40 EFFA
0140 FA42 0560   CHK    INV  @STATUS            CHECK DATA READY
     FA44 EFFC
0141 FA46 11FD          JLT  CHK
0142 FA48 C060          MOV  @ADCDAT,R1         ACTUAL POS.IN R1
     FA4A EFFE
0143 FA4C CCC1          MOV  R1,*R3+            MOVE TO MEMLOC
0144 FA4E 6075          S    *R5+,R1            ACTUAL-CMD
0145 FA50 C081          MOV  R1,R2              SAVE
0146 FA52 0742          ABS  R2                 ABSOLUTE ERROR
0147 FA54 0282          CI   R2,>28             SEE IF NEARLY IN POS
     FA56 0028
0148 FA58 1111          JLT  LAB3
0149 FA5A 0281          CI   R1,>180            SEE IF ERROR > MAX + VAL
     FA5C 0180
0150 FA5E 1104          JLT  LAB1
0151 FA60 CAA0          MOV  @>FD0C,@DAC2(R10) OUTPUT +1.8V
     FA62 FD0C
     FA64 EFF0
0152 FA66 100D          JMP  TEST               SEE IF ANY MORE AXES
0153 FA68 0281   LAB1   CI   R1,>FE80           SEE IF ERROR < MAX -VE VAL
     FA6A FE80
0154 FA6C 1504          JGT  LAB2
0155 FA6E CAA0          MOV  @>FD0E,@DAC2(R10) OUTPUT-1.8V
     FA70 FD0E
     FA72 EFF0
0156 FA74 1006          JMP  TEST
0157 FA76 CA81   LAB2   MOV  R1,@DAC2(R10)      OUTPUT ACTUAL VELOCITY
     FA78 EFF0
0158 FA7A 1003          JMP  TEST
0159 FA7C CAA0   LAB3   MOV  @>FD10,@DAC2(R10) OUTPUT 0V
     FA7E FD10
     FA80 EFF0
0160 FA82 0600   TEST   DEC  R0                 DEC COUNT FOR NO OF AXES
0161 FA84 0280          CI   R0,2
     FA86 0002
0162 FA88 13CC          JEQ  AXIS2              SERVICE AXIS TWO
0163 FA8A 0280          CI   R0,1
     FA8C 0001
```

```
0164 FA8E 13CE          JEQ   AXIS3             SERVICE AXIS THREE
0165 FA90 C1A0          MOV   @>F812,R6         SAVE DEL COUNT FROM OLD R9
     FA92 F812
0166 FA94 0286          CI    R6,0
     FA96 0000
0167 FA98 1606          JNE   CONTD             IF <>0 GO TO CONTD ROUTINE
0168 FA9A 020D          LI    R13,>F800         WP FOR MAIN PROGRAM
     FA9C F000
0169 FA9E 020E          LI    R14,START         PC FOR RETURN
     FAA0 F804
0170 FAA2 04CF          CLR   R15               CLEAR STATUS
0171 FAA4 0380          RTWP                    RETURN TO MAIN PROGRAM
0172               *
0173 FAA6 0620     CONTD DEC  @>F812            DEC COUNT IN R9 FOR DELAY
     FAA8 F812
0174 FAAA 020D          LI    R13,>F800         WP OF MAIN PROGRAM
     FAAC F800
0175 FAAE 020E          LI    R14,CONT          PC FOR RETURN
     FAB0 F8F8
0176 FAB2 04CF          CLR   R15               CLEAR STATUS
0177 FAB4 0380          RTWP                    RETURN TO MAIN PROGRAM
0178               *
0179               *
0180               *
0181               *
0182               *TSR MEM SPACE
0183               *
0184 FFAA               AORG  >FFAA
0185 FFAA 0420          BLWP  @>FA00            GO TO >FA00 TO PICK UP NEW PC&
     FFAC FA00
0186 FFAE 0380          RTWP
0187                    END
```


ADCDAT	EFFE	AXIS1	FA18	AXIS2	FA22	AXIS3	FA2C
CHK	FA42	CON	FA36	CONT	F8F8	CONTD	FAA6
CONV	EFFA	CONVRT	FA3E	DAC1	EFF2	DAC2	EFF0
DAC3	DEFE	DEL	F8C8	GAIN	EFF6	LAB1	FA68
LAB2	FA76	LAB3	FA7C	MESS	F926	MUXADR	EFF8
NEX	F8A4	NEX2	F866	OPEN	F8EA	R0	0000
R1	0001	R10	000A	R11	000B	R12	000C
R13	000D	R14	000E	R15	000F	R2	0002
R3	0003	R4	0004	R5	0005	R6	0006
R7	0007	R8	0008	R9	0009	SELF	F8C6
SELF2	F8F6	SELF3	F916	SPACE	F800	START	F884
STATUS	EFFC	STOP	F918	TEST	FA82		

0000 ERRORS

```
TXXREF         2.3.0  78.244    00:13:15    01/01/00    PAGE 0001

ADCDAT  0010      0142
AXIS1   0127
AXIS2   0130      0162
AXIS3   0133      0164
CHK     0140      0141
CON     0136      0129
CONT    0096      0175
CONTD   0173      0167
CONV    0016      0139
CONVRT  0139      0132  0135
DAC1    0012
DAC2    0013      0151  0155  0157  0159
DAC3    0014
DEL     0077      0074
GAIN    0017      0125
LAB1    0153      0150
LAB2    0157      0154
LAB3    0159      0148
MESS    0113      0110
MUXADR  0015      0127  0130  0133
NEX     0064      0062
NEX2    0070      0068
OPEN    0090      0084
R0                0049  0050  0056  0057  0099  0100  0122  0160  0161
                  0163
R1                0083  0142  0143  0144  0145  0149  0153  0157
R10               0036  0083  0085  0090  0128  0131  0134  0151  0155
                  0157  0159
R12               0040  0045  0086  0088  0091  0093
R13               0168  0174
R14               0169  0175
R15               0170  0176
R2                0064  0065  0066  0067  0145  0146  0147
R3                0021  0022  0023  0024  0025  0026  0136  0143
R4                0038  0042  0043  0044  0104  0105  0106
R5                0039  0042  0043  0044  0103  0104  0105  0106  0137
                  0144
R6                0058  0059  0060  0061  0078  0079  0165  0166
R7                0070  0071  0072  0073
R8                0037  0081  0123  0130
R9                0028  0081  0101  0124  0133
SELF    0075      0063  0069  0075
SELF2   0094      0089  0094
SELF3   0107      0102  0107
SPACE   0008      0019
START   0053      0169
STATUS  0011      0140
STOP    0109      0080
TEST    0160      0152  0156  0158

THERE ARE 0046 SYMBOLS
```

```
0001                    IDT 'DATA'
0002               *PROGRAM TO INPUT DATA
0003               *CALLED DATA ON JAN3 DISC
0004 0000               RORG
0005               *
0006 0000 02E0          LWPI >F800         WP
     0002 F800
0007 0004 2FA0          XOP  @MESS9,14     NO OF TIMES CONT SEQUENCE
     0006 011C'
0008 0008 2E41     NULL9 XOP R1,9          INPUT VALUE INTO R1
0009 000A 0008'         DATA NULL9
0010 000C 0008'         DATA NULL9
0011 000E 2FA0          XOP  @MESS1,14     NO OF POS IN SEQUENCE
     0010 0092'
0012 0012 2E42     NULL1 XOP R2,9
0013 0014 0012'         DATA NULL1
0014 0016 0012'         DATA NULL1
0015 0018 C801          MOV  R1,@>FD16     NO OF TIMES CONT SEQUENCE
     001A FD16
0016 001C 0208          LI   R8,>FD20      BASE ADD FOR TABLE OF DELAYS
     001E FD20
0017 0020 020A          LI   R10,>FDA0     BASE ADD FOR WRIST OPEN/CLOSE
     0022 FDA0
0018 0024 0201          LI   R1,>FC20      BASE ADD FOR TABLE OF POS
     0026 FC20
0019 0028 C802          MOV  R2,@>FD12     SAVE NO OF POSITIONS
     002A FD12
0020 002C 2FA0     MES  XOP  @MESS2,14     POSITION OF AXIS ONE
     002E 00A8'
0021 0030 2E71     NULL2 XOP *R1+,9
0022 0032 0030'         DATA NULL2
0023 0034 0030'         DATA NULL2
0024 0036 2FA0          XOP  @MESS3,14     POSITION OF AXIS TWO
     0038 00C0'
0025 003A 2E71     NULL3 XOP *R1+,9
0026 003C 003A'         DATA NULL3
0027 003E 003A'         DATA NULL3
0028 0040 2FA0          XOP  @MESS4,14     POSITION OF AXIS THREE
     0042 00D8'
0029 0044 2E71     NULL4 XOP *R1+,9
0030 0046 0044'         DATA NULL4
0031 0048 0044'         DATA NULL4
0032 004A 2FA0     NULL5 XOP @MESS5,14     TIMES GO ROUND DELAY LOOP
     004C 00EF'
0033 004E 2E78          XOP  *R8+,9
0034 0050 004A'         DATA NULL5
0035 0052 004A'         DATA NULL5
0036 0054 2FA0     NULL6 XOP @MESS6,14     VALUES TO OPEN/CLOSE WRIST
     0056 00F8'
0037 0058 2E7A          XOP  *R10+,9
0038 005A 0054'         DATA NULL6
0039 005C 0054'         DATA NULL6
0040 005E 0602          DEC  R2            DEC.NO OF POS REMAINING
0041 0060 0282          CI   R2,0
```

```
     0062 0000
0042 0064 16E3         JNE  MES                IF NOT FIN.INPUT DATA
0043 0066 0460         B    @>80               ELSE RETURN TO THE MONITOR
     0068 0080
0044 006A 0340         IDLE
0045 006C   50  MESS   TEXT 'PROGRAM FINISHED RETURN TO MONITOR'
     006D   52
     006E   4F
     006F   47
     0070   52
     0071   41
     0072   4D
     0073   20
     0074   46
     0075   49
     0076   4E
     0077   49
     0078   53
     0079   48
     007A   45
     007B   44
     007C   20
     007D   52
     007E   45
     007F   54
     0080   55
     0081   52
     0082   4E
     0083   20
     0084   54
     0085   4F
     0086   20
     0087   4D
     0088   4F
     0089   4E
     008A   49
     008B   54
     008C   4F
     008D   52
0046 008E 0D0A         DATA >0D0A
0047 0090   00         BYTE 0
0048 0092 0D0A  MESS1  DATA >0D0A
0049 0094   4E         TEXT 'NO OF POSITIONS'
     0095   4F
     0096   20
     0097   4F
     0098   46
     0099   20
     009A   50
     009B   4F
     009C   53
     009D   49
     009E   54
     009F   49
```

```
     00A0   4F
     00A1   4E
     00A2   53
0050 00A4 0D0A         DATA >0D0A
0051 00A6   00         BYTE 0
0052 00A8 0D0A  MESS2  DATA >0D0A
0053 00AA   50         TEXT 'POSITION OF AXIS 1'
     00AB   4F
     00AC   53
     00AD   49
     00AE   54
     00AF   49
     00B0   4F
     00B1   4E
     00B2   20
     00B3   4F
     00B4   46
     00B5   20
     00B6   41
     00B7   58
     00B8   49
     00B9   53
     00BA   20
     00BB   31
0054 00BC 0D0A         DATA >0D0A
0055 00BE   00         BYTE 0
0056 00C0 0D0A  MESS3  DATA >0D0A
0057 00C2   50         TEXT 'POSITION OF AXIS 2'
     00C3   4F
     00C4   53
     00C5   49
     00C6   54
     00C7   49
     00C8   4F
     00C9   4E
     00CA   20
     00CB   4F
     00CC   46
     00CD   20
     00CE   41
     00CF   58
     00D0   49
     00D1   53
     00D2   20
     00D3   32
0058 00D4 0D0A         DATA >0D0A
0059 00D6   00         BYTE 0
0060 00D8 0D0A  MESS4  DATA >0D0A
0061 00DA   50         TEXT 'POSITION OF AXIS 3'
     00DB   4F
     00DC   53
     00DD   49
     00DE   54
     00DF   49
```

```
     00E0   4F
     00E1   4E
     00E2   20
     00E3   4F
     00E4   46
     00E5   20
     00E6   41
     00E7   58
     00E8   49
     00E9   53
     00EA   20
     00EB   33
0062 00EC 0D0A          DATA >0D0A
0063 00EE   00          BYTE 0
0064 00EF   44   MESS5  TEXT 'DELAY'
     00F0   45
     00F1   4C
     00F2   41
     00F3   59
0065 00F4 0D0A          DATA >0D0A
0066 00F6   00          BYTE 0
0067 00F8 0D0A   MESS6  DATA >0D0A
0068 00FA   49          TEXT 'INPUT 1 FOR OPEN & 2 FOR CLOSE'
     00FB   4E
     00FC   50
     00FD   55
     00FE   54
     00FF   20
     0100   31
     0101   20
     0102   46
     0103   4F
     0104   52
     0105   20
     0106   4F
     0107   50
     0108   45
     0109   4E
     010A   20
     010B   26
     010C   20
     010D   32
     010E   20
     010F   46
     0110   4F
     0111   52
     0112   20
     0113   43
     0114   4C
     0115   4F
     0116   53
     0117   45
0069 0118 0D0A          DATA >0D0A
0070 011A   00          BYTE 0
```

```
0071 011C 0D0A  MESS9  DATA >0D0A
0072 011E   48         TEXT 'HOW MANY TIMES IS THE SEQUENCE REPEATED?'
     011F   4F
     0120   57
     0121   20
     0122   4D
     0123   41
     0124   4E
     0125   59
     0126   20
     0127   54
     0128   49
     0129   4D
     012A   45
     012B   53
     012C   20
     012D   49
     012E   53
     012F   20
     0130   54
     0131   48
     0132   45
     0133   20
     0134   53
     0135   45
     0136   51
     0137   55
     0138   45
     0139   4E
     013A   43
     013B   45
     013C   20
     013D   52
     013E   45
     013F   50
     0140   45
     0141   41
     0142   54
     0143   45
     0144   44
     0145   3F
0073 0146 0D0A         DATA >0D0A
0074 0148 0000         DATA >0000
0075                   END
```

```
DATA           TXMIRA          2.3.0  78.244 00:16:57      01/01/00      PAGE 0006

 • MES       002C    • MESS      006C    • MESS1     0092    • MESS2     00AB
 • MESS3     00C0    • MESS4     00DB    • MESS5     00EF    • MESS6     00F8
 • MESS9     011C    • NULL1     0012    • NULL2     0030    • NULL3     003A
 • NULL4     0044    • NULL5     004A    • NULL6     0054    • NULL9     0008
   R0        0000      R1        0001      R10       000A      R11       000B
   R12       000C      R13       000D      R14       000E      R15       000F
   R2        0002      R3        0003      R4        0004      R5        0005
   R6        0006      R7        0007      R8        0008      R9        0009

 0000 ERRORS


 TXXREF          2.3.0  78.244     00:17:26     01/01/00     PAGE 0001

MES       0020        0042
MESS      0045
MESS1     0048        0011
MESS2     0052        0020
MESS3     0056        0024
MESS4     0060        0028
MESS5     0064        0032
MESS6     0067        0036
MESS9     0071        0007
NULL1     0012        0013   0014
NULL2     0021        0022   0023
NULL3     0025        0026   0027
NULL4     0029        0030   0031
NULL5     0032        0034   0035
NULL6     0036        0038   0039
NULL9     0008        0009   0010
R1                    0008   0015   0018   0021   0025   0029
R10                   0017   0037
R2                    0012   0019   0040   0041
R8                    0016   0033

THERE ARE 0020 SYMBOLS
```

```
0001                 IDT 'INT4'
0002          *PROGRAM TO MOVE 3 AXES
0003          *USING INTERRUPTS
0004          *TIME  DELAY
0005          *
0006          *
0007 F800            AORG >F800
0008 F800     SPACE  BSS 32
0009          *
0010      EFFE  ADCDAT EQU >EFFE
0011      EFFC  STATUS EQU >EFFC
0012      EFF2  DAC1   EQU >EFF2
0013      EFF0  DAC2   EQU >EFF0
0014      DEFE  DAC3   EQU >DEFE
0015      EFF8  MUXADR EQU >EFF8
0016      EFFA  CONV   EQU >EFFA
0017      EFF6  GAIN   EQU >EFF6
0018          *
0019 F820 02E0         LWPI SPACE
     F822 F800
0020          *CONSTANTS FOR VELOCITIES
0021 F824 0203         LI   R3,>180            MAX +VE VALUE
     F826 0180
0022 F828 C803         MOV  R3,@>FD0C
     F82A FD0C
0023 F82C 0203         LI   R3,>FE80           MAX -VE VALUE
     F82E FE80
0024 F830 C803         MOV  R3,@>FD0E
     F832 FD0E
0025 F834 0203         LI   R3,0
     F836 0000
0026 F838 C803         MOV  R3,@>FD10     ZERO VEOCITY
     F83A FD10
0027          *
0028 F83C 04C9         CLR  R9            REGISTER FOR MEM FOR DELAYS
0029 F83E 04E0         CLR  @>FD00        MEMLOC FOR ACTUAL POS
     F840 FD00
0030 F842 04E0         CLR  @>FD02
     F844 FD02
0031 F846 04E0         CLR  @>FD04
     F848 FD04
0032 F84A 04E0         CLR  @>FD06        MEMLOC. FOR REQUIRED POS
     F84C FD06
0033 F84E 04E0         CLR  @>FD08
     F850 FD08
0034 F852 04E0         CLR  @>FD0A
     F854 FD0A
0035          *
0036 F856 0208         LI   R8,>FD20      MEM.FOR START OF DELAYS
     F858 FD20
0037 F85A 0204         LI   R4,>FC20      MEM.FOR START OF POSITIONS
     F85C FC20
0038 F85E 0203         LI   R3,>FDD0      MEM FOR START OF WRIST YAW
     F860 FDD0
```

```
F862 020A        LI    R10,>FDA0        MEM FOR START OF GRIPPER
F864 FDA0
F866 020B        LI    R11,>FE00        MEM FOR START OF WRIST SWING
F868 FE00
F86A 0201        LI    R1,1             VAL FOR USE IN COMP FOR WRIST
F86C 0001
F86E 0205        LI    R5,>FD06         MEMLOC FOR REQ'D POS
F870 FD06
F872 CD74        MOV   *R4+,*R5+        MOVE REQ'D POS FROM FE00
F874 CD74        MOV   *R4+,*R5+        ONWARDS TO FD06,FD08,FD0A
F876 CD74        MOV   *R4+,*R5+
F878 020C        LI    R12,>100         BASE ADDRESS
F87A 0100
F87C 1E00        SBZ   0                INTERRUPT MODE
F87E 1D03        SBO   3                ENABLE INTERRUPT ON 9901
F880 0300        LIMI  3                ENABLE INTERRUPT ON 9900
F882 0003
F884 0200        LI    R0,3             COUNT=1 CLOCK MODE
F886 0003
F888 33C0        LDCR  R0,15            START COUNT
           *RETURN TO START IF NO DELAY
           *
F88A 1E00  START SBZ   0                INTERRUPT EVERY 38 MS
F88C 1D03        SBO   3
F88E 0300        LIMI  3
F890 0003
F892 0200        LI    R0,>300F
F894 300F
F896 33C0        LDCR  R0,15
F898 C1A0        MOV   @>FD08,R6        SAVE REQ'D POS
F89A FD08
F89C 61A0        S     @>FD02,R6        CHECK TO SEE IF IN POS
F89E FD02
F8A0 0746        ABS   R6               ABSOLUTE ERROR
F8A2 0286        CI    R6,>28           ALLOW FOR SLIGHT ERROR
F8A4 0028
F8A6 1101        JLT   NEX              IF NEARLY THERE NEXT AXIS
F8A8 1011        JMP   SELF             OTHERWISE AWAIT INTERRUPT
F8AA C0A0  NEX   MOV   @>FD06,R2        REPEAT FOR ALL OTHER AXES
F8AC FD06
F8AE 60A0        S     @>FD00,R2
F8B0 FD00
F8B2 0742        ABS   R2
F8B4 0282        CI    R2,>28
F8B6 0028
F8B8 1101        JLT   NEX2
F8BA 1008        JMP   SELF
F8BC C1E0  NEX2  MOV   @>FD0A,R7
F8BE FD0A
F8C0 61E0        S     @>FD04,R7
F8C2 FD04
F8C4 0747        ABS   R7
F8C6 0287        CI    R7,>28
F8C8 0028
```

```
0075 F8CA 1101          JLT   DEL              IF NEARLY THERE NEXT INST
0076 F8CC 10FF   SELF   JMP   SELF             OTHERWISE AWAIT INTERRUPT
0077             *CHECK TO SEE IF LAST POSITION
0078 F8CE 0620   DEL    DEC   @>FD12           DECREASE NO OF POS LEFT
     F8D0 FD12
0079 F8D2 C1A0          MOV   @>FD12,R6        SAVE
     F8D4 FD12
0080 F8D6 0286          CI    R6,0             SEE IF LAST POSITION
     F8D8 0000
0081 F8DA 132F          JEQ   STOP             IF LAST THEN STOP ROUTINE
0082 F8DC C278          MOV   *R8+,R9          OTHERWISE NEXT DELAY
0083             *OPEN/CLOSE GRIPPER
0084 F8DE 805A          C     *R10,R1          SEE IF OPEN OR CLOSE
0085 F8E0 1305          JEQ   OPEN             IF EQUAL TO ONE JMP TO OPEN
0086 F8E2 05CA          INCT  R10              INCREASE R10 BY TWO
0087 F8E4 020C          LI    R12,>120         CRU BASE ADDRESS
     F8E6 0120
0088 F8E8 1D0C          SBO   12               CLOSE JAWS
0089 F8EA 1004          JMP   YAW
0090 F8EC 05CA   OPEN   INCT  R10
0091 F8EE 020C          LI    R12,>120
     F8F0 0120
0092 F8F2 1E0C          SBZ   12               OPEN JAWS
0093 F8F4 8053   YAW    C     *R3,R1           SEE WHICH YAW MOVEMENT
0094 F8F6 1303          JEQ   UP
0095 F8F8 05C3          INCT  R3               INCREASE BY TWO R3
0096 F8FA 1D0D          SBO   13               MOVE YAW DOWN
0097 F8FC 1002          JMP   SWING
0098 F8FE 05C3   UP     INCT  R3
0099 F900 1E0D          SBZ   13               YAW UP
0100 F902 805B   SWING  C     *R11,R1          SWING MOVEMENT OF WRIST
0101 F904 1305          JEQ   ANTI
0102 F906 1D0E          SBO   14               CLOCKWISE MOVE OF WRIST
0103 F908 05CB          INCT  R11              INCREASE R11 BY TWO
0104 F90A 020C          LI    R12,>100         CRU AREA FOR INTERRUPTS
     F90C 0100
0105 F90E 1004          JMP   SELF2            AWAIT INTERRUPT
0106 F910 1E0E   ANTI   SBZ   14               ANTI-CLOCKWISE MOVEMENT OF WRI
0107 F912 05CB          INCT  R11
0108 F914 020C          LI    R12,>100
     F916 0100
0109 F918 10FF   SELF2  JMP   SELF2            AWAIT INTERRUPT
0110             *RETURN TO CONT FROM INTERRUPT IF THERE IS A DELAY
0111 F91A 1E00   CONT   SBZ   0                INTERRUPT EVERY 38mS
0112 F91C 1D03          SBO   3
0113 F91E 0300          LIMI  3
     F920 0003
0114 F922 0200          LI    R0,>300F
     F924 300F
0115 F926 33C0          LDCR  R0,15
0116 F928 0289          CI    R9,0             CHECK TO SEE IF DELAY FIN.
     F92A 0000
0117 F92C 1605          JNE   SELF3            IF NOT AWAIT INTERRUPT
0118 F92E 0205          LI    R5,>FD06         LOCATION FOR CMD
```

```
     F930 FD06
0119 F932 CD74        MOV  *R4+,*R5+         MOVE CMD TO >FD06, >FD08, >FD0A
0120 F934 CD74        MOV  *R4+,*R5+
0121 F936 CD74        MOV  *R4+,*R5+
0122 F938 10FF  SELF3 JMP  SELF3             AWAIT INTERRUPT
0123            *STOP ROUTINE
0124 F93A 0300  STOP  LIMI 0                 DISENABLE ALL INTERRUPTS
     F93C 0000
0125 F93E 2FA0        XOP  @MESS,14          PROGRAM FINISHED
     F940 F94B
0126 F942 0460        B    @>80              RETURN TO MONITOR
     F944 0080
0127 F946 0340        IDLE
0128 F948   50  MESS  TEXT 'PROGRAM FINISHED RETURN TO MONITOR'
     F949   52
     F94A   4F
     F94B   47
     F94C   52
     F94D   41
     F94E   4D
     F94F   20
     F950   46
     F951   49
     F952   4E
     F953   49
     F954   53
     F955   48
     F956   45
     F957   44
     F958   20
     F959   52
     F95A   45
     F95B   54
     F95C   55
     F95D   52
     F95E   4E
     F95F   20
     F960   54
     F961   4F
     F962   20
     F963   4D
     F964   4F
     F965   4E
     F966   49
     F967   54
     F968   4F
     F969   52
0129 F96A 0D0A        DATA >0D0A
0130 F96C   00        BYTE 0
0131
0132            *INTERRUPT ROUTINE
0133            *
0134 FB20             AORG >FB20             ORIGIN
0135 FB20 FC00        DATA >FC00             NEW WP
```

```
0136 FB22 FB24        DATA >FB24          NEW PC
0137 FB24 0300        LIMI 0              DISENABLE ALL INTERRUPTS
     FB26 0000
0138 FB28 0200        LI   R0,3           NO OF AXES
     FB2A 0003
0139 FB2C 0208        LI   R8,1           NO.TO SELECT ADC CHANNEL
     FB2E 0001
0140 FB30 0209        LI   R9,2           NO .TO SELECT ADC CHANNEL
     FB32 0002
0141 FB34 04E0        CLR  @GAIN          GAIN = 1
     FB36 EFF6
0142             *
0143 FB38 04E0  AXIS1 CLR  @MUXADR        CHANNEL 0 ON ADC
     FB3A EFF8
0144 FB3C 020A        LI   R10,0          DISPLACEMENT FOR DAC
     FB3E 0000
0145 FB40 100A        JMP  CON
0146 FB42 C808  AXIS2 MOV  R8,@MUXADR     CHANNEL 1 ON ADC
     FB44 EFF8
0147 FB46 020A        LI   R10,2          DISPLACEMENT FOR DAC
     FB48 0002
0148 FB4A 1009        JMP  CONVRT
0149 FB4C C809  AXIS3 MOV  R9,@MUXADR     CHANNEL 2 ON ADC
     FB4E EFF8
0150 FB50 020A        LI   R10,>EF0E      DISPLACRMENT FOR DAC
     FB52 EF0E
0151 FB54 1004        JMP  CONVRT
0152 FB56 0203  CON   LI   R3,>FD00       MEMLOC FOR ACTUAL POS
     FB58 FD00
0153 FB5A 0205        LI   R5,>FD06       MEMLOC FOR REQ'D POS
     FB5C FD06
0154             *
0155 FB5E 0720  CONVRT SETO @CONV         START CONVERSION
     FB60 EFFA
0156 FB62 0560  CHK   INV  @STATUS        CHECK DATA READY
     FB64 EFFC
0157 FB66 11FD        JLT  CHK
0158 FB68 C060        MOV  @ADCDAT,R1     ACTUAL POS IN R1
     FB6A EFFE
0159 FB6C CCC1        MOV  R1,*R3+        MOVE TO MEMLOC
0160 FB6E 6075        S    *R5+,R1        ACTUAL-CMD
0161 FB70 C081        MOV  R1,R2          SAVE
0162 FB72 0742        ABS  R2             ABSOLUTE ERROR
0163 FB74 0282        CI   R2,>28         SEE IF NEARLY IN POS
     FB76 0028
0164 FB78 1111        JLT  LAB3
0165 FB7A 0281        CI   R1,>180        SEE IF ERROR > MAX + VALUE
     FB7C 0180
0166 FB7E 1104        JLT  LAB1
0167 FB80 CAA0        MOV  @>FD0C,@DAC2(R10) OUTPUT +1.8V
     FB82 FD0C
     FB84 EFF0
0168 FB86 100D        JMP  TEST           SEE IF ANY MORE AXES
0169 FB88 0281  LAB1  CI   R1,>FE80       SEE IF ERROR < MAX -VE VALUE
```

```
     FB8A FE80
0170 FB8C 1504         JGT  LAB2
0171 FB8E CAA0         MOV  @>FD0E,@DAC2(R10) OUTPUT-1.8V
     FB90 FD0E
     FB92 EFF0
0172 FB94 1006         JMP  TEST
0173 FB96 CA81  LAB2   MOV  R1,@DAC2(R10)     OUTPUT ACTUAL VELOCITY
     FB98 EFF0
0174 FB9A 1003         JMP  TEST
0175 FB9C CAA0  LAB3   MOV  @>FD10,@DAC2(R10) OUTPUT 0V
     FB9E FD10
     FBA0 EFF0
0176 FBA2 0600  TEST   DEC  R0                DEC COUNT FOR NO OF AXES
0177 FBA4 0280         CI   R0,2
     FBA6 0002
0178 FBA8 13CC         JEQ  AXIS2             SERVICE AXIS TWO
0179 FBAA 0280         CI   R0,1
     FBAC 0001
0180 FBAE 13CE         JEQ  AXIS3             SERVICE AXIS THREE
0181 FBB0 C1A0         MOV  @>F812,R6         SAVE DELAY COUNT FROM OLD R9
     FBB2 F812
0182 FBB4 0286         CI   R6,0
     FBB6 0000
0183 FBB8 1606         JNE  CONTD             IF <> 0 GO TO CONTD ROUTINE
0184 FBBA 020D         LI   R13,>F800         WP FOR MAIN PROGRAM
     FBBC F800
0185 FBBE 020E         LI   R14,START         PC FOR RETURN
     FBC0 F88A
0186 FBC2 04CF         CLR  R15               CLEAR STATUS
0187 FBC4 0380         RTWP                   RETURN TO MAIN PROGRAM
0188            *
0189 FBC6 0620  CONTD  DEC  @>F812            DEC COUNT IN OLD R9,DELAY
     FBC8 F812
0190 FBCA 020D         LI   R13,>F800         WP OF MAIN PROGRAM
     FBCC F800
0191 FBCE 020E         LI   R14,CONT          PC FOR RETURN
     FBD0 F91A
0192 FBD2 04CF         CLR  R15               CLEAR STATUS
0193 FBD4 0380         RTWP                   RETURN TO MAIN PROGRAM
0194            *
0195            *
0196            *
0197            *
0198            *TSR MEM SPACE
0199            *
0200 FFAA              AORG >FFAA
0201 FFAA 0420         BLWP @>FB20            GO TO >FB20 FOR NEW PC&WP
     FFAC FB20
0202 FFAE 0380         RTWP
0203                   END
```

ADCDAT	EFFE	ANTI	F910	AXIS1	FB38	AXIS2	FB42
AXIS3	FB4C	CHK	FB62	CON	FB56	CONT	F91A
CONTD	FBC6	CONV	EFFA	CONVRT	FB5E	DAC1	EFF2
DAC2	EFF0	DAC3	DEFE	DEL	F8CE	GAIN	EFF6
LAB1	FB88	LAB2	FB96	LAB3	FB9C	MESS	F94B
MUXADR	EFFB	NEX	FBAA	NEX2	FBBC	OPEN	FBEC
R0	0000	R1	0001	R10	000A	R11	000B
R12	000C	R13	000D	R14	000E	R15	000F
R2	0002	R3	0003	R4	0004	R5	0005
R6	0006	R7	0007	R8	0008	R9	0009
SELF	F8CC	SELF2	F918	SELF3	F938	SPACE	F800
START	F88A	STATUS	EFFC	STOP	F93A	SWING	F902
TEST	FBA2	UP	FBFE	YAW	FBF4		

0000 ERRORS

```
ADCDAT  0010     0158
ANTI    0106     0101
AXIS1   0143
AXIS2   0146     0178
AXIS3   0149     0180
CHK     0156     0157
CON     0152     0145
CONT    0111     0191
CONTD   0189     0183
CONV    0016     0155
CONVRT  0155     0148  0151
DAC1    0012
DAC2    0013     0167  0171  0173  0175
DAC3    0014
DEL     0078     0075
GAIN    0017     0141
LAB1    0169     0166
LAB2    0173     0170
LAB3    0175     0164
MESS    0128     0125
MUXADR  0015     0143  0146  0149
NEX     0065     0063
NEX2    0071     0069
OPEN    0090     0005
R0               0050  0051  0057  0058  0114  0115  0138  0176  0177
                 0179
R1               0041  0084  0093  0100  0158  0159  0160  0161  0165
                 0169  0173
R10              0039  0084  0086  0090  0144  0147  0150  0167  0171
                 0173  0175
R11              0040  0100  0103  0107
R12              0046  0087  0091  0104  0108
R13              0184  0190
R14              0185  0191
R15              0186  0192
R2               0065  0066  0067  0068  0161  0162  0163
R3               0021  0022  0023  0024  0025  0026  0038  0093  0095
                 0098  0152  0159
R4               0037  0043  0044  0045  0119  0120  0121
R5               0042  0043  0044  0045  0118  0119  0120  0121  0153
                 0160
R6               0059  0060  0061  0062  0079  0080  0181  0182
R7               0071  0072  0073  0074
R8               0036  0082  0139  0146
R9               0028  0082  0116  0140  0149
SELF    0076     0064  0070  0076
SELF2   0109     0105  0109
SELF3   0122     0117  0122
SPACE   0008     0019
START   0054     0185
STATUS  0011     0156
STOP    0124     0031
SWING   0100     0097
TEST    0176     0168  0172  0174
UP      0098     0094
YAW     0093     0089

THERE ARE 0051 SYMBOLS
```

```
0 PROGRAM DATA_INPUT;(*$DEBUG*)(*JM6 ON DISC*)
0
0    CONST
0      MAX_NO=5;
0      MAX_POSITIONS=20;
0    TYPE
0      REC1=RECORD
0      NAME_AXIS:PACKED ARRAY[1..10]OF CHAR;
0      AXIS_TYPE:PACKED ARRAY [1..10]OF CHAR;
0      MAX_TRAVEL:INTEGER;
0      MAX_VEL:INTEGER;
0      FEEDBACK:PACKED ARRAY [1..10] OF CHAR;
0      POSIT:PACKED ARRAY [1..10] OF CHAR;
0      END;
0       REC2=RECORD
0      STORE:ARRAY[1..20]OF INTEGER;
0      END;
0      TOT=ARRAY[1..MAX_NO]OF REC1;
0      TOT2=ARRAY[1..MAX_NO]OF REC2;
0   VAR
0      POINT:ARRAY[1..20]OF INTEGER;
40     TOTAL:TOT;
260    TOTAL2:TOT2;
460    NO_OF_EDITS,X,NO_OF_POS,NUM,AXIS,COUNT,NO_OF_AXES:INTEGER;
474    LINS,INSTS,INST,NO_OF_INSERTS,NO_OF_REMOVES,NOS,LNOS:INTEGER;
488    REMOVE,ALT,INSERT,EDIT:CHAR;
496    ALTER:ARRAY[1..5]OF INTEGER;
506    REM:ARRAY[1..5]OF INTEGER;
516    INS:ARRAY[1..5]OF INTEGER;
  1 BEGIN(*DETAILS OF AXES*)
  1    COUNT:=0;
  2    RESET(INPUT);
  3    REWRITE(OUTPUT);
  4    WRITELN(OUTPUT,'HOW MANY AXES ARE THERE?');
  5    READLN(INPUT,NO_OF_AXES);
  6    WHILE COUNT < NO_OF_AXES DO
  7    BEGIN
  7      COUNT:=COUNT+1;
  8      WRITELN(OUTPUT,'WHAT IS THE NAME OF THE AXIS?');
  9      READLN(INPUT,TOTAL[COUNT].NAME_AXIS);
 10    END;
 10    COUNT:=0;
 11    WHILE COUNT < NO_OF_AXES DO
 12      BEGIN (*TYPE OF AXIS*)
 12        COUNT:=COUNT+1;
 13        WRITELN(OUTPUT,' TYPE OF AXIS FOR  ',TOTAL[COUNT].NAME_AXIS);
 14        READLN(INPUT,TOTAL[COUNT].AXIS_TYPE);
 15      END;
 15      COUNT:=0;
 16      WHILE COUNT < NO_OF_AXES DO
 17      BEGIN
 17        COUNT:=COUNT+1;
 18        WRITELN('WHAT IS THE MAX VELOCITY FOR   ',TOTAL[COUNT].NAME_AXIS);
 19        READLN(INPUT,TOTAL[COUNT].MAX_VEL);
 20      END;
 20      COUNT:=0;
 21      WHILE COUNT < NO_OF_AXES DO
 22        BEGIN(*FEEDBACK*)
 22          COUNT:=COUNT+1;
 23          WRITELN('WHAT IS THE FEEDBACK FOR  ',TOTAL[COUNT].NAME_AXIS);
 24          READLN(INPUT,TOTAL[COUNT].FEEDBACK);
 25        END;(*FEEDBACK*)
 25      COUNT:=0;
 26      WHILE COUNT < NO_OF_AXES DO
 27        BEGIN(*1*)
 27          COUNT:=COUNT+1;
 28          WRITELN('HOW MANY POSITIONS HAS   ',TOTAL[COUNT].NAME_AXIS);
 29          READLN(INPUT,TOTAL[COUNT].POSIT);
 30        END;(*1*)
 30      COUNT:=0;
 31      WHILE COUNT < NO_OF_AXES DO
 32        BEGIN(*2*)
```

```
         COUNT:=COUNT+1;
         WRITELN('WHAT IS THE MAX VALUE FOR POS ',TOTAL[COUNT].NAME_AXIS);
         READLN(INPUT,TOTAL[COUNT] MAX_TRAVEL);
       END;(*2*)
     BEGIN(*INPUT POSITIONS*)
       WRITELN('HOW MANY POSITIONS ARE THERE?');
       READLN(INPUT,NO_OF_POS);
       FOR NUM.=1 TO NO_OF_POS DO
         BEGIN(*1*)
           COUNT:=0;
           WHILE COUNT < NO_OF_AXES DO
             BEGIN(*2*)
               COUNT:=COUNT+1;
               WRITELN('POSITION FOR  ',TOTAL[COUNT].NAME_AXIS);
               READLN(INPUT,TOTAL2[COUNT].STORE[NUM]);
             END;(*2*)
           WRITELN('NEXT INSTRUCTION NUMBER');
           READLN(POINT[NUM]);
         END;(*1*)
     END;(*INPUT POSITIONS*)
BEGIN(*PRINT LIST*)
  WRITELN;
  WRITELN;
  FOR COUNT:=1 TO NO_OF_AXES DO
    BEGIN(*1*)
    WRITELN(TOTAL[COUNT].NAME_AXIS);
    WRITELN;
    WRITELN('THE TYPE OF AXIS IS                 ',TOTAL[COUNT].AXIS_TYPE);
    WRITELN;
    WRITELN('THE MAX VALUE FOR POSITION IS',TOTAL[COUNT].MAX_TRAVEL);
    WRITELN;
    WRITELN('THE MAX VELOCITY IS               ',TOTAL[COUNT].MAX_VEL);
    WRITELN;
    WRITELN('THE METHOD OF FEEDBACK IS           ',TOTAL[COUNT].FEEDBACK);
    WRITELN;
    WRITELN('NO OF POSITIONS ON THE AXIS          ',TOTAL[COUNT].POSIT);
    WRITELN;
    WRITELN;
    END;(*1*)
    WRITE('INSTRUCTION NO       ');
    COUNT:=0;
    WHILE COUNT < NO_OF_AXES DO
     BEGIN(*HEADINGS*)
       COUNT:=COUNT+1;
       WRITE(TOTAL[COUNT].NAME_AXIS);
     END;(*HEADINGS*)
     WRITE('POINTER');
     WRITELN;
    FOR NUM:=1 TO NO_OF_POS DO
      BEGIN(*2*)
      COUNT:=0;
      WRITE(NUM);
      WHILE COUNT < NO_OF_AXES DO
        BEGIN(*3*)
          COUNT:=COUNT+1;
          WRITE('    ',TOTAL2[COUNT].STORE[NUM]);
        END;(*3*)
        WRITE(' ',POINT[NUM]);
        WRITELN
      END;(*2*)


END;(*PRINT LIST*)
WRITELN('DOES THE SEQUENCE REQUIRE EDITING INPUT Y FOR YES & N FOR NO');
READLN(INPUT,EDIT);
   CASE EDIT OF
   'Y':BEGIN(*EDIT*)
          WRITELN('DOES THE VALUES FOR THE POSITIONS REQUIRE ALTERING');
          WRITELN('INPUT Y FOR YES & N FOR NO');
          READLN(ALT);
          CASE ALT OF
          'Y':BEGIN(*INST*)
            NOS:=0;
            WRITELN('HOW MANY INSTRUCTIONS REQUIRE ALTERING?');
            READLN(NO_OF_EDITS);
```

```
WRITELN('WHICH INSTRUCTIONS REQUIRE ALTERING?');
WHILE NOS <NO_OF_EDITS DO
 BEGIN(*1*)
   NOS:=NOS+1;
   READLN(ALTER[NOS]);
 END;(*1*)
LNOS:=NOS;
NOS:=0;
WHILE NOS < LNOS DO
 BEGIN(*2*)
   NOS:=NOS+1;
   WRITELN(ALTER[NOS],'      INPUT THE CORRECT VALUES');
   NUM:=NOS;
   COUNT:=0;
   WHILE COUNT ‹ NO_OF_AXES DO
     BEGIN(*3*)
       COUNT:=COUNT+1;
       WRITELN('POSITION FOR   ',TOTAL[COUNT].NAME_AXIS);
        READLN(TOTAL2[COUNT].STORE[NUM]);
      END;(*3*)
   END;(*2*)
  END;(*INST*)
 'N':WRITELN('NO ALTERATIONS TO VALUES REQUIRED');
 END;(*CASE*)
 WRITELN('ARE ANY INSTRUCTIONS TO BE REMOVED?');
 READLN(REMOVE);
 CASE REMOVE OF
 'Y':BEGIN(*REMOVE*)
    WRITELN('HOW MANY INSTRUCTIONS TO BE REMOVED?');
    READLN(NO_OF_REMOVES);
    NOS:=0;
    WRITELN('WHICH INSTRUCTIONS REQUIRE REMOVING?');
    WHILE NOS < NO_OF_REMOVES DO
      BEGIN(*REMOVE*)
      NOS:=NOS+1;
      READLN(REM[NOS]);
      END;(*REMOVE*)
      LNOS:=NOS;
      NOS:=0;
      WHILE NOS < LNOS DO
        BEGIN(*CH*) of pointer +)
          NOS:=NOS+1;
          INST:=REM[NOS];
          INST:=INST-1;
          WRITELN('CORRECT POINTER FOR INST NO ',INST);
          READLN(POINT[INST]);
        END;(*CH*)
     END;(*REMOVE*)
   'N':WRITELN('NO LINES TO BE REMOVED');
   END;(*CASE 2*)
 WRITELN('ANY INSTRUCTIONS TO BE INSERTED');
 READLN(INSERT);
 CASE INSERT OF
 'Y':BEGIN(*INSERT*)
 NOS:=0;
 WRITELN('HOW MANY INSTRUCTIONS TO BE INSERTED?');
 READLN(NO_OF_INSERTS);
 WRITELN('WHICH INST ARE TO HAVE VALUES AFTER THEM?');
 WHILE NOS < NO_OF_INSERTS DO
  BEGIN(*INS*)
    NOS:=NOS+1;
    READLN(INS[NOS]);
  END;(*INS*)
LNOS:=NOS;
NOS:=0;
WHILE NOS < LNOS DO
  BEGIN(*INSER*)
    NOS:=NOS+1;
    NUM:=INS[NOS];
    WRITELN('NEW POINTER FOR ',INS[NOS]);
    READLN(POINT[NUM]);
    WRITELN('INSERT INSTR WHICH',POINT[NUM],'     POINTS TO');
    INSTS:=NO_OF_POS;
     BEGIN(*I*)
       COUNT:=0;
```

```
                    INSTS:=INSTS+1;
                    WHILE COUNT < NO_OF_AXES DO
                      BEGIN(*IN*)
                        COUNT:=COUNT+1;
                        WRITELN('POS FOR ',TOTAL[COUNT].NAME_AXIS);
                        READLN(TOTAL2[COUNT].STORE[INSTS]);
                      END,(*IN*)
                   END;(*I*)
                 LINS:=INSTS;
               END;(*INSCR*)
              END;(*INSERT*)
              'N':WRITELN('NO INSERTS' );
            END;(*CASE3*)
            END;(*EDIT*)
        'N':WRITELN('YOUR PROGRAM IS CORRECT' );
      END,(*CASE*)
    WRITE('INSTRUCTION NO          ' );
    COUNT:=0;
    WHILE COUNT < NO_OF_AXES DO
      BEGIN(*HEADINGS*)
        COUNT:=COUNT+1;
        WRITE(TOTAL[COUNT] NAME_AXIS);
      END;(*HEADINGS*)
      WRITE('POINTER' );
      WRITELN;
    FOR NUM:=1 TO LINS DO
     BEGIN(*PRINT NEW LIST*)
       COUNT:=0;
       WRITE(NUM);
       WHILE COUNT < NO_OF_AXES DO
         BEGIN(*1*)
           COUNT:=COUNT+1;
           WRITE('     ',TOTAL2[COUNT].STORE[NUM]);
         END;(*1*)
       WRITE('     ',POINT[NUM]);
       WRITELN
     END;(*PRINT NEW LIST*)
END.
```

APPENDIX 8

SECTION 1

INTRODUCTION

1.1 GENERAL

The Texas Instruments TM 990/101M is a self-contained microcomputer on a singl printed-circuit board. The board's component side is shown in Figure 1-1, which als highlights major features and components. Figure 1-2 shows board dimensions. Thi microcomputer board contains features found on computer systems of much larger size including a central processing unit (CPU) with hardware multiply and divide, programmable serial and parallel I/O lines, external interrupts, and a debug-monito to assist the programmer in program development and execution. Other features include:

- TMS 9900 microprocessor based system: the microprocessor with the minicomputer instruction set - software compatible with other members of the 99 family.

- 1K x 16 bits of TMS 4045 random-access memory (RAM) expandable on-board t 2K x 16 bits.

- 1K x 16 bits of TMS 2708 erasable programmable read-only memory (EPROM), expandable on-board to 2K x 16 bits. Simple jumper modifications enable substitution of the larger TMS 2716 EPROM's (16K bits each) for the smaller TMS 2708's (8K bits each). Four TMS 2716's permit EPROM expansion to 4K x 16 bits.

NOTE

Three board configurations are available. The characteristics of each configuration are explained in paragraph 1.3.

- Buffered address, data, and control lines for off-board memory and I/O expansion; full DMA capabilities are provided by the buffer controllers.

- 3 MHz crystal-controlled clock.

- One 16-bit parallel I/O port, each bit is individually programmable.

- Modified EIA RS-232-C serial I/O interface, capable of communication to both EIA-compatible terminals and popular modems (data sets).

- A local serial I/O port, with interfaces for an EIA terminal and either a Teletype (TTY) or a twisted-pair balanced-line multidrop system (interface choices are detailed in paragraph 1.3).

- Three programmable interval timers.

- 17 prioritized interrupts, including RESET and LOAD functions. Interrupt 6 is level triggered (active LOW) and edge-triggered (either polarity) and latched on-board.

- A directly addressable five-position DIP switch and an addressable light emitting diode (LED) for custom system applications.

- PROM memory decoder permits easy reassignment of memory map configuration; see Figure 1-3 for memory map of the standard board.

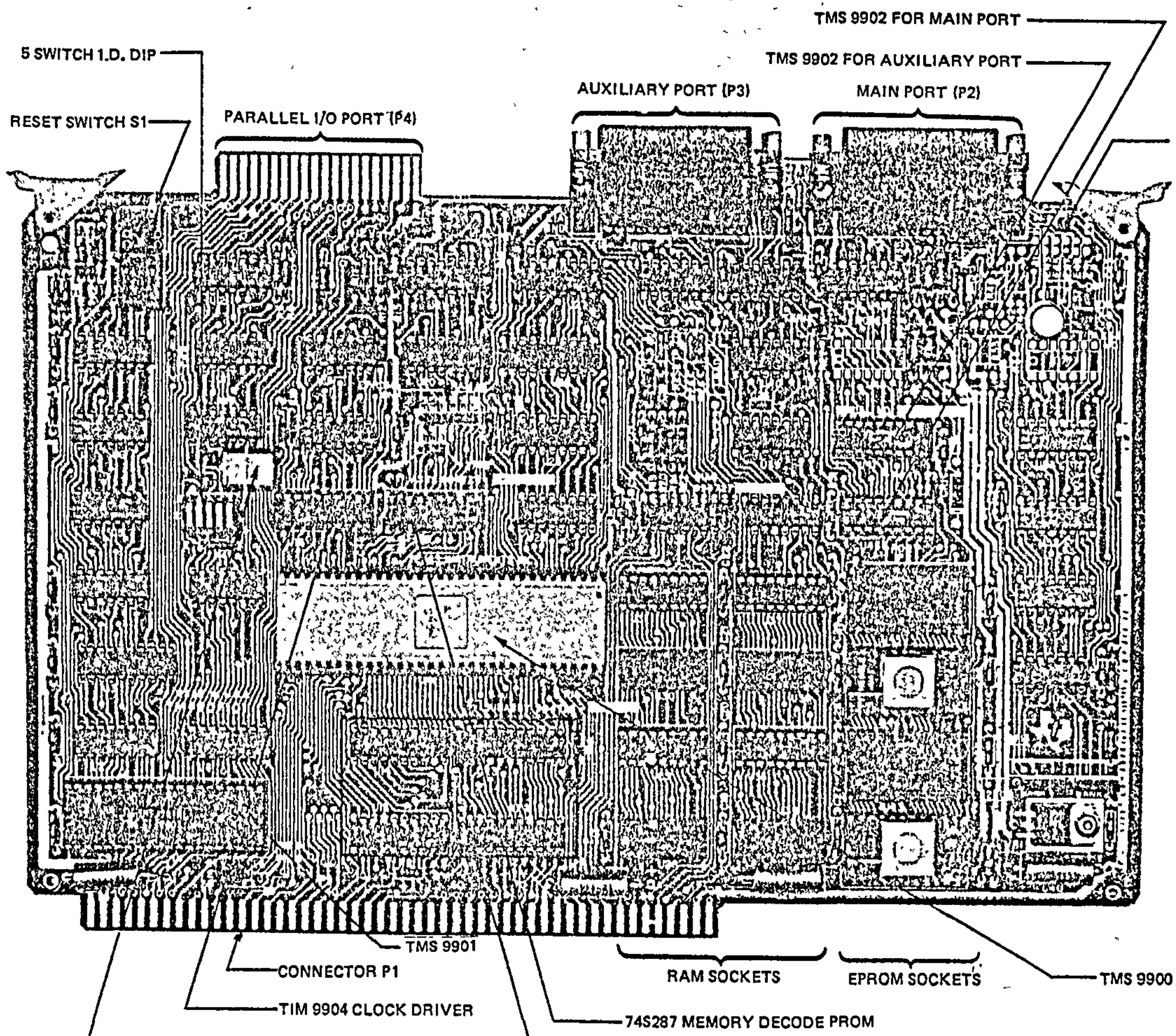


Figure 1-1. TM 990/101M Major Components

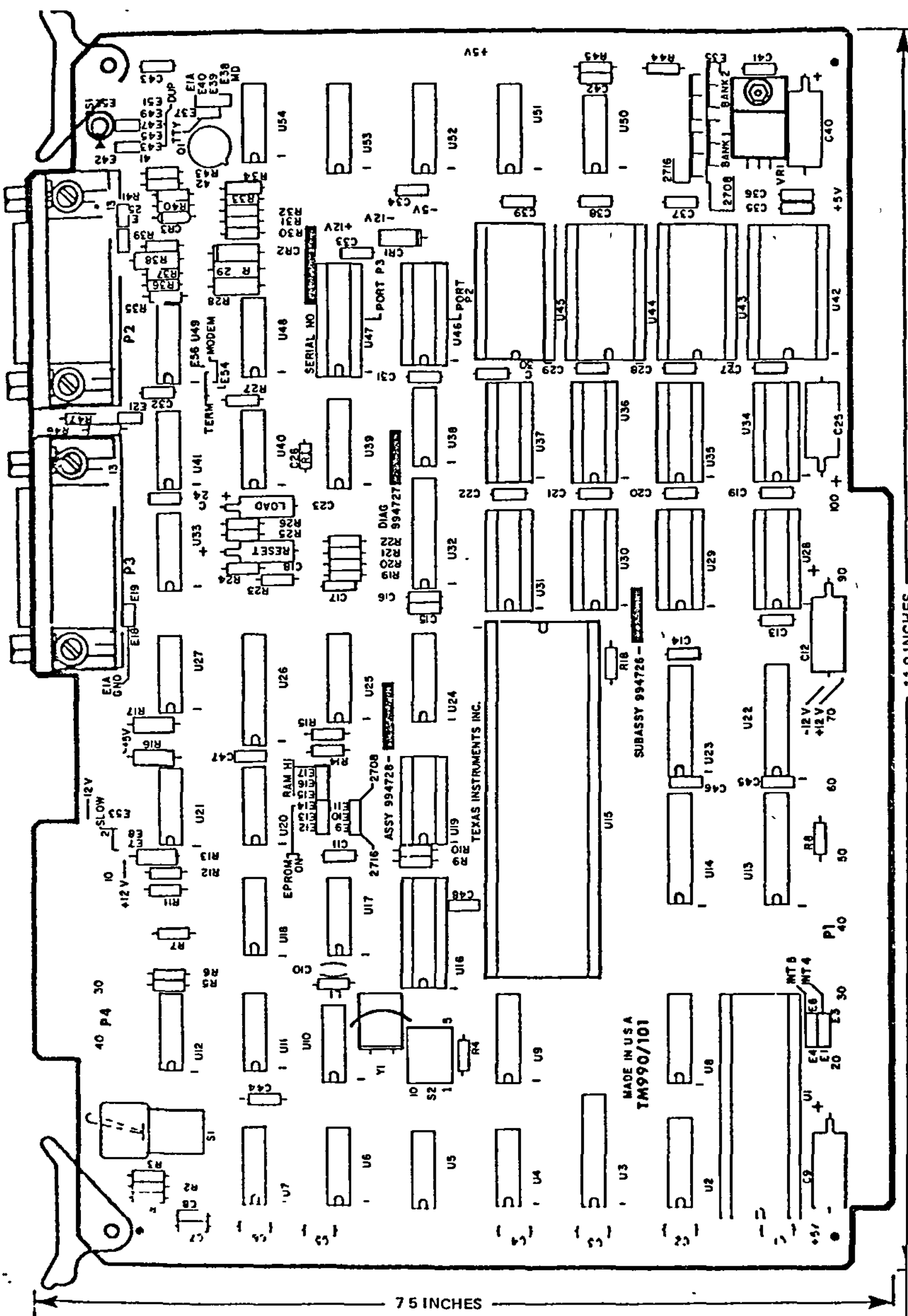


Figure 1-2. TM 990/101M Dimensions and Component Placement

1.2 MANUAL ORGANIZATION

Section 1 covers board specifications and characteristics. A glossary in paragraph 1. explains terms used throughout the manual.

Section 2 explains how to install, power-up, and operate the TM 990/101 microcompute with the addition of a data terminal, power supplies, and appropriate connectors.

Section 3 explains how to communicate with the TM 990/101M using the TIBUG monitor This versatile monitor, complete with supervisor calls and operator communicatio commands, facilitates the development and execution of software.

Section 4 describes the instruction set of the TM 990/101M, giving examples of eac class of instructions and providing some explanation of the TMS 9900 architecture.

Section 5 explains basic programming procedures for the microcomputer, giving a explanation of the programming environment and hardware-dependent features. Numerou program examples are included for utilizing the various facilities of the TM 990/101M

Section 6 is a basic theory of operation, explaining the hardware design configuratio and circuitry. This section provides explanations of the bus structure, the contro logic, and the various subsystems which make up the microcomputer.

Section 7 describes various options available for the microcomputer, both thos supplied on-board and those which Texas Instruments offers for off-board expansion o the system.

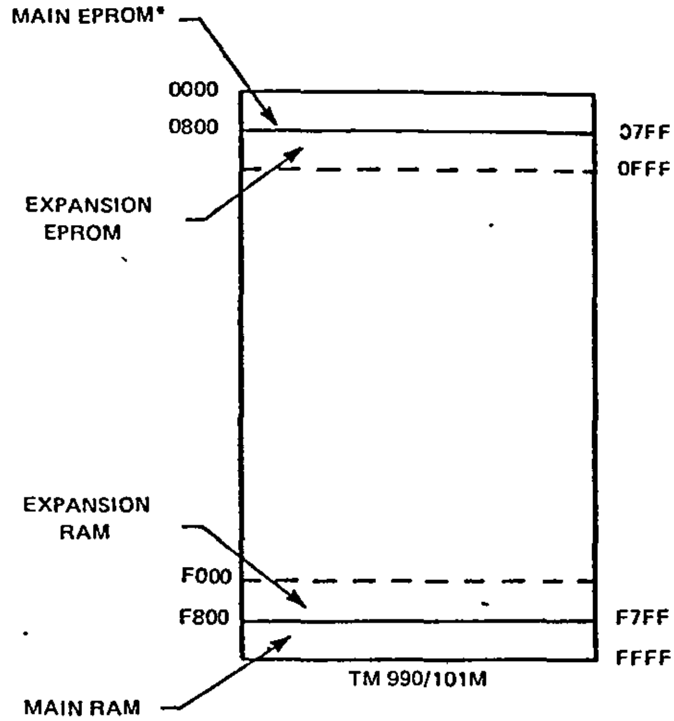
Section 8 features various hardware applications which can be built using the T 990/101M.

1.3 PRODUCT INDEX

The TM 990/101M microcomputer is available in three different configurations, whic are specified by a "dash number" appended to the product name; e.g., TM 990/101M-1 These configurations are listed in Table 1-1. A memory map is shown in Figure 1-3.

Table 1-1. TM 990/101M Configurations

TM 990/101M Dash No.	EPROM Socketed	EPROM Program	RAM	Main Serial Por Option (EIA Terminal I/F Stand)
-1	2 TMS 2708 (1K x 16)	TIBUG Monitor	4 TMS 4045 (1K x 16)	TTY
-2	2 TMS 2716 (2K x 16)	Blank	4 TMS 4045 (1K x 16)	Multidrop
-3	4 TMS 2716 (4K x 16)	Blank	8 TMS 4045 (2K x 16)	TTY



*EPROM's programmed with TIBUG monitor

Figure 1-3. Main And Expansion EPROM and RAM

1.4 BOARD CHARACTERISTICS

Figure 1-1 shows the major portions and components of the microcomputer. The systen bus connector is P1, which is a 100-pin (50 each side) PC board edge connector spacec on 0.125 inch centers. Connector P2 is the main serial port and P3 is the RS-232-(auxiliary serial port. Both connectors are standard 25-position female jacks used ir RS-232-C communications. The parallel I/O port is PC board edge connector P4, whicl has 40 pins (20 each side) spaced on 0.1-inch centers.

Figure 1-2 shows the PC board silkscreen markings which detail the various component: on the board; also included are the board dimensions and tolerances.

1.5 GENERAL SPECIFICATIONS

	+5 V		+12 V		-12 V	
Power Consumption	TYP	MAX	TYP	MAX	TYP	MAX
TM 990/101M-1	1.8	2.6	0.30	0.50	0.25	0.40
TM 990/101M-2	1.8	2.6	0.30	0.50	0.25	0.40

Clock Rate: 3 MHz

Baud Rates (set by TIBUG): 110, 300, 600, 1200, 2400, 4800, 9600, 19200

Memory Size: The microcomputer is shipped with:
RAM: Four TMS 4045 (1K x 4 bits each)
EPROM: Two TMS 2708 (1K x 8 bits each), preprogrammed with TIBUG.

Total capacity is:
RAM: Eight TMS 4045's (1K x 4 bits each)
EPROM: Four TMS 2708's (1K x 8 bits each)
or
Four TMS 2716's (2K x 8 bits each)

Board Dimensions: See Figure 1-2

Parallel I/O Port (P4): One 16-bit port, uses TMS 9901 programmable systems interface

Serial I/O Port (P2 and P3): Two asynchronous ports:

Main port (P2) has two interfaces: RS-232-C answer mode and either a TTY or a balanced-line differential multidrop interface.

Auxiliary port (P3) meets RS-232-C specification interface, capable of either originate or answer mode.

Both serial ports use TMS 9902 asynchronous communication controllers, but th Auxiliary Port will readily accept the TMS 9903 synchronous communication controller. Simply plug in the TMS 9903 for synchronous systems.

1.6 REFERENCE DOCUMENTS

The following documents provide supplementary information for the TM 990/101M user's manual.

- TMS 9900 Microprocessor Data Manual
- TMS 9901 Programmable Systems Interface Data Manual
- TMS 9902 Asynchronous Communication Controller Data Manual
- TMS 9903 Synchronous Communication Controller Data Manual
- TMS 990 Computer, TMS 9900 Microprocessor Assembly Language Programmer's Guide (P/N 943441-9701)
- TM 990/301 Microterminal
- TM 990/401 TIBUG Monitor Listing
- TM 990/402 Line-by-Line Assembler User's Guide
- TM 990/402L Line-by-Line Assembler Listing
- TM 990/502 Cable Assembly (RS-232-C)
- TM 990/503 Cable Assembly (TI Terminal 743 or 745)
- TM 990/504 Cable Assembly (Teletype)
- TM 990/506 Cable Assembly (Modem cable for /101 board)
- TM 990/510 Card Chassis
- TM 990/511 Extender Board User's Guide
- TM 990/512 Prototyping Board User's Guide

1.7 GLOSSARY

The following are definitions of terms used with the TM 990/101M. Applicable areas in this manual are in parentheses.

Absolute Address: The actual memory address in quantity of bytes. Memory addressing is usually represented in hexadecimal from 0000_{16} to $FFFF_{16}$ for the TM 990/101M.

Alphanumeric Character: Letters, numbers, and associated symbols.

ASCII Code: A seven-bit code used to represent alphanumeric characters and control (Appendix C).

Assembler: Program that translates assembly language source statements into object code.

Assembly Language: Mnemonics which can be interpreted by an assembler and translated into an object program (paragraph 4.6).

Bit: The smallest part of a word; it has a value of either a 1 or 0.

Breakpoint: Memory address where a program is intentionally halted. This is a program debugging tool.

Byte: Eight bits or half a word.

Carry: A carry occurs when the most-significant bit is carried out in an arithmetic operation (i.e., result cannot be contained in only 16 bits), (paragraph 4.3.3.4).

Central Processing Unit (CPU): The "heart" of the computer: responsibilities include instruction access and interpretation, arithmetic functions, I/O memory access. The TMS 9900 is the CPU of the TM 990/101M.

Chad: Dot-like paper particles resulting from the punching of paper tape.

Command Scanner: A given set of instructions in the TIBUG monitor which takes the user's input from the terminal and searches a table for the proper code to execute.

Context Switch: Change in program execution environment, includes new program counter (PC) value and new workspace area.

CRU (Communications Register Unit): The TMS 9900's general purpose, command-driven input/output interface. The CRU provides up to 4096 directly addressable input and output bits (paragraph 4.8).

Effective Address: Memory address value resulting from interpretation of an instruction operand, required for execution of that instruction.

EPROM: See Read Only Memory.

Hexadecimal: Numerical notation in the base 16 (Appendix D).

Immediate Addressing: An immediate or absolute value (16-bits) is part of the instruction (second word of instruction).

Indexed Addressing: The effective address is the sum of the contents of an index register and an absolute (or symbolic) address (paragraph 4.5.3.5).

Indirect Addressing: The effective address is the contents of a register (paragraph 4.5.3.2).

Interrupt: Context switch in which new workspace pointer (WP) and program counter (PC) values are obtained from one of 16 interrupt traps in memory addresses 0000_{16} to $003E_{16}$ (paragraph 4.9).

I/O: The input/output lines are the signals which connect an external device to the data lines of the TMS 9990.

Least Significant Bit (LSB): Bit having the smallest value (samllest power of base 2); represented by the right-most bit.

Link: The process by which two or more object code modules are combined into one, with cross-referenced label address locations being resolved.

Load: Transfer control to operating system using the equivalent of a BLWP instructio to vectors in upper memory ($FFFC_{16}$ and $FFFE_{16}$). See Reset.

Loader: Program that places one or more absolute or relocatable object programs into memory (Appendix G).

Machine Language: Binary code that can be interpreted by the CPU (Table 4-4).

Monitor: A program that assists in the real-time aspects of program execution such as operator command interpretation and supervisor call execution. Sometimes called supervisor (Section 3).

Most Significant Bit (MSB): Bit having the most value; the left-most bit representing the highest power of base 2. This bit is often used to show sign with a 1 indicating negative and a 0 indicating positive.

Object Program: The hexadecimal interpretations of source code output by an assembler program. This is the code executed when loaded into memory.

One's Complement: Binary representation of a number in which the negative of the number is the complement or inverse of the positive number (all ones become zeroes, vice versa). The MSB is one for negative numbers and zero for positive. Two representations exist for zero: all ones or all zeroes.

Op Code: Binary operation code interpreted by the CPU to execute the instruction (paragraph 4.5.1).

Overflow: An overflow occurs when the result of an arithmetic operation cannot be represented in two's complement (i.e., in 15 bits plus sign bit), (paragraph 4.3.3.5).

Parity: Means for checking validity of a series of bits, usually a byte. Odd parity means an odd number of one bits; even parity means an even number of one bits. A parity bit is set to make all bytes conform to the selected parity. If the parity is not as anticipated, an error flag can be set by software. The parity jump instruction can be used to determine parity (paragraph 4.3.3.6).

PC Board: (Printed Circuit Board) a copper-coated fiberglass or phenolic board on which areas of copper are selectively etched away, leaving conductor paths forming a circuit. Various other processes such as soldermasking and silkscreen markings are added to higher quality PC boards.

Program Counter (PC): Hardware register that points to the next instruction to be executed or next word to be interpreted (paragraph 4.3.1).

PROM: See Read Only Memory.

Random Access Memory (RAM): Memory that can be written to as well as read from (vs. ROM).

Read Only Memory (ROM): Memory that can only be read from (can't change contents). Some can be programmed (PROM) using a PROM burner. Some PROM's can be erased (EPROM's) by exposure to ultraviolet light.

Reset: Transfer control to operating system using the equivalent of a BLWP instructio to vectors in lower memory (0000_{16} and 0002_{16}). See Load.

Source Program: Programs written in mnemonics that can be translated into machine language (by an assembler).

Status Register (ST): Hardware register that reflects the outcome of a previous instruction and the current interrupt mask (paragraph 4.3.3).

Supervisor: See Monitor

Utilities: A unique set of instructions used by differnt parts of the program tc perform the same function. In the case of TIBUG, the utilities are the I/O XOP's (paragraph 3.3).

Word: Sixteen bits or two bytes.

Workspace Register Area: Sixteen words, designated registers 0 to 15, located in RAM for use by the executing program (paragraph 4.4).

Workspace Pointer (WP): Hardware register that contains the memory address of the beginning (register 0) of the workspace area (paragraph 4.3.2).

SPECIFICATION – MODEL 500 P

STANDARD EQUIPMENT			
POWER SUPPLY	415V ± 10%, 3ph, with Earth and Neutral wires, 50Hz, 6.5kVA.		
SIZE	LENGTH	WIDTH	HEIGHT
Mechanical Unit	1188mm (47 in)	711mm (28 in)	1854mm (73 in)
Control Console	750mm (29 in)	532mm (21 in)	1170mm (46 in)
WEIGHT			
Mechanical Unit	740kg (1630 lb)		
Control Console	100kg (220 lb)		
ARM MOVEMENT			
Vertical Travel	760mm (30 in)		
Horizontal Travel	*760mm (30 in)		
Swing Arc	240°		
WRIST MOVEMENT			
Rotation	Up to 180°, 2 position at 90°/sec maximum		
Sweep	*Up to 180°, 2 position at 90°/sec maximum		
GRIPPER	*None as part of standard equipment		
ARM SPEED			
Vertical axis	910mm (36 in)/sec		
Horizontal axis	910mm (36 in)/sec		
Swing axis	90°/sec		
REPEATABILITY (Approach from one direction only)			
At maximum reach	Better than ± 3mm (± 0.125 in) in Swing axis Better than ± 2mm (± 0.080 in) in Vertical and Horizontal axes		
LOAD CAPABILITY (Typical figures only; these will vary with details of each application. Figures are for gripper plus component(s) being handled)	*At rated speed 23kg (50 lb) *At reduced speed 55kg (120 lb)		
PROGRAMME			
Number of arm positions	30 (Positions may be visited more than once per programme)		
Number of steps in sequence	100 (A short sequence may be repeated several times around programme drum)		
INTERLOCKS			
Number of incoming and outgoing circuits	12 total		
GRIPPER & WRIST CONTROLS	Two circuits (one –on, one –off) are available as standard		
COOLING	*Air		
TEMPERATURE			
Ambient temperature range	0° to 45°C for mechanical unit		
HYDRAULIC FLUID	Mobil DTE Light (Mineral Oil)		

*ALTERNATIVES AVAILABLE	
Power Supply	220V, 380V, 500V ± 10%, 3ph., with Earth and Neutral wires, 50 or 60 Hz, 6.5kVA
Horizontal travel of arm	1060mm (42 in) with reduction in load capability to: At rated speed 16kg (35 lb) At reduced speed 36kg (80 lbs)
Wrist sweep movement	Servo control sweep axis (in lieu of one standard servo axis) with up to 290° arc
Gripper	Standard hydraulic mechanism with special jaws; vacuum, mechanical, electro–magnetic or other gripper type to special design
Cooling	Water
Hydraulic power unit	Separate from column
Lateral movement of mechanical unit	2 position system, or servo system (in lieu of one standard servo axis)

Data

A. Dimensions

	Mechanical unit	Console
Height	73 in (186 cm)	46 in (117 cm)
Depth (with arm extended)	80 in (202 cm)	20 in (51 cm)
Width	27 in (69 cm)	29.5 in (75 cm)
Weight	1300 lb (590 kg)	300 lb (136 kg)

B. Power requirements (into mechanical unit)

415V ± 10% 3 phase and neutral 50Hz at 11A. Other voltages in the range 220 to 500V are available.

C. Movements

Arm Horizontal	30 in (77 cm) (A 42 in (107 cm) reach arm can be supplied to order)
Arm Velocity with 20 lb (9.1 kg) load (See Note)	
Horizontal and vertical 36 in/sec (92 cm/sec)	
Swing 90°/sec	
Vertical	30 in (77 cm)
Swing	240°
Wrist rotate	0 up to 180°)*
Sweep	0 up to 180°)*
Gripper close	Dependent upon type

NOTE: Loads up to 100 lb can be handled at reduced speeds by the arm and gripper attachment, but not by the wrist, although this assembly can be locked by its stops whilst heavy loads are being manipulated.

*Only two positions on each, limited by adjustable mechanical stops.

May 15/69

D. Number of arm command positions

100, made up from a combination of:

Arm	30 discrete points		
Wrist	2 (in both rotate and sweep)		(ON-OFF)
Gripper	2		(ON-OFF)
Interlock	12		

E. Repeatability of arm command positions

Better than 0.125 in (0.32 cm) in swing
Better than 0.080 in (0.2 cm) in horizontal and vertical

F. Operating temperature range

0°C (32°F) to 45°C (113°F) ambient

G. Free air flow required

1000 ft^3/min (28m^3/min)

H. Hydraulic fluid

Type	Normally:	Mineral oil	Mobil DTE light
	But can use:	Phosphate ester	Mobil pyrogard 210 (fireproof)
Reservoir capacity		44 pints (25 litres)	
Operating pressure		1000 lb/in^2 (70 kg/cm^2)	
Pump capacity		8 imp gal/min (36.4 litre/min)	

Safety cut out operates if pressure drops below 800 lb/in^2
Normal oil operating temperature 55°C (130°F)

(Hydraulics locked until this temperature is reached, this takes about 6 minutes from switch-on. Safety cut out operates if fluid temperature exceeds 80°C).

I. Interlock equipment

8 relays, 24V coil double pole change-over contacts (24V, 0.5A supply available for energisation).

Contact rating 10A at 440Vac or 250Vdc. Maximum power switching capabil[illegible] (non-inductive load) 1.5kVA, 150W at 30Vdc or 70W at 100Vdc.

A 24V, 3.0A dc supply is available for operating external equipment.