A DISTRIBUTED MICROPROCESSOR CONTROL SYSTEM FOR AN INDUSTRIAL ROBOT

by

© Raad F. Rafauli, B.Eng.

A Thesis

Submitted to the School of Graduate Studies

in Partial Fulfilment of the Requirements

for the Degree

Master of Engineering

McMaster University

June 1981

MASTER OF ENGINEERING (1981)
Electrical and Computer Engineering

McMASTER UNIVERSITY
Hamilton, Ontario

TITLE: A Distributed Microprocessor Control System for an Industrial Robot

AUTHOR: Raad F. Rafauli, B.Eng. (McMaster University)

SUPERVISORS: Dr. N. K. Sinha, Department of Electrical and Computer Engineering
Dr. J. Tlusty, Department of Mechanical Engineering

NUMBER OF PAGES: ix, 81 , Appendices A-D (60), References (2)

ABSTRACT

Complex automation systems, such as industrial robots, require a computer-based control system for the effective utilization of the advanced technology.

A state-of-the-art was studied and presented in this thesis.

A distributed computer control system for a modified Unimate 2000 robot, is presented. The 16-bit Intel 8086 microprocessor was used as the master computer, and the 8-bit Intel 8748 microprocessor as the slave processor.

The system is effective and the experimental results agree with the simulation.

ACKNOWLEDGEMENTS

The author wishes to express his sincere thanks to his supervisors, Dr. N. K. Sinha and Dr. J. Tlusty for their support and guidance throughout the course of this work.

Also, many thanks to Dr. R. Kitai and Mr. P. Young for their time spent in constructive discussions.

Finally, a special thank you to my two brothers for their financial support, and to my wife for her moral support.

TABLE OF CONTENTS

	Page
CHAPTER 1 - THE INDUSTRIAL ROBOT	
1.1 Introduction	1
1.2 Major Coordinate System	1
1.2.1 Cartesian Work Envelope	3
1.2.2 Cylindrical Work Envelope	3
1.2.3 Spherical Work Envelope	3
1.2.4 Articulated Work Envelope	3
1.3 Power Sources	4
1.4 The Control	4
1.4.1 Non-Servo Controlled	4
1.4.2 Servo Controlled	4
1.4.3 Manipulator Path	5
1.5 Command Generation	7
1.5.1 Cartesian Interpolation	8
1.5.2 Joint Interpolation	8
1.5.3 Cubic Spline Interpolation	
1.5.4 Constant Acceleration, Constant Velocity Algorithm	17
1.6 Programming the Robot	22
1.6.1 On-Line Programming	22
1.6.2 Off-Line Programming	25
1.6.3 Programming Languages	27

Page

CHAPTER 2 - THE CONTROL SYSTEM

2.1 Introduction 30

2.2 System Architecture 32

2.3 Detailed System Functional Description 35

2.3.1 The Master Computer 35

2.3.2 The Slave Processors 35

2.3.3 The Manual Controller 36

2.3.4 The Interlock Signals 37

2.3.5 The Mechanical Range Limitations and the Safety Features 37

2.4 Position Control System 38

CHAPTER 3 - HARDWARE CONSIDERATION

3.1 The West-Amp Servo Amplifier 43

3.2 The Electro Craft Servo Amplifier 43

3.3 The Electro Craft Servo Motors 44

3.4 The Position Feedback Transducer 45

3.5 The Master Computer 45

3.6 The Servo Cards 46

3.6.1 Control Board Select 49

3.6.2 Digital Data Input 50

3.6.3 Digital Data Output and Transfer Acknowledge 51

3.6.4 Synchronizations 53

3.6.5 Analog Signal Generations 55

3.6.6 System Protection and Manual/Computer Select 57

3.6.7 The Positional Feedback 58

3.6.8 The Central Processor Unit (CPU) 60

Page

CHAPTER 3 (continued)

3.7 The Manual Controller 62

CHAPTER 4 - SOFTWARE CONSIDERATION 69

CHAPTER 5 - CONCLUSION 80

APPENDIX A - PLM/86 PROGRAM LISTING

APPENDIX B - ASM48 PROGRAM LISTING

APPENDIX C - SCHEMATIC DIAGRAMS

APPENDIX D - iSBC86/12A MONITOR COMMANDS

REFERENCES

LIST OF FIGURES

Figure		Page
1.1	Major Robot Coordinate Systems	2
1.2	Manipulator Path Control	6
1.3	Quadratic Interpolation of Joint Position	10
1.4	Single-Axis Point-To-Point Motion Using A Single Cubic Spline Function	15
1.5	Single-Axis Point-To-Point Motion Using Cubic Spline Functions	16
1.6	Single-Axis Point-To-Point Motion Using A Single Trapezoidal Velocity Profile	19
1.7	Single-Axis Point-To-Point Motion Using A Triangular Velocity Profile	19
1.8	Velocity Profile for Three Joints Motion	20
1.9	Single-Axis Path Control Motion Using A Trapezoidal Velocity Profile	21
1.10	Constant Velocity for Off-Line Programming	27
2.1	The Five Axes of Motion of the Unimate 2000 Robot	30
2.2	The Distributed Microprocessor Control System	34
2.3	Closed-Loop Servo System	39
2.4	Computer Within the Positional Loop	40
2.5	Photograph of the Velocity Profile	40
2.6	Approximate Model for One Joint of the Robot	40
2.7	The Root-Locus for the Positional Servo System	42
3.1	Common Bus Configuration	47

Figure		Page
3.2	Control Board Enable Signal Generation	48
3.3	Functional Block Diagram of the Servo Card	48
3.4	Control Board Select Signal Generation	49
3.5	Digital Data Input	50
3.6	Digital Data Output and Transfer Acknowledge	52
3.7	Synchronizing Signals	54
3.8	2-Byte Parallel Loading of the DAC	56
3.9	Digital to Analog Converter, Bipolar Operation	56
3.10	Computer/Manual Select and System Protection	57
3.11	The Positional Feedback	59
3.12	The CPU Interface on the Servo Card	60
3.13	Velocity, Acceleration Control	63
3.14	Command Velocity Generation	63
3.15	Generated Command Controller	65
3.16	Miscellaneous Functions Generation	67
4.1	System's Software Organization	70
4.2	Plot of the Fortran Simulation for one Joint of the Robot	74
4.3	Control Program Flowchart	76

CHAPTER 1

THE INDUSTRIAL ROBOT

1.1 Introduction

All industrial robots include three basic elements: an articulated arm, controls and power source. The arm moves in up to six different axes. Three are provided by the arm itself which may rotate in an arc around its base and reaches up and down vertically and in and out horizontally. As many as three other motions are added at the wrist which allows the hand or gripper attached to it to rotate, and move in vertical and horizontal planes.

These articulations enable the arm to move to some predetermined point in space, work on an object, and then return to the original position or to a sequence of other points in space.

The pattern of movements depends on the need of the job and the programming capacity of the controls.

1.2 The Major Coordinate Systems for Industrial Robots

The point in space accessible to a robot depends on its designed work envelope. There are four basic design categories: Cartesian, cylindrical, spherical, and articulated, that carve out slightly different geometric work areas in space, as shown in Figure 1.1.

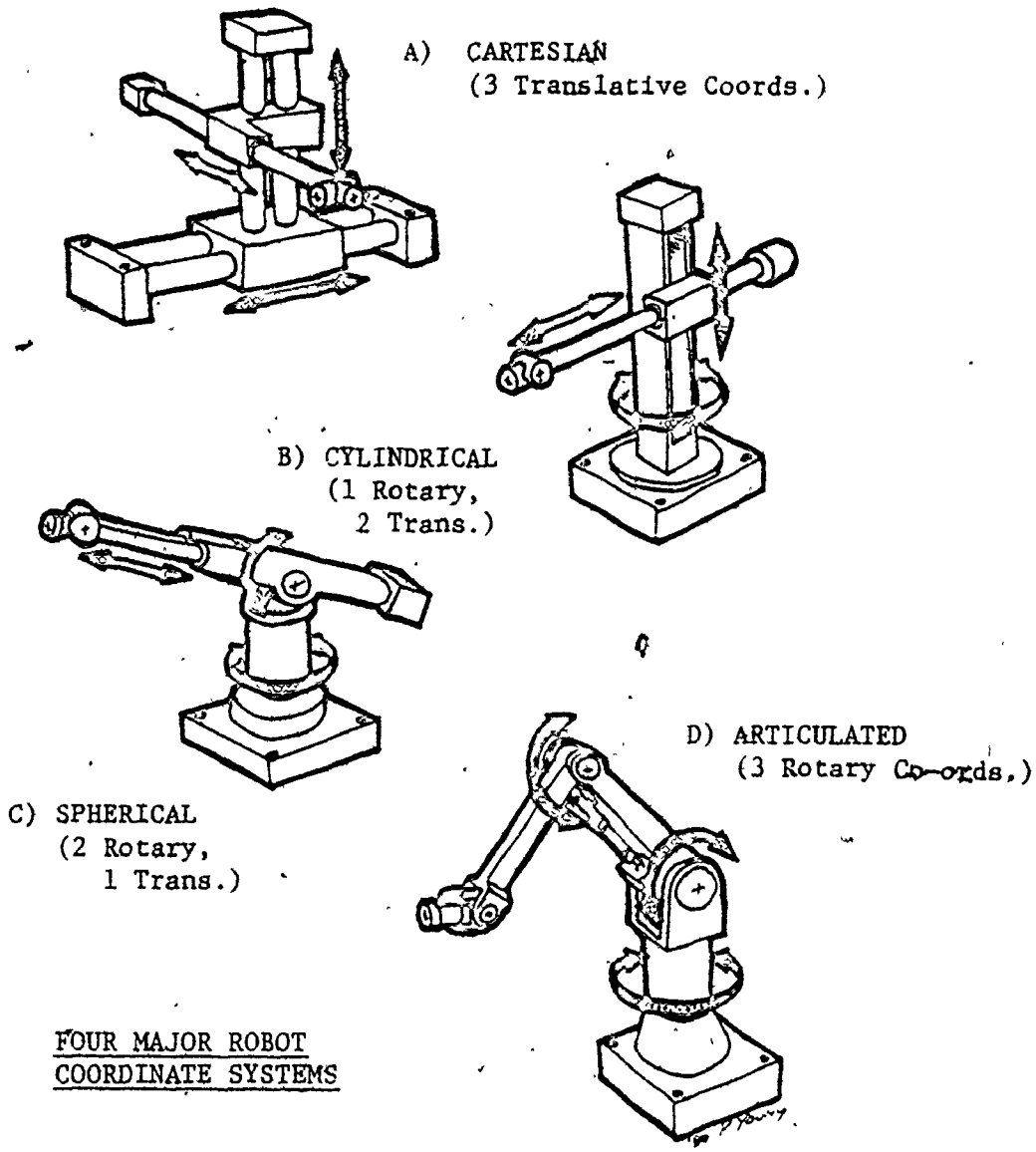


Figure 1.1 Major Robot Coordinate Systems

1.2.1 The Cartesian work envelope is created when the horizontal robot arm is attached to a vertical column and moves up and down and in and out in relation to this column. The column itself is mounted on a third linear axis which moves in two directions. These three movements combine to generate a work envelope in the shape of a three-dimensional rectangle.

1.2.2 The cylindrical work envelope is created when the horizontal arm is attached to a vertical carriage and moves up and down and in and out in relation to this carriage. The carriage itself is mounted on a pedestal which rotates. These three movements combine to generate a work envelope that is a portion of a cylinder.

1.2.3 In a spherical work envelope, the robot is mounted on a base somewhere near its midpoint so that it can tilt up and down and in and around its support point. The arm extends horizontally. These motions combine to generate a work envelope that is a portion of a sphere.

1.2.4 In an articulated work envelope, the arm is attached to a pedestal that rotates. The arm itself is jointed, adding a kind of elbow movement to the articulations. Thus, as the arm rotates and reaches in the horizontal and vertical planes, it generates a portion of a sphere in space and as the elbow bends, it enables the robot arm to come in close to the base. The jointed action enlarges the work area accessible to the robot.

1.3 Power Sources

Power is delivered to robots by hydraulic, pneumatic, or electic motors. Most robot makers have gone the hydraulic route, which is especially good where heavy loads are involved. Electric motors have the advantage of being less noisy, cleaner, more accurate, and may be less expensive. Pneumatically activated robots take advantage of the compressed air supply that is commonly available.

1.4 The Controls

When classified according to the type control, robots fall into two general categories:

1.4.1 The non-servo are less flexible than the servo controlled, but are also less expensive and very accurate. In this type of robot, each joint moves between two end points that are determined by mechanical stops. When hydraulic or pneumatic power is delivered to the joints, the motors drive the joint until it reaches a mechanical stop. A limit switch then signals the power off until the program calls for another move. The valves then open and again deliver fluid to the joints.

In these robots, controlled acceleration and deceleration require special designs. The number of different positions to which the robot can go in sequence is limited.

1.4.2 The servo controlled robots can do many things the simpler

ones can't. They can be programmed to do a more complex sequence of tasks and can have controlled velocity, acceleration, and deceleration. A servo controlled robot has encoders, potentiometers, resolvers or other feedback devices at each of the joints. These devices feed position signals back to the controller which compares them with the commanded position. The controller then sends correction signals to the hydraulic or electric motors at the joint. This thesis will consider the servo controlled robots only.

1.4.3 Manipulator Path

Consider a manipulator consisting of an arm, the end of which is its wrist, and an end-effector, which may be a hand or a tool. The end-effector is usually attached directly to the wrist. The three-dimensional path of the end-effector of a manipulator is described by two variables. [1]

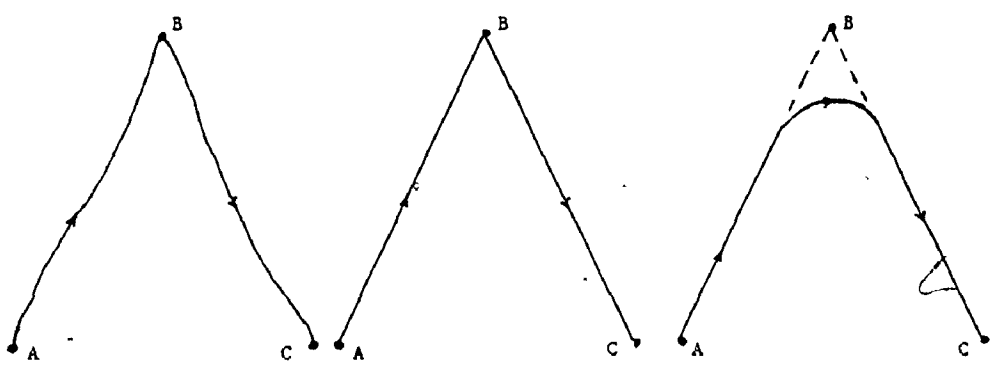
1) End-effector position: The position of a point fixed in the end-effector in a reference cartesian coordinate system.
2) End-effector orientation: The orientation of a cartesian coordinate system attached to the hand or tool such that the system's origin coincides with the end-effector position.

The above definition of end-effector position and orientation is based on attaching a cartesian reference frame to the end-effector. The word "state" is used to represent both position and orientation of

an end-effector.

There are two common ways to control an industrial robot with position servo.

(a) For material handling jobs, a point-to-point control is sufficient where the joints may start moving at the same time but don't all necessarily finish moving at the same time under program control. When this is the case, the end-effector assumes various states as it moves between the two end points and the shape of the path is not predictable. Figure 1.2a shows three trained end-effector positions, A through C, in three dimensions and illustrates the path travelled by the end-effector between these positions. Point-to-point movements are fast and the shape of the path is irrelevant, but the start and end points are important.



(a) Point-to-point

(b) Interpolated Straight-Segment Path

(c) Interpolated Straight-Segment Path with Smooth Transitions Between Segments

Figure 1.2 Path Cont

(b) For some jobs, such as arc welding, or spray painting, the joints must move in coordination so that each moves the same proportion of the programmed distance as the other, thus the end-effector position will follow a specified three-dimensional path with high resolution and the end-effector orientation must vary smoothly along that path.

Straight line path control is a special case of path control. Moving the end-effector along a straight line is desired for other reasons than job constraints; no torque is exerted on a load carried by the hand, reduced inertial load for a rotary manipulator, ease of computation, and ease of collision avoidance. However, fast end-effector motion along a straight segment path, such as the one shown in Figure 1.2b causes sudden changes in end-effector velocity. The corresponding acceleration, or deceleration, can exceed the limit and can excite structural vibrations in the manipulator. To avoid these vibrations, the end-effector bypasses each corner point along a smooth transition between segments as shown in Figure 1.2c and described in the next section.

1.5 Command Generation

Commands should be generated for simultaneous motions in all axes in such a way that all the axes start and end, more or less, simultaneously. A certain magnitude of the resultant velocity vector will be obtained. The hand, or tool, can be controlled to move smoothly along a straight line between two states by interpolating positions

between the states.

From the point of view of the interpolation, the distinction is made between: Cartesian interpolation and joint interpolation.

1.5.1 Cartesian Interpolation

In moving the hand from point S to E in space, cartesian interpolation can be used if the time, T, that the hand takes to travel this distance is known, by dividing T into short intervals, each of duration t, and computing the joint position (J) of all axes for each interval. The cartesian interpolation interval t is chosen to be as short as possible and still allow the computer to make all necessary computations. The shorter t, the smoother is the hand motion.

1.5.2 Joint Interpolation

The manipulator motion can be smoothed even more by using joint interpolation between the J values computed by cartesian interpolation. Joint interpolation amounts to dividing t into shorter intervals, each of duration $\Delta t = t/N$, where N is the number of interpolations, and computing new J values.

If the joint positions J0, J1 and J2 of Figure 1.3 are known from the cartesian interpolation for two successive intervals, and no more interpolation is done, then joint velocity will undergo a sudden change at joint position J1. But if joint J1 is bypassed by means of interpolation between segment J0-J1 and J1-J2, then lower acceleration and smoother motion is obtained. Quadratic interpolation

can be used to produce a new J value for each Δt interval along a quadratic function of time, that is, tangent to the straight segments from J0 to J1 and from J1 to J2 at their midpoints, (J0 + J1)/2, and (J1 + J2)/2, respectively. The quadratic transition between these midpoints is characterized by four values:

1. An initial joint position JPOS,

   where

   $$JPOS = (J0 + J1)/2 \tag{1.1}$$

2. An initial joint velocity $JVEL_o$,

   which is the slope of segment J0 - J1

   $$JVEL_o = (J1 - J0)/t \tag{1.2}$$

3. A final joint velocity, $JVEL_f$, which is the slope of segment J1 - J2

   $$JVEL_f = (J2 - J1)/t \tag{1.3}$$

4. A constant acceleration JACC, where

   $$JACC = (JVEL_f - JVEL_o)t \tag{1.4}$$

Solving for equations (1.1 to 1.4) is done every t seconds, therefore, if we take t as a time unit, then the above equations are simplified to:

$$JVEL_o = J1 - J0 \tag{1.5}$$

$$JVEL_f = J2 - J1 \tag{1.6}$$

$$JACC = JVEL_f - JVEL_o \tag{1.7}$$

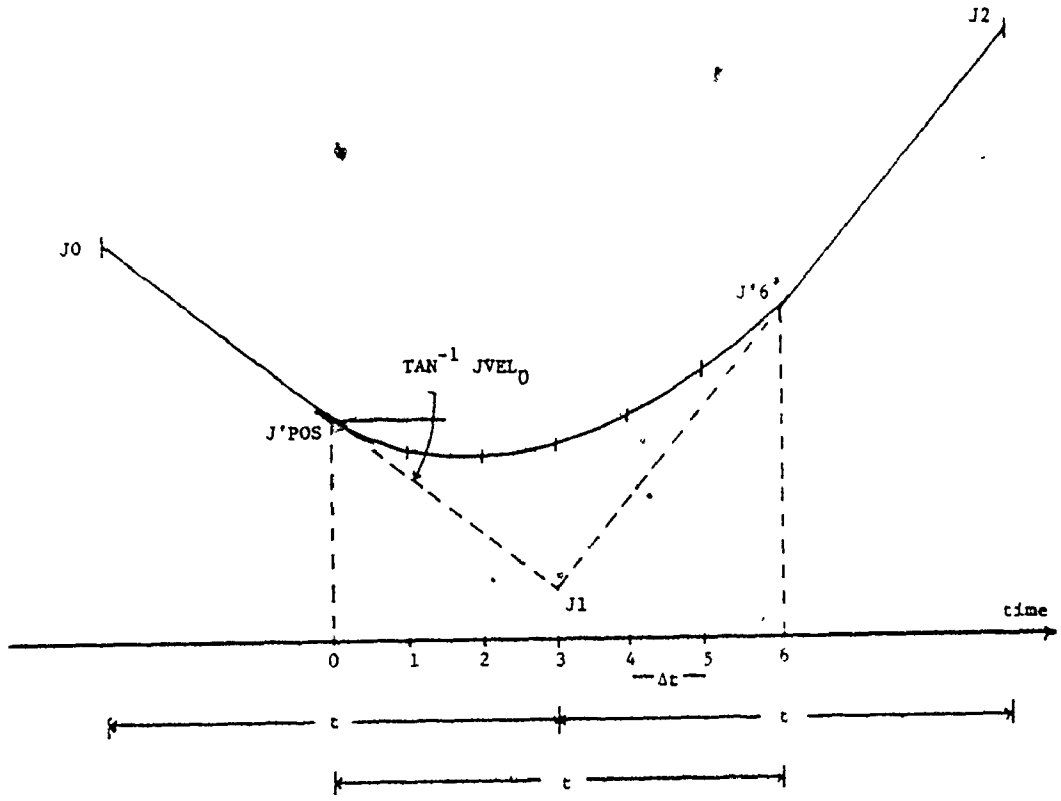


Figure 1.3 Quadratic Interpolation of Joint Position

Figure 1.3 shows six joint interpolation intervals for each cartesian interpolation interval, i.e. $\Delta t = t/6$.

1.5.3 Cubic Spline Interpolation

Another algorithm for the command generation is to define the commanded distance as cubic function of time. [2]

Consider that the three-dimensional vector $\underline{X}(t)$ represents the position of 3-axis robot as a function of time t. If t is equal to t_o when the motion starts at the beginning of a contour, then the

remainder of the contour can be broken up into K segments with end points at times $t = t_1, t_2, t_3, \ldots t_k$. Each of these segments can then be fitted with a set of third-order polynomials, each polynomial representing one dimension of the segment as a function of time.

Position along the ith segment can be approximated by the following equation:

$$\underline{X}(t) = Ai\underline{b}(t) \tag{1.8}$$

where

$$\underline{b}(t) = [t^3 \ t^2 \ t \ 1] \tag{1.9}$$

A is an N x 4 coefficient matrix that needs to be computed and stored in memory for each segment. Then it will be used to compute $\underline{X}(t)$ within the ith segment. Similarly, the velocity along the ith segment can be approximated by the following equation:

$$\dot{\underline{X}}(t) = Ai\dot{\underline{b}}(t) \tag{1.10}$$

where

$$\dot{\underline{b}}(t) = [3t^2 \ 2t \ 1 \ 0] \tag{1.11}$$

A_i can be determined by using the cubic spline interpolation. The mathematical spline is continuous and has both a continuous first derivative and a continuous second derivative which guarantees the position, velocity and acceleration match at the end points of adjacent segments. [3]

Four boundary conditions at the end points of the ith segment are required to compute the Ai matrix. In mechanical systems the

position, velocity and acceleration are likely to be important, given a total of six boundary conditions to choose from.

Position and velocity $\underline{X}(t)$, $\dot{\underline{X}}(t)$ at time t_{i-1} and t_i are used to compute the spline coefficient matrix for the ith segment.

$$Ai = [\underline{X}(t_{i-1})\ \dot{\underline{X}}(t_{i-1})\ \underline{X}(t_i)\ \dot{\underline{X}}(t_i)][\underline{b}(t_{i-1})\ \dot{\underline{b}}(t_{i-1})\ \underline{b}(t_i)\ \dot{\underline{b}}(t_i)]^{-1} \quad (1.12)$$

This interpolation can be further simplified to suit the microprocessor environment by normalizing the time t. The variable S can be introduced to simplify computation of the coefficient matrices and generation of joints along the path. S varies from 0 to 1 as t varies from t_{i-1} to t_i, t is related to S by:

$$t = t_{i-1} + (t_i - t_{i-1})S \quad (1.13)$$

Equations 1.8 through 1.13 can be rewritten as:

$$\underline{X}(S) = Ai\ \underline{b}(S) \quad (1.14)$$

$$\dot{\underline{X}}(S) = Ai\ \dot{\underline{b}}(S) \quad (1.15)$$

where

$$\underline{b}(S) = [S^3\ S^2\ S\ 1] \quad (1.16)$$

$$\dot{\underline{b}}(S) = [3S^2\ 2S\ 1\ 0] \quad (1.17)$$

Boundary conditions at the end points are now

At $t = t_{i-1}$: $S = 0$, $X(S) = X(t_{i-1})$

$$\dot{X}(S) = X(t_{i-1})\ dt/ds$$

At $t = t_i$: $S = 1$, $X(S) = X(t_i)$

$$\dot{X}(S) = X(t_i)\ dt/ds$$

The coefficient matrix A_i is now

$$A_i = [\underline{X}(t_{i-1})\ \underline{X}(t_{i-1})\ dt/ds\ \underline{X}(t_i)\ \underline{X}(t_i)\ dt/ds][\underline{b}(0)\dot{b}(0)\underline{b}(1)\dot{\underline{b}}(1)]^{-1} \quad (1.18)$$

and the matrix inversion is much simpler than before.

Let's consider a case where the position on an X axis is to be changed by $\Delta X = X_e - X_s$, with velocity $X = 0$ before and after the motion. Assume that Δt is allowed to complete the motion. To use one cubic spline function, the function must satisfy the end conditions,

at $S = 0, X(0) = X_s$, $\dot{X}(0) = 0$

at $S = 1, X(1) = X_e$, $\dot{X}(1) = 0$

Substituting the above conditions in equations 1.14, we get,

$$\underline{X}(S) = -2(\Delta X)\ S^3 + 3(\Delta X)\ S^2 + X_s \quad (1.19)$$

$$\dot{\underline{X}}(S) = -6(\Delta X)\ S^2 + 6(\Delta X)\ S \quad (1.20)$$

$$\ddot{\underline{X}}(S) = -12(\Delta X)\ S + 6(\Delta X) \quad (1.21)$$

Let's set

$$\ddot{X}(S) = 0 \quad \text{in Equation 1.21}$$

Then

$$-12(\Delta X)\ S + 6(\Delta X) = 0$$

and solve for S

$S = .5$ which is a point midway between X_s and X_e, where the velocity, $\dot{X}(S)$, is maximum. Substituting $S = 0.5$ into Equation 1.20 gives positive results (Maximum).

$$\dot{X}(S)_{max} = 3|\Delta X|/2$$

or

$$\dot{X}(t)_{max} = 3|\Delta X|/2\Delta t \tag{1.22}$$

$$\ddot{X}(t)_{max} = 6|\Delta X|/\Delta t^2 \tag{1.23}$$

In general, most mechanical systems have some limitations, namely, velocity and acceleration. The constraints in this case are maximum velocity and acceleration. Let V_x and A_x be the maximum allowable velocity and acceleration, respectively, then the total time for a given segment Δt can be calculated accordingly from Equation 1.22 and 1.23.

$$\Delta t_v = 3|\Delta X|/2V_x \tag{1.24}$$

$$\Delta t_a = (6|\Delta X|/A_x)^{1/2} \tag{1.25}$$

Thus, when using one spline to move a distance ΔX, the choice of Δt_x or Δt_a, whichever is greater, will result in the fastest possible motion without exceeding the velocity or acceleration limits.

Figure 1.4 shows the path generated by one spline, the average velocity over the entire path is $\Delta X/\Delta t$ which is equal to $2/3V_x$. If ΔX is a large distance, then the single spline method is not so effic-

ient because it takes too long to go from X_s to X_e.

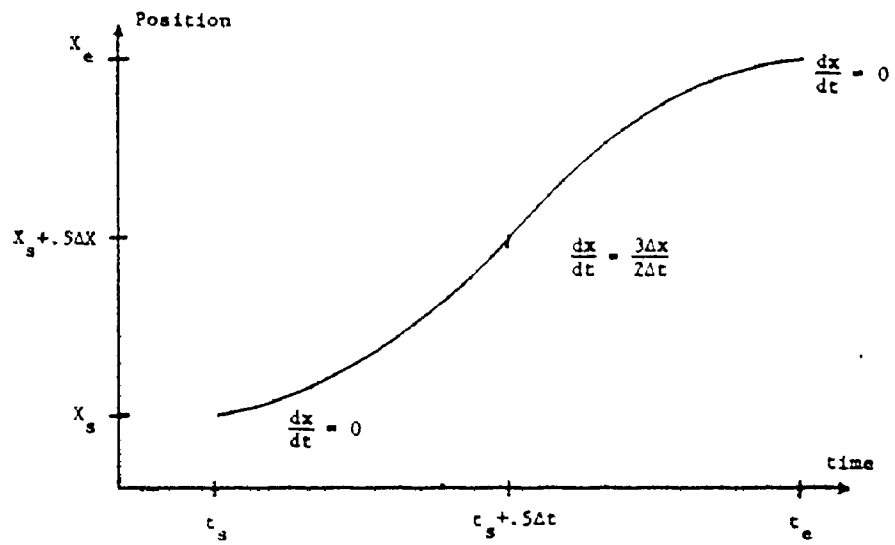


Figure 1.4 Single-Axis P-T-P Motion Using a Single Cubic Spline Function

Three spline functions are more efficient in case of large ΔX, first spline accelerates until maximum velocity is reached, second spline maintains a constant velocity, and the third function is to decelerate to stop. Figure 1.5 shows the path produced by three splines method.

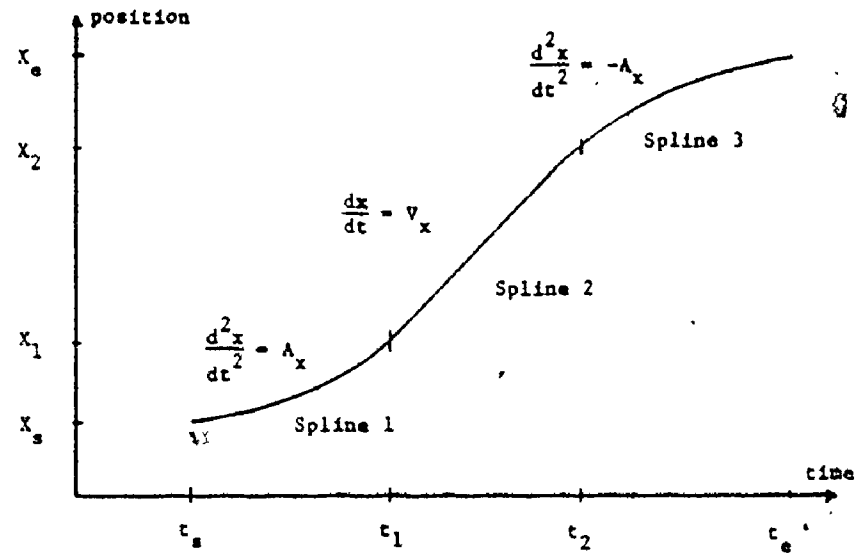


Figure 1.5 Single-Axis P-T-P Motion Generated Using Cubic Spline Functions

Three points are to be made at this stage:

1. The above described method can be applied to multi-axes system by relating all axes to common time base, then all the axis will start and end at, more or less, the same time.
2. If it is necessary to scale the velocity with which the machine follows the path, either the velocities used to compute the coefficient matrix can be scaled by a constant factor, or alternately, the time frame within the computer can be speeded or slowed with respect to real time to produce the desired result.
3. This method will generate a parabolic velocity profile

which is the most energy efficient profile from the point of view of the armature heat dissipation in the servo motor. [4]

1.5.4 Constant Acceleration, Constant Velocity Algorithm

In this algorithm, the maximum acceleration is determined from the mechanical system which should never be exceeded. It is then used to accelerate the joint from rest to some given constant velocity. [5] Figure 1.6 shows the trapezoidal profile, which is divided into three segments: Acceleration, run and deceleration. Figure 1.6a is the graph of the velocity. The rate of change of the velocity in the first segment is constant and equals to the maximum acceleration, second segment is constant and equals to the commanded velocity, the third segment is a constant rate of change in velocity, which is the deceleration part. Figure 1.6b shows the acceleration profile, which is equal to A_{max}, 0, $-A_{max}$. Figure 1.6c shows the distance travelled from X_s to X_e. Let's assume that $X_s = 0$.

In Figure 1.6 there are two auxiliary numbers,

X_1, which is the distance travelled during the acceleration time.

X_2, which is the distance travelled up until the deceleration time.

For a multi-axis system, it is desired that all the joints involved in the given motion, to start and stop at approximately the same time. In this case, the time it takes to execute the longest motion is computed as follows:

$$X1 = A_{max}/2\ t^2 = V^2_{max}/2\ A_{max} \tag{1.26}$$

If $$X1 < X_e/2 \tag{1.27}$$

then

$$T = 2*(2X1/A_{max})^{1/2} + X_e - 2X1/V_{max} \tag{1.28}$$

or if $$X1 \geq X_e/2 \tag{1.29}$$

then

$$T = 2*(X_e/A_{max})^{1/2} \tag{1.30}$$

Then this time T is chosen for all the joints and a new velocity value V'_{max} is computed for the other joints.

$$V'_{max} = \frac{A_{max} * T - (A^2_{max} T^2 - 4 A_{max} X_e)^{1/2}}{2} \tag{1.31}$$

This means that only one joint will utilize its maximum attainable velocity, V_{max}.

From the above, if condition (1.27) is true, then a trapezoidal profile can be used and maximum velocity will be reached. However, if condition (1.29) is true, then a triangular profile must be used.

Triangular profile is a special case of the trapezoidal profile where the run on constant velocity is omitted, as it is shown in Figure 1.7.

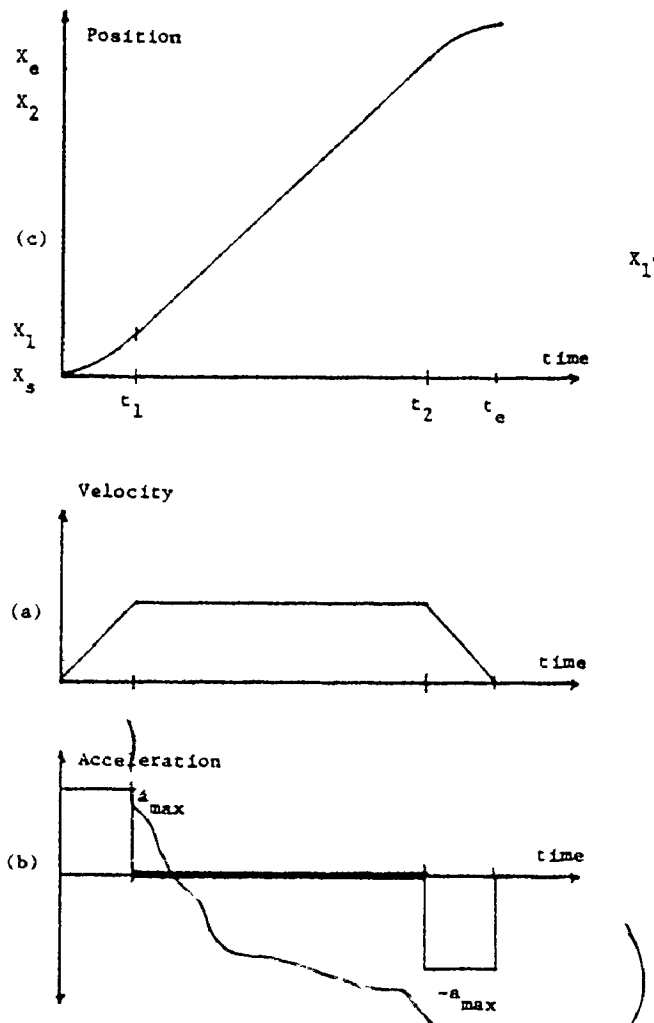


Figure 1.6 Single-Axis P-T-P Motion Using A Single Trapezoidal Velocity Profile

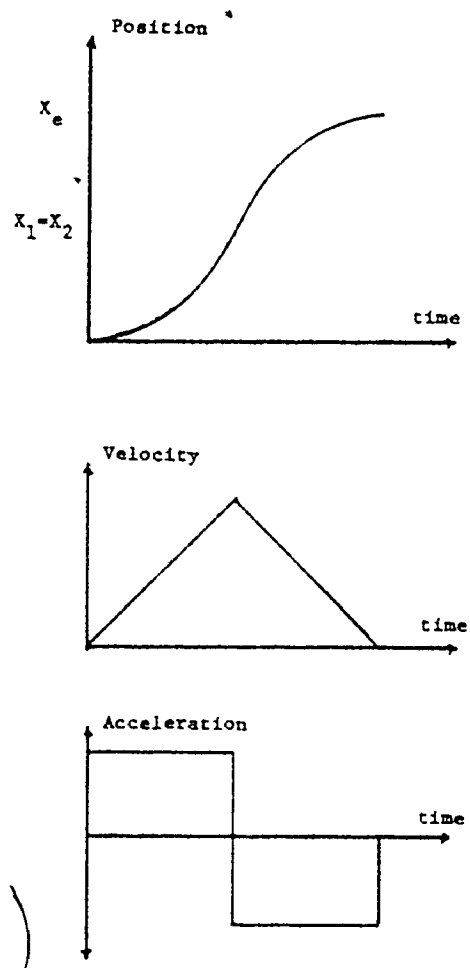


Figure 1.7 Single-Axis P-T-P Motion Using A Triangular Velocity Profile

Figure 1.8a shows a motion that requires three joints to move, none of which has enough distance to reach V_{max}, but they all start and stop at the same time. Triangular profile is used in one joint. Figure 1.8b is the same three joints, moving a longer distance, and one joint reaches V_{max}.

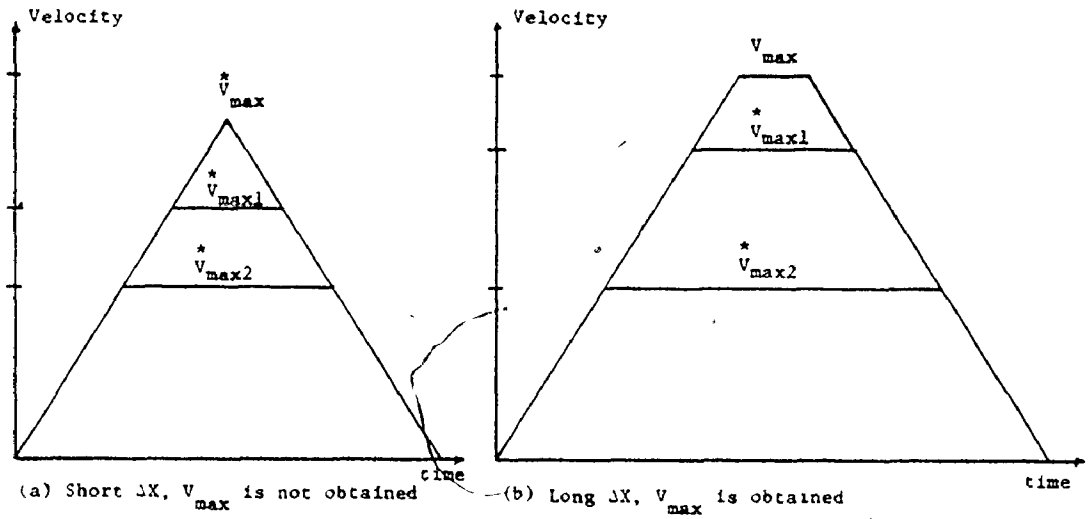


Figure 1.8 Velocity Profile for Three Joints Motion

The command generation for point-to-point control is accomplished by a software routine which sends out an updated commanded distance, X, once per constant time interval DT. The distance is expressed as an integer number of basic motion unit (1 BMU = .004 In or .1 mm), velocity, V, is expressed as an increment in distance per DT, and the acceleration is expressed as the increment of the velocity per DT.

Correspondingly, the parameters XE, VMAX, AMAX are now integers.

$$XE = INT(Xe/BMU)$$

$$VMAX = INT(V_{max}*DT/BMU)$$

$$AMAX = INT(A_{max}*DT/BMU)$$

Also

For condition (1.27)

$$X1 = INT(V_{max}/2A_{max}/BMU)$$

$$X2 = XE - X1$$

For condition (1.29)

$$X2 = INT(XE/2)$$

A similar strategy is employed for the path control, however, for every section between points J and J + 1, the start and end velocity are not zero. Also, the motion is not performed at maximum velocity in one of the joints coordinates but at a, more or less, constant resulting velocity. All the time the limits of acceleration and deceleration have to be observed.

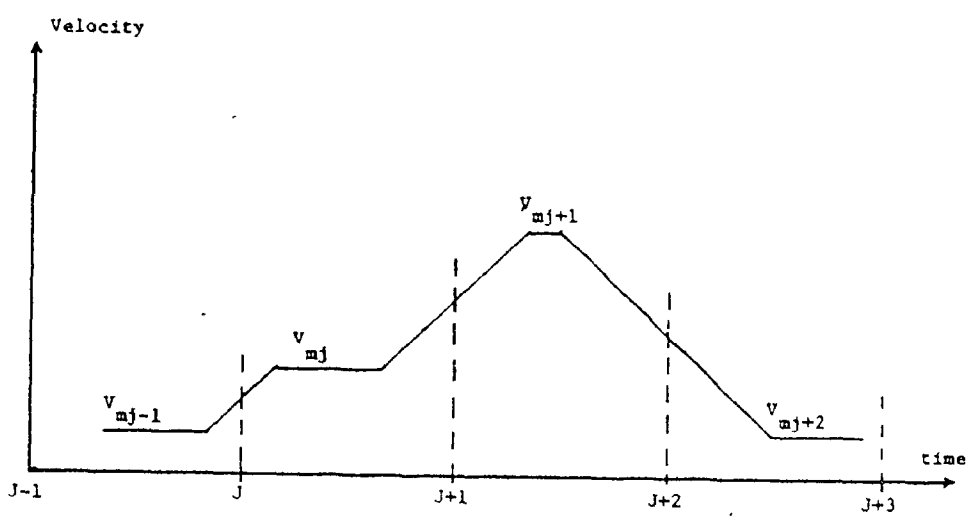


Figure 1.9 Single-Axis Path Control Motion Using a Trapezoidal Velocity Profile

The basics of this strategy are explained in Figure 1.9 which shows the velocity profile of the motion in three successive sections of one joint coordinate. The velocity of each section, V_j, V_{j+1}, is computed and a constant acceleration connects them. In general, the velocities will vary from section to section. They will be determined from the requirement of a uniform resulting velocity along a straight line between the end points of the section.

1.6 Programming the Robot

Programming the industrial robot can be done either by on-line teach mode or off-line programming.

1.6.1 On-Line Programming

Since the introduction of robots in the early sixties, all commercially manufactured servo-controlled industrial robots have been equipped for on-line programming. Whether the robot is designed for point-to-point or continuous-path operation, it is necessary for the operation to 'teach' the arm what to do.

Teaching the robot is accomplished by controlling manually a motion from the starting point S to the end point E one coordinate after another. This manual control can either be just a velocity command or positional. In any case, it is important that acceleration limitation be built into this control. When the end-effector reaches a desired state (position and orientation), the coordinate of point E can be recorded in memory upon the operator's request.

When using ordinary manual control, the only problem with this type of recording which is made regardless of the time factor, is to position the robot. If, for example, one has to combine five motions, one at a time, in order to put the end-effector in a desired state, it is a very complicated task to give the right orders to each of the five axes for the required movements. A 'syntaxer', a joy-stick with five degrees of freedom, overcomes this difficulty. It is an advanced manual control system which enables movements to be made in an intuitive manner. The feasibility of this syntaxer is governed by the structure of the robot.

When playing back the above taught motion, the end-effector will assume an unpredictable path between points S and E. The trajectory may not therefore be suitable if an obstacle hampers the movement. In most computer control robots, trajectory modifications can be made to avoid the obstacle, by inserting so-called 'passing' points between the stops. However, if the path between the two points is important, then some kind of interpolation between points S and E is needed before or during the playback. The interpolation will result in a controlled path, rather than just point-to-point. [6]

Other methods of teaching the robot would be by guiding by hand. Typically, robots using this type of teach are taught by the operator physically grasping the unit and leading it through the desired path in the exact manner and speed by which he wishes the robot to repeat the motion. While the device is moved through the desired path, the position of each axis is recorded on a constant

time base, thus generating a continuous time history of each axis position. Every motion that the operator makes, whether intentional or not, will be recorded and played back by the robot in the same manner. Since the operator must physically grasp the robot, it must be designed to be essentially counterbalanced and free under no power so that the task could be performed. Therefore, this method is generally limited to light-duty robots. [7]

However, it is also possible to guide a heavy-duty robot if it is equipped with force sensors that can convert the deflector on the end-effector to an electrical signal. The electrical signals are then used to drive the servo motor of each joint in a manner to reduce the deflection on the end-effector. [8]

Teaching the robot can also be done through a simulator. The simulator is kinematically identical with the robot and it also has identical positional feedback encoders. However, it does not contain any drives of the coordinate motions and its structure may be rather light. For programming, the simulator is moved along the desired path by pulling its end manually. Not having any drives, it does not exert any resistance against being moved. The signals of the encoders are recorded in memory in regular time intervals. [9]

Automatic tracing of a path is also used to teach industrial robots. In this mode, a more complicated tracer and algorithm are used. The robot is simply led to the beginning of the path either manually or by computer control. Then the tracer guides the robot through the entire path, and the computer keeps track of the position of each

joint for future playback. [10]

A number of sensors of both the contact (mechanical) and non-contact (magnetic) type have been conceived and developed. [11]

While the on-line teaching procedure is sufficient for many applications, it does become very tedious when hundreds of points must be programmed individually.

1.6.2 Off-Line Programming

Off-line programming can be defined as the task of programming the robot through the use of remotely generated points. Since the entire programming operation is carried out off-line in another computer, the robot can remain in production during the development phase. Also, the program can be verified by observing the operation of the arm on a graphic terminal. In this way, any missing or incorrectly defined locations or functions can be detected, and potential collisions can be observed and corrected.

In the point-to-point off-line programming, the program is established by giving the coordinates of the two points in the rectangular coordinate working space of the robot. Thus, the coordinate transformation technique is used to transform the points to the joint coordinates.

However, for the path control, the desired path is determined mathematically. It may be the case of straight line or of a sequence of straight line sections. The end points of these sections may be determined as in the P.T.P. programming. The intermediate points are

determined by interpolation with a desired density.

While off-line programming offers many advantages, it also has a set of unique problems. [12] These include:

1) Need to adjust for inaccuracies in the robot. This includes the deflection of the arm when carrying a heavy load.

2) Aligning for coordinate reference frame is needed whenever cartesian coordinate information for a work piece is generated off-line, because of the zero-reference point of the robot. Therefore, a coordinate transformation is employed to transform the work piece coordinate to the robot coordinate.

3) In the case of a straight line from point S to point E where the straight line passes close to the center of the joint coordinates α, r, as shown in Figure 1.10, the distance a is small. For a uniform motion with velocity V along the line S-E, large acceleration in α coordinate arises around the middle point. Actually, for a = 0 it would be $\alpha = 0$ for the first half of the motion and $\alpha = 180^{\circ}$ for the second half with infinite acceleration in α coordinate in the middle point [5].

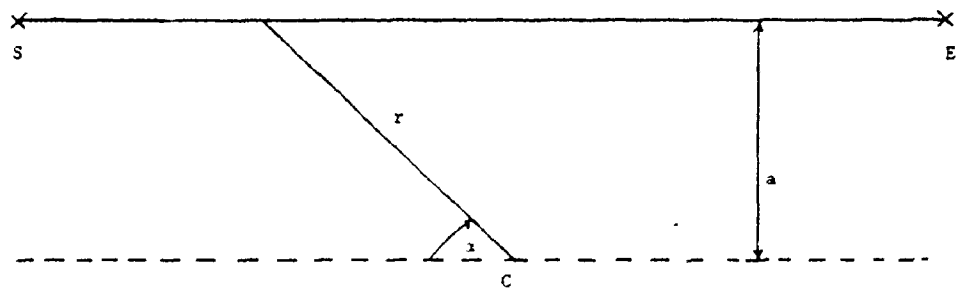


Figure 1.10 Constant Velocity for Off-line Programming

1.6.3 Programming Languages

A high level language must be employed in off-line programming. Over the past years, several high level languages have been developed for programming robots. Some of the high level languages are:

1) AL - This language was developed and used at Stanford University. There are three components in the integrated AL system

AL compiler

AL Interpreter

AL Runtime System, which executes the code generated by the compiler and interpreter.

The language statements consist of, motion commands, conditional clauses, and DO operations. Error handling capabilities for the interpreter were designed keeping in mind that the user is always present to handle the errors. The user is given the option of con-

tinuing after correcting the error, to swap to the text editor to correct the source file, to do a local correction which will be recorded in a new file, or to abandon compiling of the program. [13]

2) VAL - This language is also currently used in programming industrial robots. It consists of series of one word commands which often are followed by information that the command can operate upon. It has program editing commands, which are used for creating and modifying user programs.

It is also used in interactive mode for an on-line programming technique. [14]

3) INDA - A general purpose software package for fast interactive programming of computer controlled robots. Developed by SRI INTERNATIONAL using the RTL/2 Programming Language. INDA system consists of three parts, a compiler, an interpreter, and an editor. The interpreter supports single-step operation, execution tracing, breakpoints. It also works in an immediate mode, where the operator can input one statement through the keyboard, and the robot will move accordingly.

Each statement consists of actions that describe steps in the algorithm of the program, declarations, that define symbols for variables, procedures and labels. [15]

4) AUTOPASS - This language and operating system is currently under development at IBM.

5) APT - The APT language has been used for years to program CNC machine tools. At present time, an 'ANSI' study group has been

formed to determine the feasibility of expanding this language so it can be used to off-line program an industrial robot.

CHAPTER 2

THE CONTROL SYSTEM

2.1 Introduction

This chapter discusses the design of a distributed control system for a 5-axis industrial robot. The chapter also outlines the elements of the system. The design of a position control is discussed.

The robot is a unimate MARK II series 2000, which was modified at McMASTER UNIVERSITY to have DC servo-motor drives instead of its old hydraulic drives. It has a spherical work envelope with five degrees of freedom as described and labelled in Figure 2.1 [16].

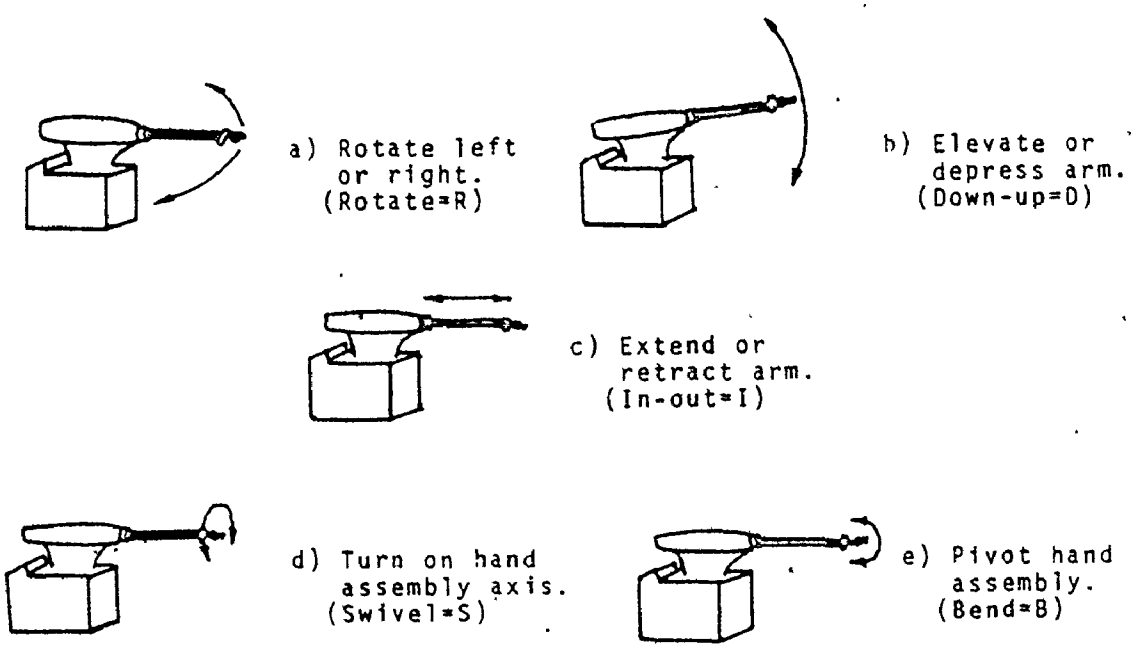


Figure 2.1 The Five Axes of Motion of the Unimate 2000 Robot

Two of the axes (lift and rotate) have mechanical limitations on acceleration and deceleration. However, the limitation on acceleration and deceleration is imposed on all five axes by using the trapezoidal velocity profile in all the five axes.

One motivation for adopting a distributed system is modularity, both functional and physical. Functional modularity reflects the tendency of the system to remain constant over possible system sizes. Physical modularity reflects the fact that we are trading software cost for low hardware cost. It also provides greatly improved fault detection, ease of maintenance and higher reliability.

Once the decision was made on using a distributed system, the following set of points were followed.

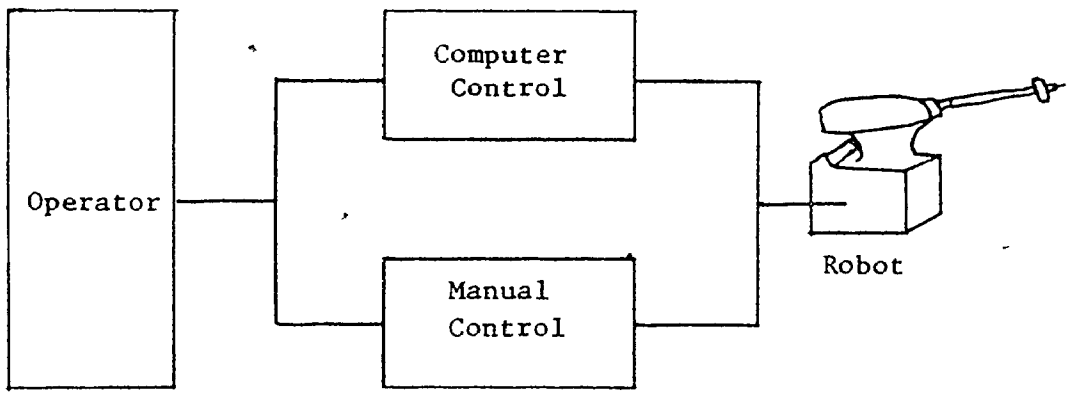
1) Each joint should be controlled optimally by a separate processor.
2) Each DC servo motor has a hardwired velocity loop, and a software positional loop.
3) Each processor should communicate with the servo motor and the master computer asynchronously.
4) Finally, a design approach should be used that will keep as much of the control system as possible independent of a particular robot configuration. If one more degree of freedom is added, then only one servo card need be added to the system, and some software modifications to support this card.

2.2 System Architecture

The development of a distributed processing system for the control of an industrial robot has as its basis the concept of horizontal decomposition scheme. Each separate processor provides coordination and control of a single joint of the robot manipulator and has a means of communication with its master processor. This scheme can be compared with the vertical decomposition, (pipe lining) [17], [18], where the processors are assigned tasks by computational function. In this case, one processor is dedicated for communicating with the operator. Another processor computes all the joint transformations and interpolations, while other processors generate the control functions for all joints. [19].

However, the software is partitioned into a three level hierarchy. The lowest level is where the servo function is computed. Computed joint commands are compared against position feedback data. The differences between these values are error signals that will cause drive voltages to be sent to the actuators eliminating the error. The second level provides a transformation from unit distance to basic motion units and feeds its information to the first level. The third level supports the communication task between the system and the operator. The first level is written in assembly language ASM48, while the second and third levels are written in higher level language, PL/M-86.

Figure 2.2 shows the parallel system that was partially built as the goal of this thesis.



The communication between the master and slave processors is via a fast parallel path. The operator is connected to the master computer via a keyboard-CRT terminal over a relatively slow serial data link.

The above control system is based on the INTEL 8086 Microprocessor which offers an advantageous mix of speed and functions, and INTEL 8748 microprocessors.

The system consists of the following:

a) Five 8-bit microprocessors (INTEL 8748), each of which has dedicated Clock, RAM, ROM, and I/O ports.

b) One master computer which is a Single Board Computer iSBC 86/12A with 32k Byte of RAM, 8k Byte of ROM, serial and parallel I/O

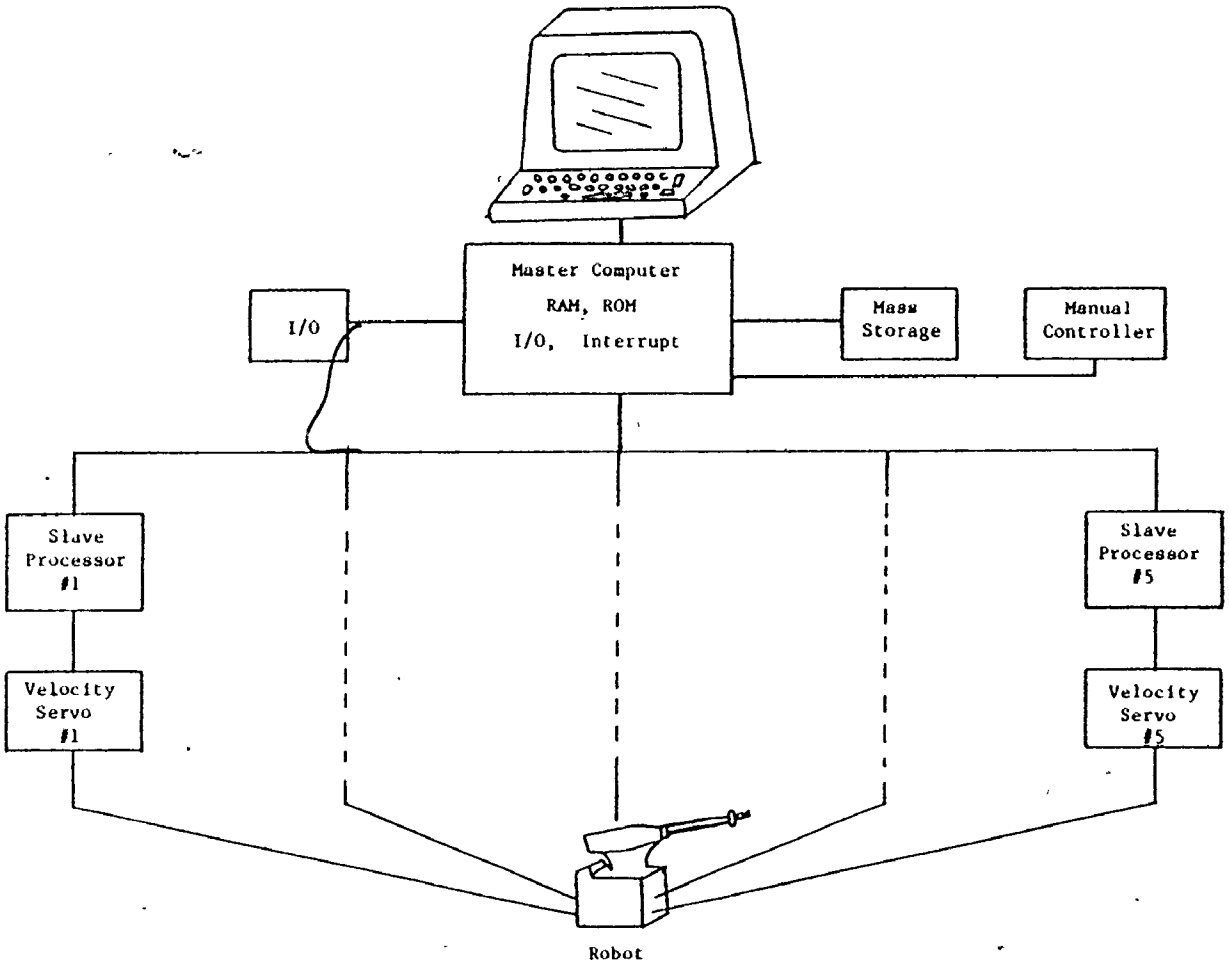


Figure 2.2 The Distributed Microprocessor Control System

lines, and a math processor.

c) Manual control/teach pendant which is used to move the robot manually, and to program the robot in the teach mode.

d) The system also has a cassette interface and a Qantex TM Model 200 Minidrive, each data cartridge can hold up to one million bytes of information.

2.3 Detailed System Functional Description

2.3.1 The Master Computer

The duties of the master computer are:

1) Receive commands from the operator via the Keyboard-CRT terminal.

2) Generate the joint position data, and send them to the proper slave processor, along with the velocity at which the joint should move.

3) Put the robot in manual/teach or computer control/playback mode.

4) In teach mode, it records the joint position data, and the functions that have to be performed at the end of the motion (i.e. output signal, wait for input, griper open/closed).

5) In playback, it will perform the previously taught functions.

2.3.2 The Slave Processors

Each processor performs the following functions:

1) In teach mode, it keeps track of the position of the end-effector, and sends the joint position data to the master computer upon receiving the "RECORD" signal from the operator via the manual control/teach pendant.

2) In playback mode, it receives the commanded distance and velocity from the master computer.

3) It generates the proper velocity command to the servo amplifier.

4) It controls the acceleration of the servo motor and maintains the commanded velocity.

5) It closes the positional loop, and performs the slowing down to stop function.

6) When the joint has moved the entire commanded distance, the motion is stopped, the end position is maintained, (the position loop would correct for any drift due to servo-amp or a physical load on the end-effector) and the master computer is informed of the completion.

2.3.3 The Manual Controller

When the manual controller is selected, it can perform the following functions:

1) Move only one axis of the robot, at a time, at a preset velocity, controlled acceleration and deceleration if needed.

2) Keep all other axes locked out in a closed position loop.

3) When the desired state of the end-effector is reached, it could be recorded by pushing the "RECORD" button on the manual controller.

4) It also programs the "WAIT FOR N" where N is an input line, and "OUTPUT M" where M is an output line. The status of the gripper (OPEN, CLOSED) is also programmed by this manual controller.

5) The stored program can be single stepped by setting single step

mode via the Keyboard-CRT terminal and using the "NEXT" key on the manual controller.

6) One other function key that will be used in future development is "END/MID" point which indicates whether the programmed point is an end point or intermediate point.

2.3.4 The Interlock Signals

Of primary importance in a robot operation are the interlocks which provide communication between the robot and the equipment it is serving.

When interlocks are properly used and of sufficient number, costly collisions, jam-ups, and costly damage to the robot and equipment can be prevented.

There are a total of eight interlock signals (this can be expanded), four of which are used as buffered input signals. The robot can be programmed to come to stop and wait for an input signal or sequence of signals to occur, then the robot will continue executing the program. Alternatively, it can come to a point in the program and output an electrical signal to one of the other four interlock lines. This signal can be buffered to drive a small solinoid or a large motor.

2.3.5 Mechanical Range Limitation and Safety Features

Each axis has two limit switches at its extreme positions, they are activated if the operator accidentally tries to drive the robot manually past its limits, or if the computer fails and the robot remains in motion.

If one or more limit switches are activated, then the following procedure is executed by a reliable hardware:

1) If the robot is under computer control, it will be switched over immediately to manual control mode and all the five axes will slow down to safe stop simultaneously.
2) The operator must then move the robot manually to clear the limit switches and a restart procedure must be followed.

A tapeswitch will be put around the working area of the robot, when activated by the operator, the same safety stop procedure will be executed.

2.4 Position Control System

One means of controlling the angular position of a motor is by an open-loop control system. In the open-loop system, the output will follow the desired function as long as all the system variables are constant. Any change in load, amplifier gain, or any other system variable will cause a deviation from the desired value. In order for the motor to conform to a desired function independent of small changes in these variables, a closed-loop servo system, such as the one shown in Figure 2.3 must be built.

In the closed-loop system, the output variable is measured, fed back and compared to the input function. Any difference between the two is deviation from the desired result, the deviation is amplified and used to correct the error. In this manner, the closed-loop system is essentially insensitive to variations in parameters, and

therefore performs correctly despite changes in load condition and other parameters.

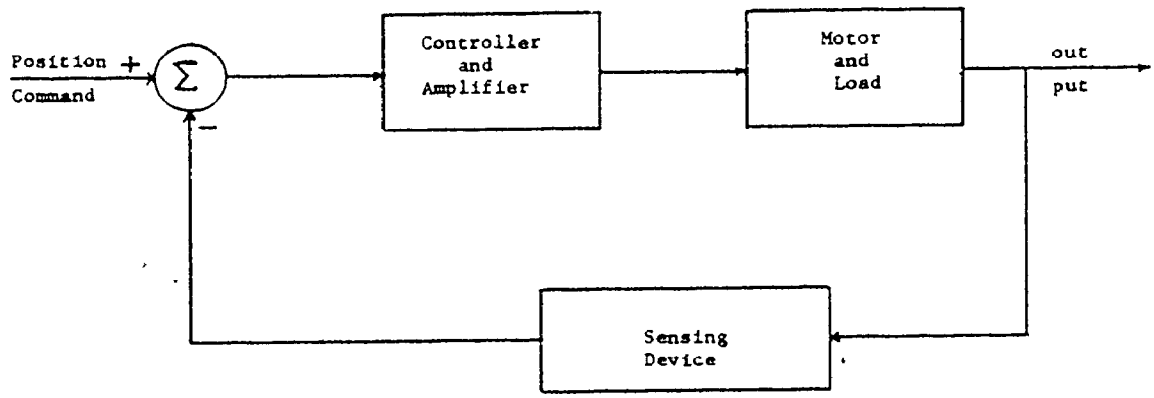


Figure 2.3 Closed-Loop Servo System

The response of the system now depends on the closed-loop configuration and as such it may be overdamped, underdamped or even unstable. In the design of the robot control system, the slave processor was put inside the position loop, as shown in Figure 2.4, therefore it was easy to achieve a critical damping, as is clear in Figure 2.5.

For the analysis of the position control and the Fortran simulation, a block diagram, such as the one shown in Figure 2.6, was used.

The system parameters are explained as follows:

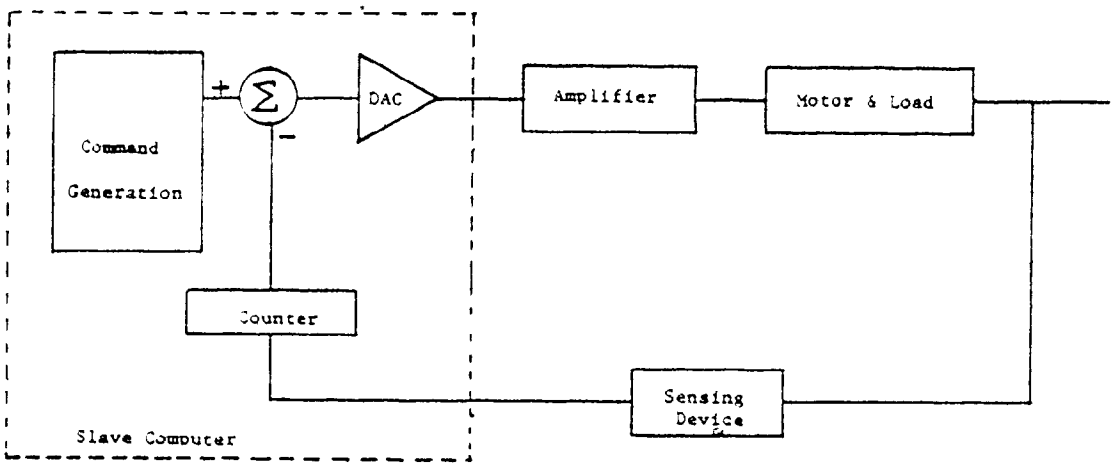


Figure 2.4 Computer Within the Positional Loop

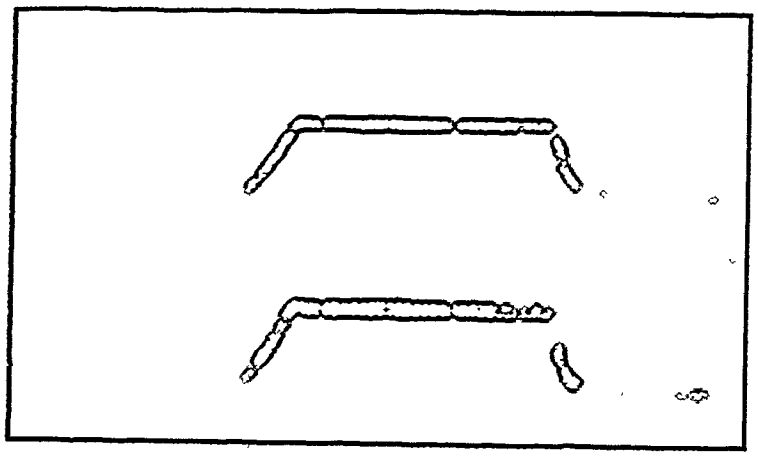


Figure 2.5 VELOCITY PROFILE

TOP: Command Velocity
BOTTOM: Actual Velocity

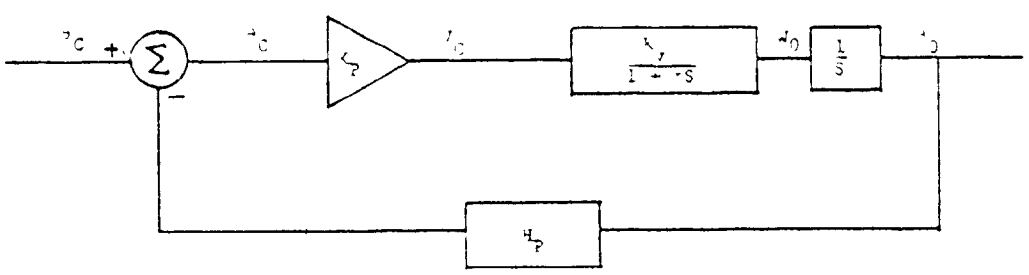


Figure 2.6 Approximate Model for One Joint of the Robot

Motor velocity is denoted by W_o and the angular position is θ_o, θ_c is the command position, also θ_e is the error in position which is equal to the difference between the command position and the actual position.

H_p is the transfer function of the positional feedback. The feedback device is the incremental encoder which gives 250 pulse/rev.

K_p is the coefficient of the 10-bit Digital-to-Analog Converter, DAC, which is (10/1023),

K_v is the gain of the velocity servo which is (1500 RPM/5V x 60 sec).

τ is the time constant of the velocity servo, which was measured as 50 msec.

The transfer function of the system is

$$G_p(s) = \theta_o(S)/\theta_c(S) \tag{2.1}$$

$$G_p(S) = \frac{K_p * \frac{K_v}{1+\tau S} * \frac{1}{S}}{1 + K_p * \frac{K_v}{1+\tau S} * \frac{1}{S} * H_p} \tag{2.2}$$

In order to analyze the system response, we look at the roots of the open-loop, which are the roots of the second term of the characteristic equations

$$K_p K_v H_p / S(1+\tau s) = 0 \tag{2.3}$$

Let $K = K_p K_v H_p$

$$K/S\ (ST+1) = 0 \tag{2.4}$$

The root locus for this equation is shown in Figure 2.7. Note that the system is inherently stable, and the gain margin is equal to infinity.

The above described position system is used in the Fortran Simulation of the trapezoidal profile.

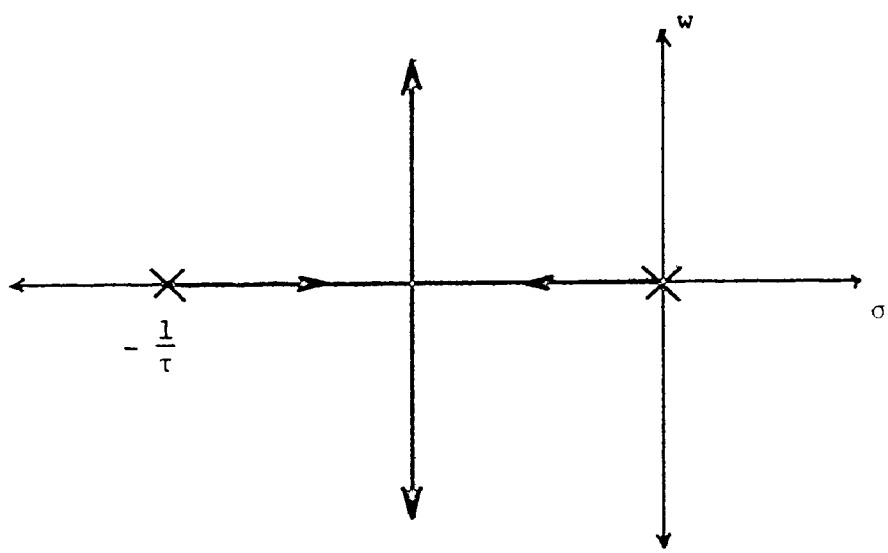


Figure 2.7 The Root-Locus for the Positional Servo System

CHAPTER 3

HARDWARE CONSIDERATIONS

This chapter briefly discusses the servo amplifier, servo motors, and position encoders. The detailed description of the hardware that was built for the control system and the interface is also discussed.

3.1 West-AMP, Servo Amplifier A6513-10E2

The West-amp is a transistorized pulse width modulated amplifier, which has an output of 1.1 HP. Output peak current of 22 Amp, and peak voltage of 100 VDC. It has a frequency response of 500 HZ minimum, and switching frequency of 5 KHZ. The input power is 120 VAC at 50/60 HZ, and signal input of 10 VDC. It has a built-in adjustable current limitation and compensation.

The unit has the most advanced safety features available today in the marketplace. There are a total of three West-amp units in the robot system. [20]

3.2 The Electro-Craft Servo Amplifier E586-BPC

The E586-BPC system is a bi-directional positional control system employing a high-gain, closed-loop electronic control. The input command can be an ungrounded 0 to 15 VDC. The motor angular

position is proportional to the commanded position. Velocity feedback is employed to control the travel rate while the motor is "homing in" on the desired position.

The position accuracy of the system is limited by the accuracy of the command and the position feedback signals. The system travel speed is controlled by velocity feedback from an integral tachometer and closed-loop servo. The standard E586-BPC controls the motor to run at 5000 rpm during travel to the "target" position.

The maximum output is 36 VDC and 3.3A. It also has a delayed torque adjust feature which can set the steady state torque over a range of 10 to 30 oz. in. The peak torque that can be obtained from the motor and the maximum drive current to the output stage can be adjusted. [21]

3.3 The Electro-Craft Servo Motors

There are two large families of DC motors, the integral horsepower types having a power rating of one horsepower or more, and the fractional horsepower motors, with power rating of less than one horsepower. The robot that we have, uses three motors of the first type and two motors of the second. However, all of the five motors are of the Permanent Magnet type.

Since the stator magnetic field of the Permanent Magnet motors is generated by permanent magnets, no power is used in the field structure. The stator magnetic flux remains essentially constant at all levels of armature current and, therefore, the speed-torque curve of

the (PM) motor is linear over an extended operating range.

The electro-craft motors which were used (3 of E703, and 2 of E586), share the following advantages:

1) Linear torque-speed characteristic.
2) High Stall (accelerating) torque.
3) No need for electric power to generate the magnetic flux.
4) A smaller frame and lighter motor for a given output.

3.4 The Position Feedback Transducers

Although various position feedback transducers could be used, digital shaft encoders are used as the position feedback devices. Their outputs are in the form of two pulse trains, one for each direction of motion (CW, CCW).

Dynamics Research Corporation (DRC) optical encoders, which provide high accuracy encoding and can be operated efficiently at a high speed, are used in this system. [22] The feedback resolution of the encoder is (500 pulse/rev).

3.5 The Master Computer

The master computer is the Single Board Computer "iSBC 86/12A" combined with the 80-bit Numeric Data Processor "iSBC 337" for high speed fixed and floating point functions.

The iSBC 86/12A Single Board Computer takes full advantage of Intel's LSI technology to provide self-contained computer systems. The

iSBC 86/12A is a complete system on a single 6.75 x 12.00 inch printed circuit board. The CPU, System Clock, read/write memory, non-volatile read-only memory, I/O ports and drivers, serial communication interface priority interrupt logic and programmable timers, all reside on the board. Full MULTIBUS interface logic is included to offer compatibility with the Intel OEM microcomputer systems. [23]

The iSBC 337 MULTIMODULE Numeric Data Processor offers high performance integer and floating point math functions to the iSBC 86/12A single board computer. The iSBC 337 module incorporates the Intel 8087 and because of the multimodule implementation, it allows on-board expansion on iSBC 86/12A boards, eliminating the need of additional boards for floating point requirements.

The iSBC 337 product consists of a small PC board containing the 8087 and 40 pin socket. The connection with the iSBC 86/12A board is accomplished by removing the 8086 CPU from its socket on the board, installing the iSBC 337 module in the socket previously occupied by the 8086 CPU and finally, installing the CPU in the iSBC 337's 40 pin socket. This arrangement allows the 8087 to operate as a co-processor to the 8086. [24]

3.6 The Slave Processor "Servo Card"

There is one servo card per axis. All servo cards are identical to each other and interchangeable by changing the board address switches. They all plug into a common bus which has the configuration shown in Figure 3.1. They are interfaced to the master computer by "MEMORY MAP-

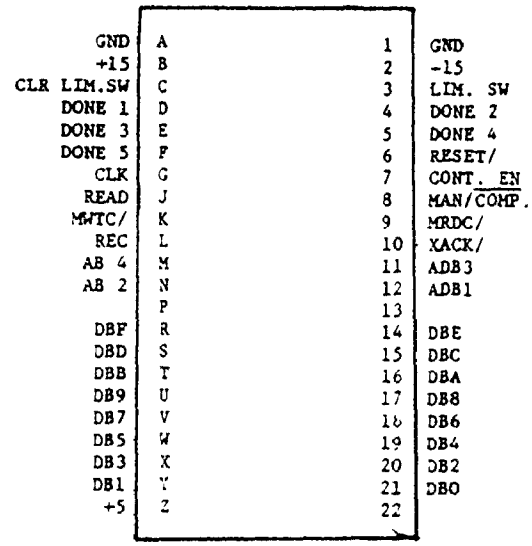


Figure 3.1: Common Bus Configuration

PED I/O" method, and each board is treated as one word in memory. The memory assigned to the control boards is 100nnH, where nn is a hexdecimal value between 0 and 1EH in increments of 2 (i.e. nn = OH, 2H, 4H, ..., 1EH), which means that the master computer can address up to 16 servo cards. The master computer has an address bus which is 20 bits wide, therefore it was necessary to encode the highest 15 bits of the address bus into one signal called Control Board Enable "CBE". "CBE" is sent out to the common bus and received by all servo cards. The hardware that generates this signal is shown in Figure 3.2 and the circuitry is MULTIBUS compatible (i.e. the address lines are low true logic).

It is best to partition the servo card into functional blocks as shown in Figure 3.3 and deal with each block separately.

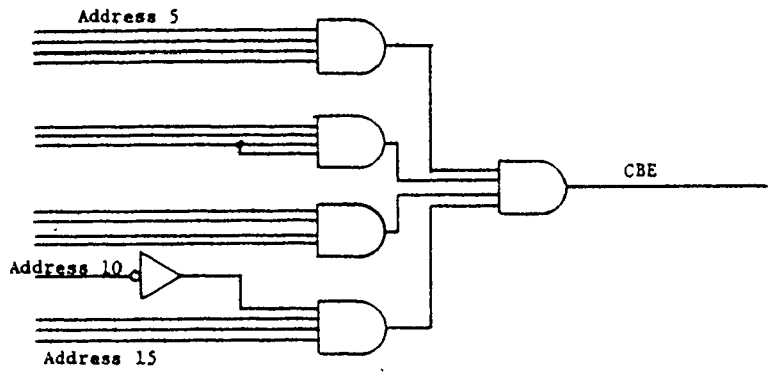
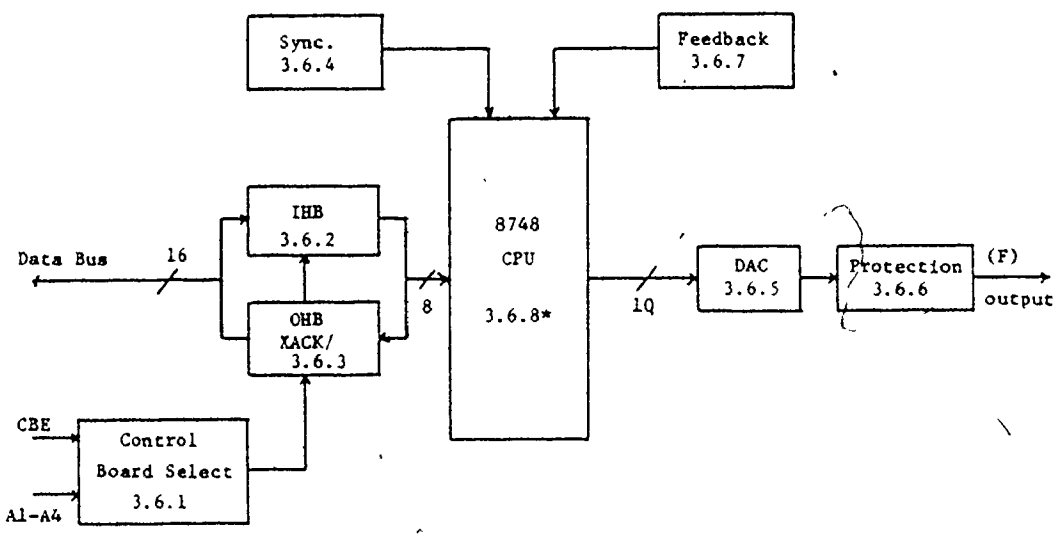


Figure 3.2 Control Board Enable



* 3.6.8 Refer to Section 3.6.8

Figure 3.3 Functional Block Diagram at the Servo Card

3.6.1 Control Board Select

Each control board has its own address which can be set by four SPST DIP switches. Any control board can be used to control any one axis provided that the address switches are set properly. Table 3.1 shows the board address with respect to the DIP switch setting.

When the master computer addresses a word in memory, it must be in the even memory bank, therefore the least significant bit (LSB) is always reset to zero. Since the upper 15 address lines are used to encode the "CBE" signal, this will leave only four address lines for addressing the control boards (A1 to A4). If these four address lines are matched to the DIP switches setting, then the board is selected for "READ" or "WRITE".

Figure 3.4 shows the four-bit magnitude comparator (SN74S85) that generates Control Board SELect signal "CBSEL" when enabled by "CBE" signal. The "CBSEL" signal is available (10 n sec) after the board address becomes valid on the MULTIBUS.

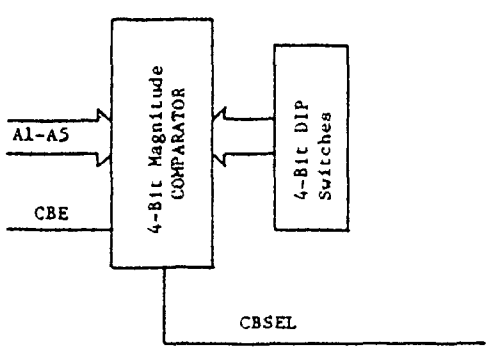


Figure 3.4 Control Board Select

Dip Switch Setting 4	3	2	1	Board Address in Hex
o	o	o	o	1 0 0 0 0
o	o	o	c	1 0 0 0 2
o	o	c	o	1 0 0 0 4
o	o	c	c	1 0 0 0 6
o	c	o	o	1 0 0 0 8
o	c	o	c	1 0 0 0 A
o	c	c	o	1 0 0 0 C
o	c	c	c	1 0 0 0 E
c	o	o	o	1 0 0 1 0
c	o	o	c	1 0 0 1 2
c	o	c	o	1 0 0 1 4
c	o	c	c	1 0 0 1 6
c	c	o	o	1 0 0 1 8
c	c	o	c	1 0 0 1 A
c	c	c	o	1 0 0 1 C
c	c	c	c	1 0 0 1 E

o - open
c - closed

TABLE 3.1

3.6.2 Digital Data Input

As mentioned previously, each control board has an INTEL8748 CPU which has data bus of 8 bits, the master computer's data bus is 16 bits wide. To interface these two data buses, an Input Holding Buffer "IHB" is required which consists of two 8-bits tri-state latches, INTEL 8212. When the master computer wants to write a word to one of the slave processors, it would put the 16 bits of data on the MULTIBUS and address the proper slave. During the memory write "MW" cycle, a Memory Write Command signal "MWTC/", is generated on the MULTIBUS and is used to load the data word in the "IHB" of the selected control board.

Figure 3.5 shows the hardware required to achieve the above function. Note that the latches must be tri-state, because their outputs are connected in parallel, directly to the 8748 data bus. Only one 8-bit latch chip is enabled by the software at any given time.

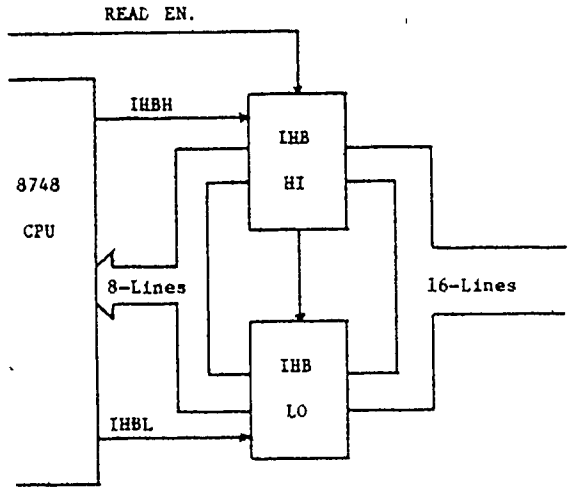


Figure 3.5 Digital Data Input

3.6.3 Digital Data Output and Transfer Acknowledge

Once again, we have the problem of interfacing between the 8-bit and 16-bit data buses. An Output Holding Buffer "OHB" is used to interface the two data buses. The 8748 CPU would write two bytes, one at a time, to the holding buffer to assemble one word, and then the master computer will read one word using the Memory Read Command signal "MRDC/".

During Read and Write cycles the master computer expects a Transfer Acknowledge Signal "XACK/" from the peripheral which it is communicating with. If this signal is not present, the master CPU will wait for 100 ms before it executes the next step in the program. For fast execution time, "XACK/" is generated in each servo card.

Figure 3.6 shows the circuitry required for the digital data output and transfer acknowledge signal generation. The "OHB" is put into low impedance to the MULTIBUS when the board is selected and, "MRDC/", is present. "XACK/" is generated when "MWTC/" or "MRDC/" is available but its tri-state buffer is disabled unless the "CBSEL" is true, then "XACK/" is put on the MULTIBUS.

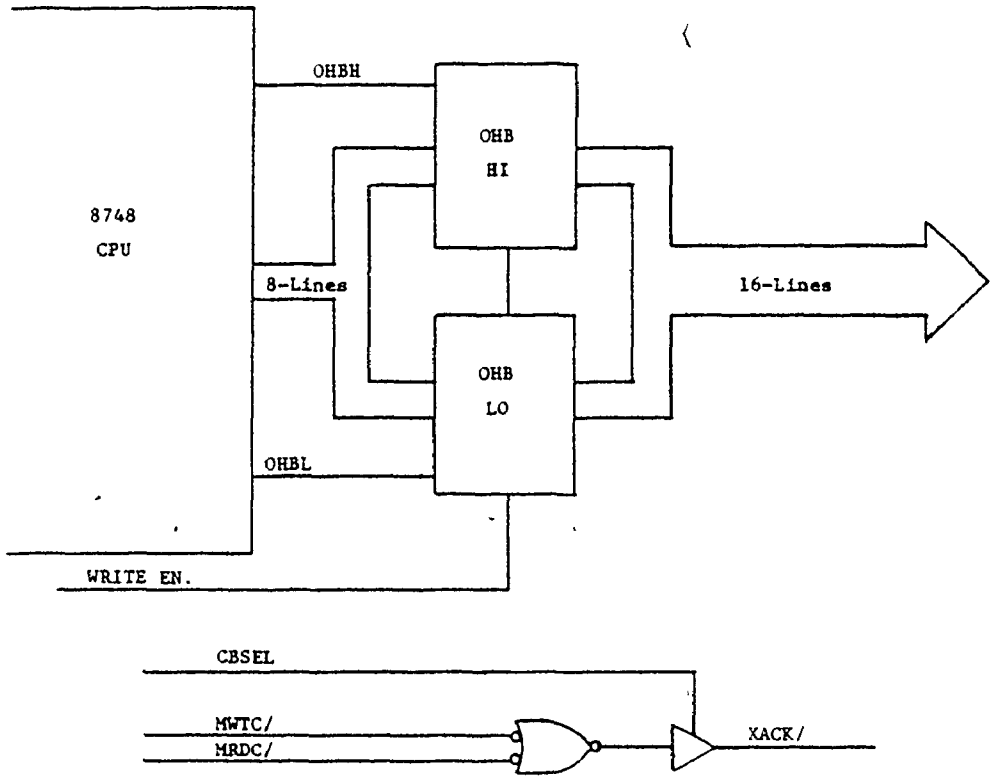


Figure 3.6 Digital Data Output and Transfer Acknowledge

3.6.4 Synchronization

To synchronize the Master Computer to all the slave processors, each servo card must have five synchronization signals.

1) Reset in: This signal is issued by the master computer. When received by the slave processor, the software resident in the 8748 EPROM starts execution at address OH. The duration of the Reset-in is (1 μs).

2) DONE: This signal is issued by the slave processor and received by the master computer. It has two basic functions depending on the time that it occurs in.

   a) When the slave CPU is reading data from the Master, this signal is a 5 us pulse indicating that the "IHB" is empty.

   b) When the slave has finished the job that was assigned to it, this signal is used to inform the master computer of the completion of the job.

3) CLOCK: The clock is common to all the slave processors and is used to close the positional loop in each servo card. It is also used as a timing loop for the execution of the trapezoidal velocity profile. The period of the clock is 4 ms.

4) READ: The read signal is generated by the Master Computer. It is issued when the master computer has loaded one word in the "IHB" of each slave. To the slave CPU, this signal means "IHB" full. The duration of the Read signal is (1 μs).

5) RECORD: This signal is common to all slave CPU's and is generated by the manual controller/teach pendant. It could be detected by

the slave CPU only when in "TEACH" mode and ignored at any other time.

Figure 3.7 shows the circuitry required for synchronization. Note that "CLOCK", "READ", and "RECORD" signals are interfaced in the slave CPU by handshaking method, (i.e. the signal is latched and will not be cleared until acknowledged by the CPU).

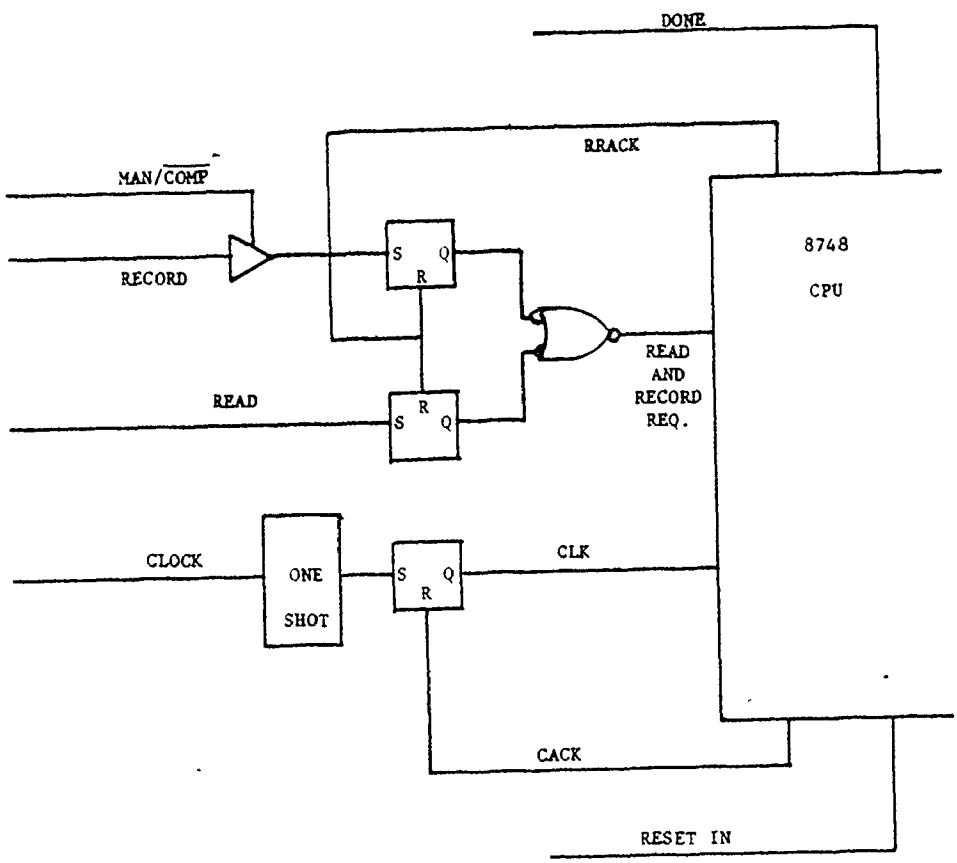


Figure 3.7 Synchronizing Signals

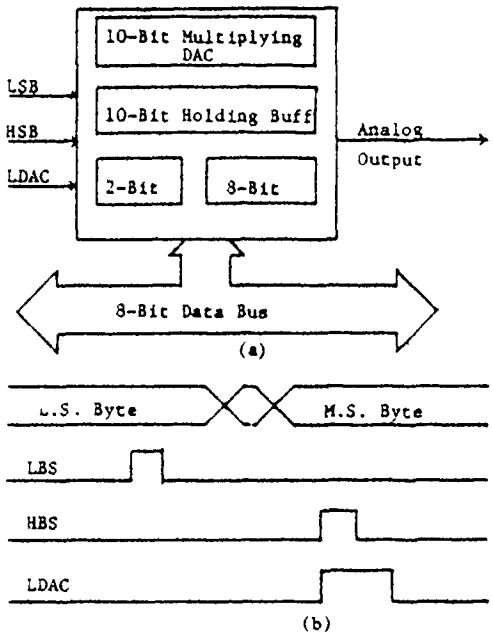
3.6.5 Analog Signal Output

When the position error is established, it has to be converted to a velocity signal to drive the servo motor to null that error. The error is a 16-bit digital word, the least 10 significant bits of it is converted to its analog equivalent using a 10-bit Digital-to Analog Converter, DAC. The Analog Devices 10-bit buffered multiplying DAC AD7522 is used which has an input buffer and holding register. The input buffer is used to assemble the 10-bit data from the 8-bit data bus by loading two bytes, one 8-bit and one 2-bit. Subsequently, it moves this data to a 10-bit holding register where the digital word is converted into an analog current.

Figure 3.8a shows the block diagram of the AD7522 and Figure 3.8b shows the timing diagram for two byte parallel loading of the DAC. Figure 3.9 shows the circuitry required to support the DAC in the bipolar operation mode. Table 3.2 shows the bipolar code. Note that the digital input code is an "OFFSET BINARY" form, and the generated error word is in "TWO'S COMPLEMENT" code, therefore, a code conversion is needed.

The most significant bit of the error word was inverted before the input to the DAC. This gave us a two's complement form. Resistors R1 and R2 are used in the gain adjust, R3 is used in zero adjust. The Schottky diodes CR1, CR2 are used to clamp the amplifier input to -300 mV if they attempt to swing negative during power up or power down. The silicone diode CR3, and the shottky diode CR4 are only for protection purposes if VCC exceeds VDD. The monostable multivibrator is

used to generate the pulse that loads the holding register from the input buffer, "LDAC". The "LDAC" pulse width is about 800 ns.



Digital Input	Analog Out
1111111111	$-V_{REF}(1-2^{-9})$
1000000001	$-V_{REF}(2^{-9})$
1000000000	0
0111111111	$V_{REF}(2^{-9})$
0000000001	$V_{REF}(1-2^{-9})$
0000000000	V_{REF}

TABLE 3.2

Bipolar Code Table

Figure 3.8(a) 2-Byte Parallel Loading of the DAC (b) Timing Diagram

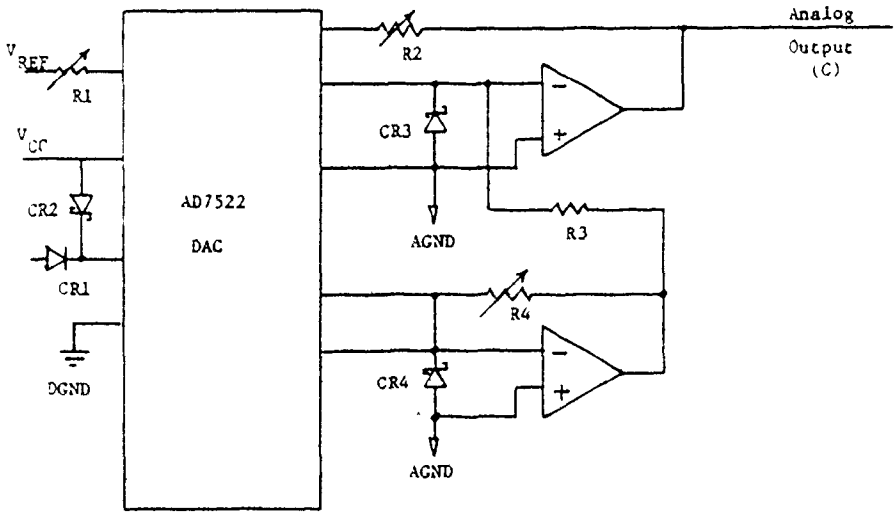


Figure 3.9 Digital-to-Analog Converter, Bipolar Operation

3.6.6 System Protection and Manual/Computer Selection

This part of the servo card is responsible for selecting the analog output from the manual controller or the computer, also it switches from computer control to manual control when an emergency stop is required. When any limit switch is set, this part of the hardware will switch to manual mode, and the robot will come to stop with controlled deceleration. The operator will then move the robot manually to clear the fault, and then reset the limit switch.

Consider Figure 3.10a, the analog output "F" is equal to either the computer command "C" or the manually generated signal "M". It is equal to "C" when the computer mode is selected and none of the limit switches is set, and is equal to "M" if the manual mode is selected or any limit switch is set. Figure 3.10b shows the circuit for this function. Note that there are three control signals, namely: manual/computer select, limit switch set, limit switch clear, that direct the analog signal flow. When any limit switch is set it will be latched until cleared by the operator. There are also three analog signals, namely: "M", "C", "F" in this figure.

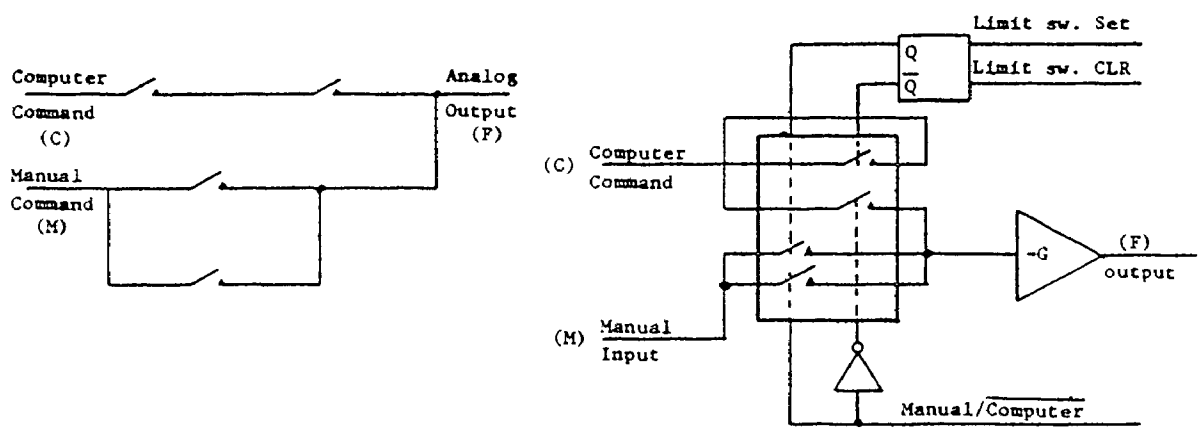


Figure 3.10(a) Computer/Manual Select (b) Protection and Select

3.6.7 The Positional Feedback

The positional feedback is achieved by discriminating between the pulses from the two lines (CW, CCW) of the incremental encoder. The pulse width of the encoder is typically 1 μs therefore a 150 ns (470Ω, 330 pF) low-pass filter is put on each line as shown in Figure 3.11. The signals are shaped and buffered to TTL level with threshold of 1.3 V by the voltage comparator. The DRC encoders have a 500 pulse per shaft revolution. This hardware divides this signal by 2 to give 250 pulse per revolution and each pulse is shaped by the monostable multivibrators. An 8-bit up/down counter is used to discriminate between the "CW" and "CCW" lines with the MSB used as the sign bit. A two's complement number results from this process and it is between -2^7 to 2^7-1 (-128 to 127).

From the encoder specifications, the shaft speed can be as high as 5000 RPM at 10% duty cycle. This angular speed and the 250 pulse per revolution will give a maximum frequency of (5 KHZ) on the UP/DOWN inputs of the 8-bits UP/DOWN counter. This will allow a sampling rate of the position feedback to be as low as 160 Hz. However, a sampling rate of 250 samples per second is used for smoother control and higher stability. The UP/DOWN counter is gated to the data bus of the 8748 by a tri-state octal latch, when it is a time for reading the positional feedback the CPU will issue a Read Feedback "RFB/" signal which is used to latch the contents of the Counter, put the contents of the octal latch on the 8748 data bus, and clear the contents of the counter. The 8748 CPU will then read a valid feedback data.

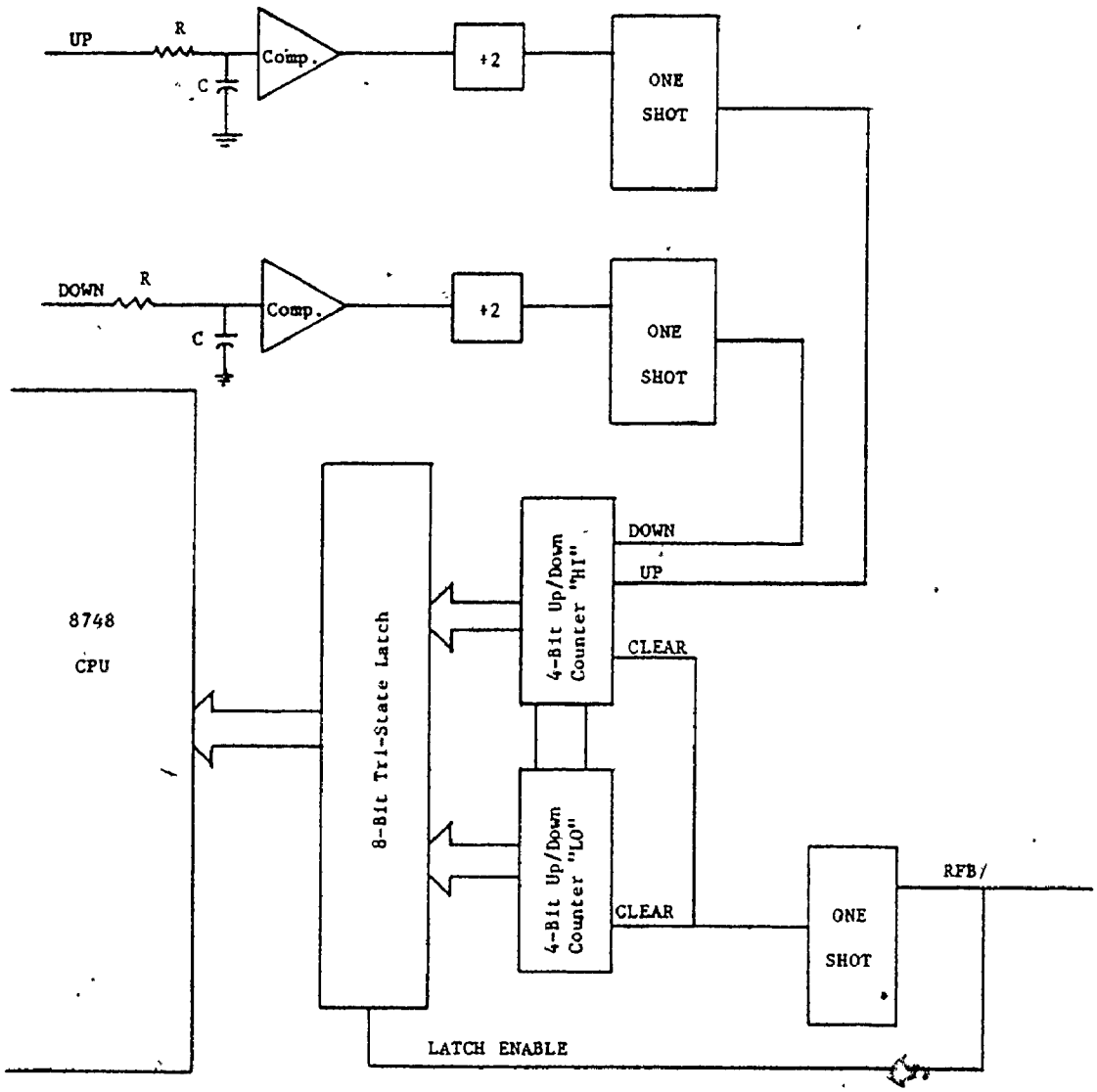


Figure 3.11 The Positional Feedback

3.6.8 The Central Processing Unit (CPU)

The Intel 8748 is a totally self-sufficient, 8-bit microprocessor. It has a 2.5 us cycle time, and over 90 instructions. It is 40-pin dual in line package (DIP), 27 of which are Input/Output lines (three 8-bit ports, and 3 test points). It has a 1k byte of user programmable and erasable "EPROM" Program memory, and 64,8-bit, bytes of Random Access Memory (RAM).

The CPU is interfaced to the hardware on the servo card in two sections, as shown in Figure 3.12, INPUT section and OUTPUT section. The INPUT section consists of the test points T0 and T1, Reset in, and the Data bus. T0 is used to detect the start of the timing loop and start execution of the control algorithm. T1 is used to sense the "READ", and "RECORD" signals (mentioned before), Reset is used to synchronize all slave processors.

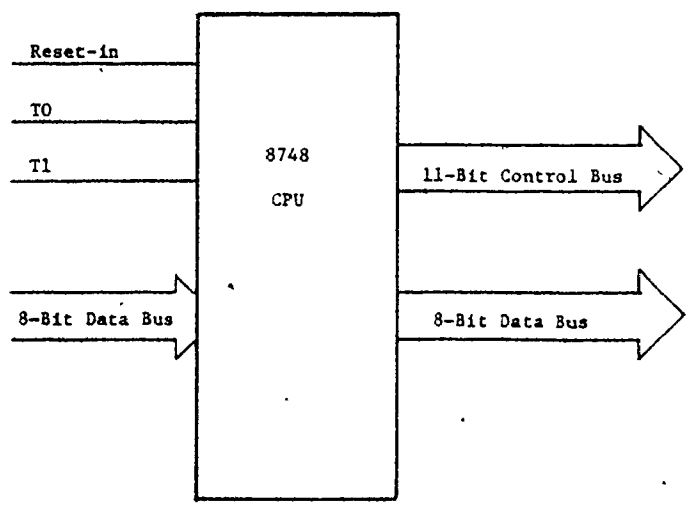


Figure 3.12 The CPU Interface on the Servo Card

The data bus is an input/output bus used when reading data from the master computer, reading the positional feedback, output the commanded position to the DAC, and output the joint positions to the master computer in "TEACH" mode.

The OUTPUT section consists of the data bus mentioned above, and the control bus which has the following functions.

- Two lines "IHBL", "IHBH", are used for enabling the Input Holding buffer when it's being read.
- Two lines "OHBL", "OHBH", are used to load the Output Holding buffer.
- Two lines "LSDACL", "LSDACH", are used to Load and Start the Digital-to-Analog Converter.
- Two lines "RRACK", "CACK", are used to acknowledge the "READ", "RECORD", and "CLOCK" request.
- One line is used for "DONE" signal.
- One line is used for reading the feedback, "RFB/".
- One other line "POS" is used when the system is in manual/teach mode, to indicate whether the joint is put in close positional loop or is moving under velocity control only as described in Section 3.7.

The MCS-48 Microcomputer design aid "PROMPT 48" was used in testing the prototype, therefore port one (P10-P17) is used as the bi-directions data bus in the servo control card for compatability, ports zero and two are used for the control bus.

3.7 The Manual Controller

When the manual controller is selected, it can generate a velocity command with or without controlled acceleration, deceleration. Figure 3.13 shows two first-order integrators to control acceleration and deceleration, the time constant of these integrators is 0.4 second and the gain can be adjusted between .1 and 1.3. Three of the five joints can take a very high acceleration and deceleration, therefore no control on these parameters is needed. Figure 3.13 has three amplifiers used only for gain adjust, they could have a gain between .3 and 3.75.

The output of the amplifier is connected to the manual control input, as shown in Figure 3.10b and described in section 3.6.6. The input is supplied by the adjustable, regulated power supplies (±5V). shown in Figure 3.14, and controlled by the analog switches of Figure 3.15.

Figure 3.14 shows the dual regulated power supply, resistors R1A, R1B are mechanically linked together so that the percentage of V_{max} is the same for positive and negative voltage, (forward and reverse velocity command, respectively). Switches PSW1 and PSW2 are analog protection switches. They are normally closed and will open if one of the limit switches is activated, and the robot cannot be commanded to move. The flip-flop is used to latch the fault condition and the open collector inverter is used to drive the LED ON until the fault is cleared by the operator.

The manual controller has five DPDT center-off Command Switches,

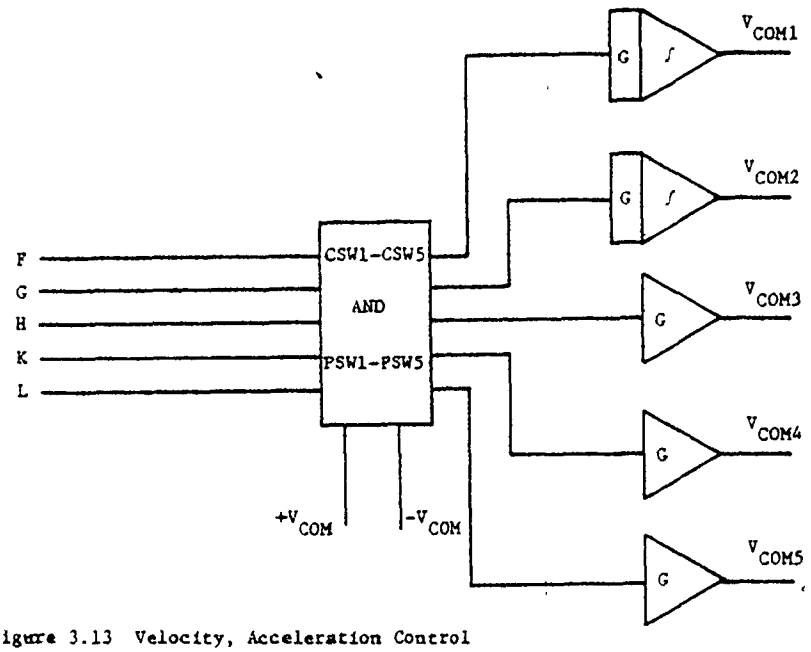
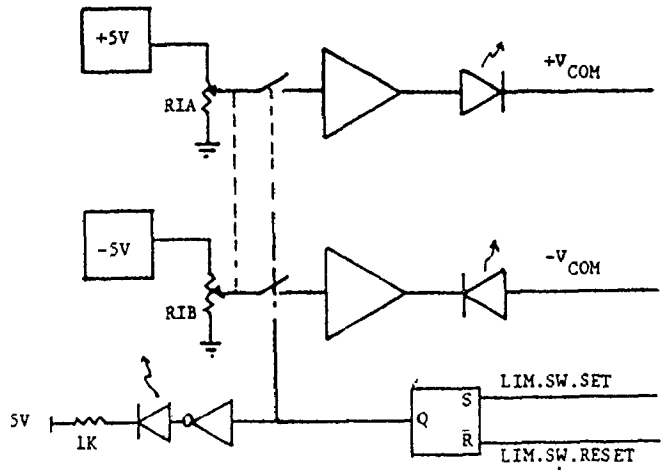


Figure 3.13 Velocity, Acceleration Control

Figure 3.14 Command Velocity Generation



(CSW1 - CSW5), each command a velocity to one joint of the robot, however, under manual controls only one joint is permitted to move at any point in time. Figure 3.15 shows the circuitry required to move only one joint and lock out the other four joints in closed positional loop. Labels A to E are the center tap of the command switches. One pole of each switch is used to control the position loop (open or closed), and the other is to pass the velocity command to the servo card. Initially, all Analog Switches (ASW1 - ASW5) are closed, and all the command switches (CSW1 - CSW5) are in the center-off position. If any command switch (for instance, CSW1) is depressed, its corresponding analog switch, namely, "ASW1" will stay closed. However, "ASW2" to "ASW5" are now open and switches (CSW2 to CSW5) are disabled.

The signals "POS1" to "POS5" are used to open or close the analog switches, they are also used to indicate whether the position loop of its corresponding joint should be open or closed. This is to say that if "CSW1" is depressed, "POS1", will be HI, "ASW1" is closed and the servo card which corresponds to "CSW1" is under velocity control only. Also "POS2" to "POS5" are LOW, "ASW2" to "ASW5" are open and the other servo cards are in closed positional loop.

Since the "CSW2" to "CSW5" are disabled, they will be ignored if pressed by the operator at this time.

Each command switch can be activated in the positive or negative direction. The commanded direction is indicated by one of two LED's, as shown in Figure 3.14.

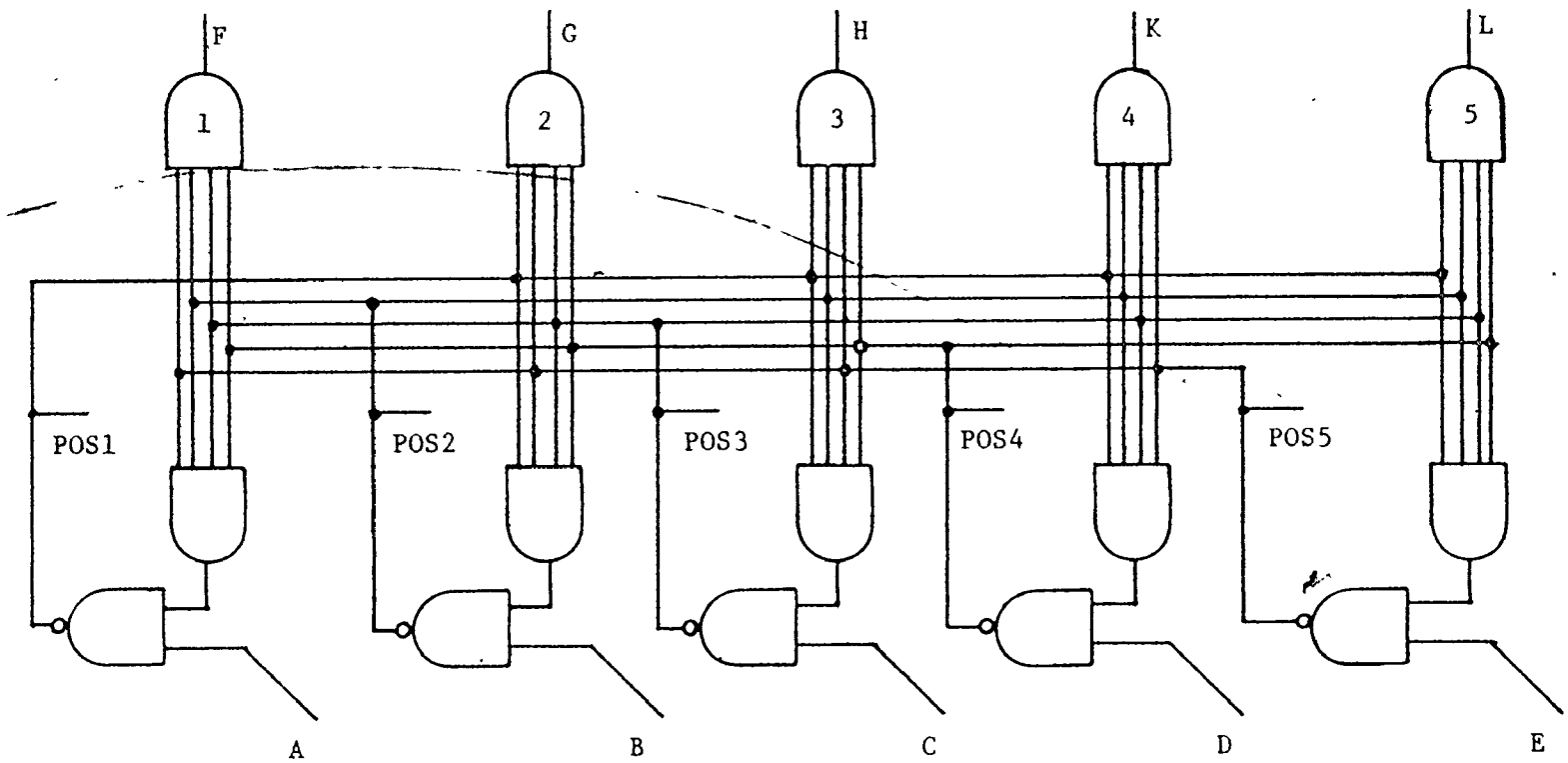


Figure 3.15 Generated Command Controller

RECORD, SINGLE-STEP, WAIT, GRIPPER-STATUS, are also implemented in the manual controller. "RECORD" is normally a high signal, as shown in Figure 3.16a, it is pulled low by the operator when it is time to record the joint positions. This signal is latched on each servo card.

"SINGLE-STEP", is another function performed by the manual controller. When the system is in the single-step mode, it can execute one instruction at a time of a given program. Figure 3.16b shows a simple circuit, the LED is ON when the computer is ready to execute a motion between two previously taught points, and it goes off when the robot is moving between the points.

Figure 3.16c shows the circuitry needed to generate the wait for input or output signal. DPDT center off switch is used to indicate via the LED whether there is a wait state or not, and also to generate the wait signal. Along with this switch, there is a thumbwheel switch for selecting I/O ports, there are four input and four output lines. The status of the GRIPPER is set by one switch, as shown in Figure 3.16d. it can be open or closed at the time of recording the joint position.

As was mentioned earlier, the task of teaching the robot with the teach pendant is done by moving the end-effector from one point to another. Therefore, it is possible to follow a continuous path by a large number of small P-T-P intervals. The points are identified as Mid-Point which will indicate to the computer at playback time, that the end-effector should not come to stop at these points. The velocity between two adjacent line segments, however, must be changed under controlled acceleration or deceleration, as was described in section 1.5.4. The last programmed point in the path must be identified as a Terminal

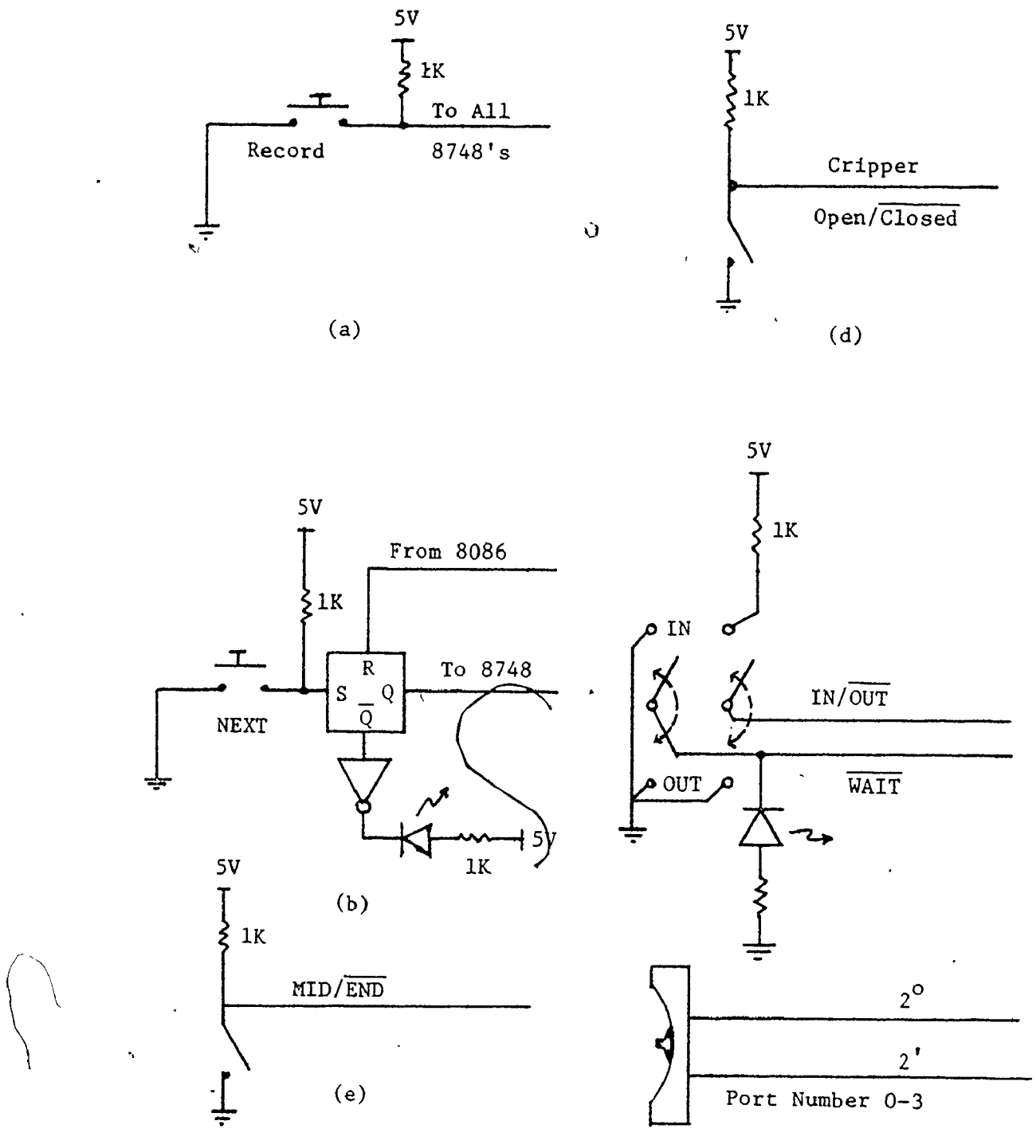


Figure 3.16

a) Record signal generation
b) Single-step
c) Wait state and Port Num. generation
d) Gripper status
e) MID/END Point generation

Point so that the end-effector comes to a complete stop when it reaches the last point. Figure 3.16e shows the switch that gives the Terminal/ Mid. point on the manual controller/teach pendant.

Note that there is no need for latching or debouncing the "WAIT", "I/O" channel number, "GRIPPER-STATUS", or the "TERMINAL/MID." point indicator, because they should be set before pressing the record button.

CHAPTER 4

SOFTWARE CONSIDERATIONS

The master computer was programmed with the high level language, PL/M-86 [25] [26]. The slave computer was programmed with assembly language, ASM48 [27].

Modularity was used in building up the system software. The system contains four modules as shown in Figure 4.1. Each module is a compilable program. In the future, when more functions and algorithms are added to the software, only the main module requires updating to support the new modules.

The main module initializes the hardware, and contains the command dispatcher which can transfer control to the monitor command module or the robot command module. It can communicate with the input/output module and the utility module.

The input/output module supports the communication protocols between the operator and the system, and between the system and the robot. The procedures in this module can be used by the main module and the command modules.

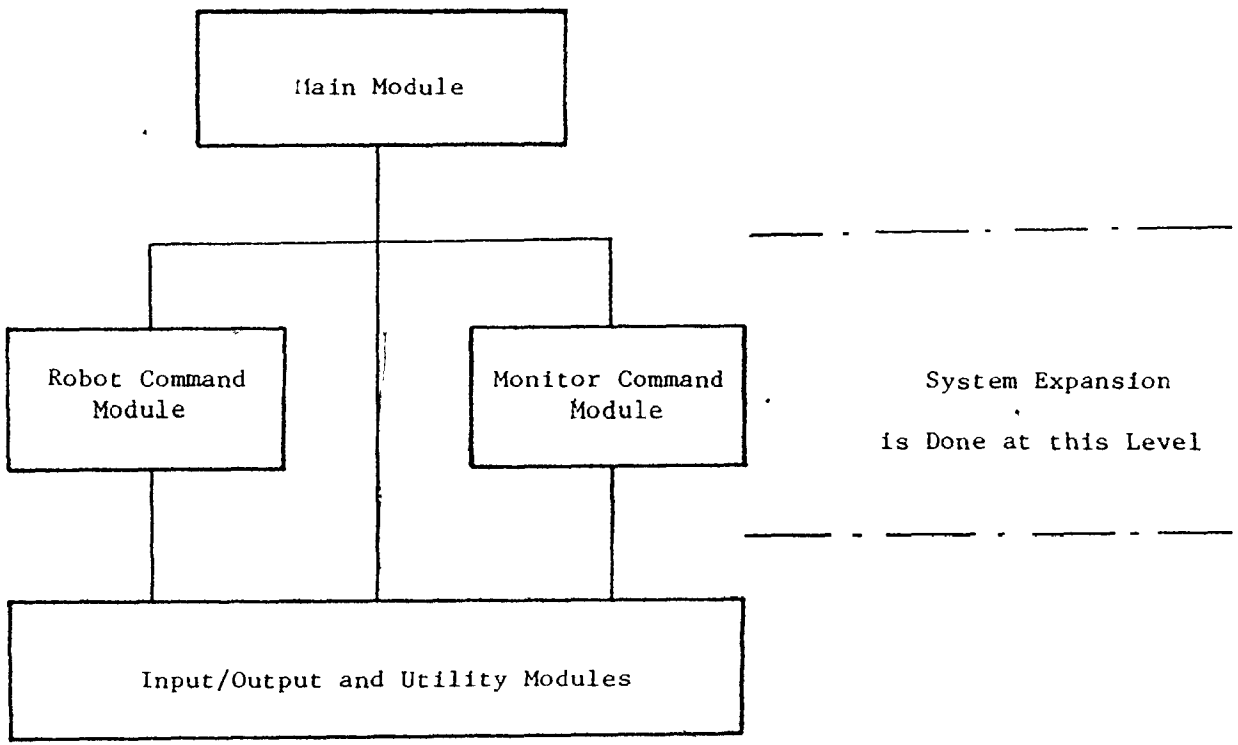


Figure 4.1 System's Software Organization

The utility module is used by the higher level modules. Some of the functions that this module can perform is to check for valid command or valid input data.

The Monitor Command module supports the master computer. It can be used to examine and substitute memory, examine and modify registers, display memory, move memory, and execute a user program at a given address. For details of the monitor, see Appendix D.

The Robot Command module tests the point-to-point control system used with the robot, it contains the following commands:

RESET$8748

This procedure initializes the servo cards, and starts the execution of the control program in the 8748 microprocessor.

READ$FROM$BOARD

This procedure reads the output holding buffer of the servo card, and outputs it to the CRT. It informs the operator with the position or velocity of the end-effector when requested.

WRITETOBOARD

This procedure writes two words, BOARD$BUFFER$0 and BOARD$BUFFER$1 to the servo card.

BOARD$BUFFER$0 is at location 700H.

BOARD$BUFFER$1 is at location 702H.

FORWARD$RUN

This procedure tests the repeatibility of the motion command, it moves the contents of FORWARD$BUFFER$0 and FORWARD$BUFFER$1 to BOARD$BUFFER$0, and BOARD$BUFFER$1, respectively. It then calls the WRITETOBOARD procedure and starts executing the motion.

FORWARD$BUFFER$0 is at location 704H.

FORWARD$BUFFER$1 is at location 706H.

FORWARD$TEST$LOAD

This procedure accepts input from the keyboard, and stores it in FORWARD$BUFFER$0 and FORWARD$BUFFER$1. The input is a distance and a velocity command. It then outputs the command to the servo card.

REVERSE$TEST$RUN and REVERSE$TEST$LOAD have the same functions as the previous two procedures but they use the memory location 708H for REVERSE$BUFFER$0 and 70AH for REVERSE$BUFFER$1.

The PLM/86 is programm is listed in Appendix A.

The Servo Control

This control program is resident in the program memory of the Intel 8748 microprocessor. It generates the TRAPEZOIDAL velocity profile. The program is written in assembly language (ASM48). The algorithm was developed and simulated in FORTRAN. The simulation results are shown in Figure 4.2.

P_{COM} is the command position in basic motion units.

P_{ACT} is the actual position in basic motion units.

V_{COM} is the command velocity expressed as an increment in distance per DT.

This algorithm was desdribed previously in section 1.5.4.

Figure 4.3 shows the flowchart, the numbers on the flowchart are described below:

(1) At reset or power up, the program starts by initializing the control port, setting the command distance and the increment velocity to zero.

(2) It then checks for input holding buffer full signal, it will read the IHB if it is full, or it will execute the program with zero command distance. The IHB consists of the total distance, XEND, and the maximum velocity, VMAX, at which the joint should move.

(3) It also loads the Output Holding Buffer, OHB, with the actual position, so that it would be displayed on the CRT up on the operator request.

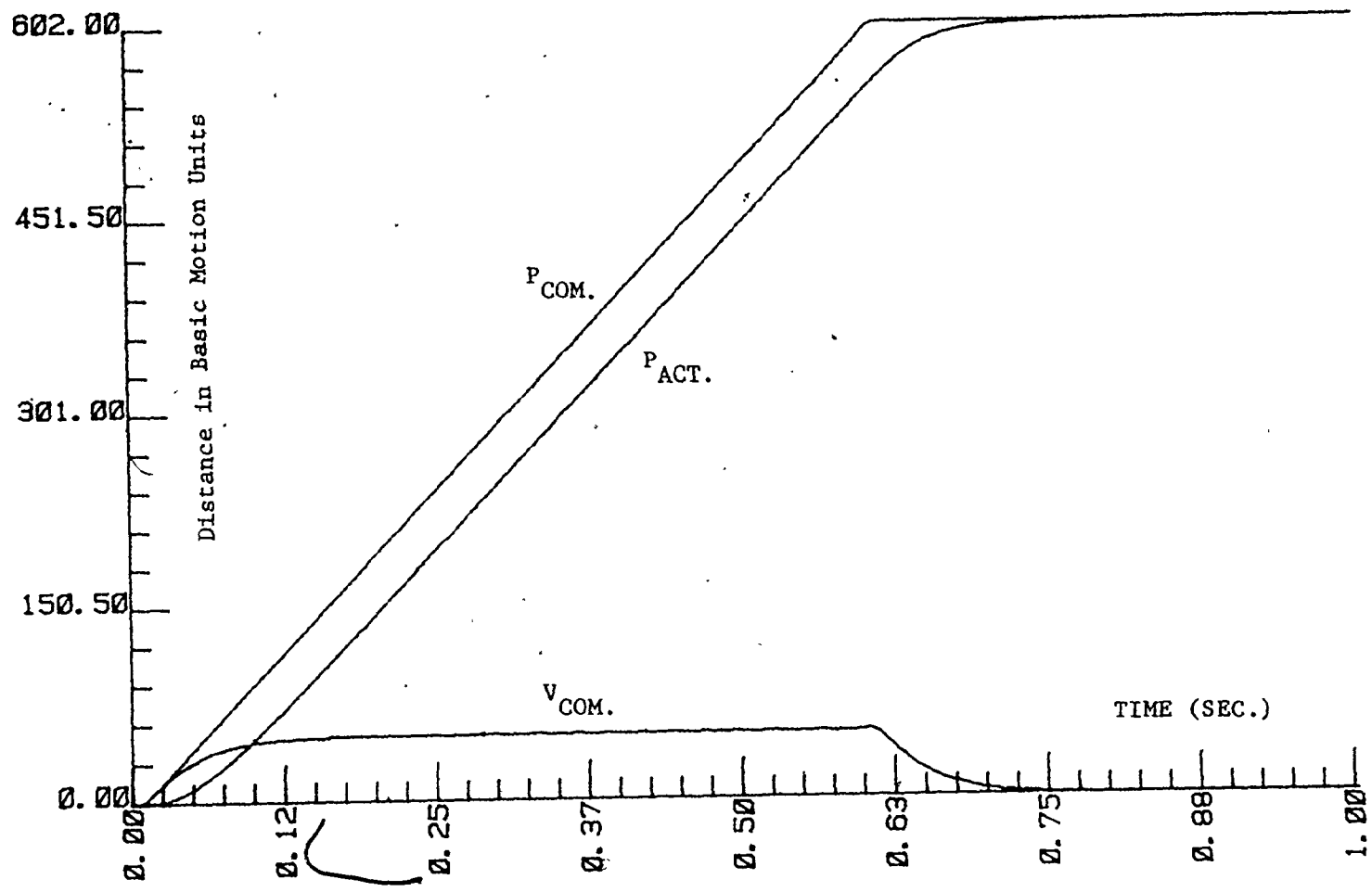


Figure 4.2 Plot of the Fortran Simulation for one Joint of the Robot

(4) The processor will wait for loop trigger signal which occurs every 4 ms.

(5) When the loop is started, the processor will issue a read feedback signal "RFB/" to get the actual position from the encoder.

(6) IFL, is a flag used to indicate the completion of motion.

(7) X_{COM} is then generated by either adding or subtracting the incremented velocity, DX, depending on the direction of motion. (8) At this point, the program checks if the acceleration part is completed, when it is, (9) The program enters the RUN part. (10) The Auxiliary parameters, X1, X2, and XSP are set to their proper values.

During the execution of the run part, V_{COM} is constant (11) and the program checks for the beginning of the deceleration part. (12) When the deceleration part begins, V_{COM} is decremented to reach zero at the end of the motion.

(13) When the motion is completed, the DONE signal is issued to inform the master computer the completion of the motion. The positional loop is kept closed to correct for any drift in the servo motors.

The assembly program is listed in Appendix B.

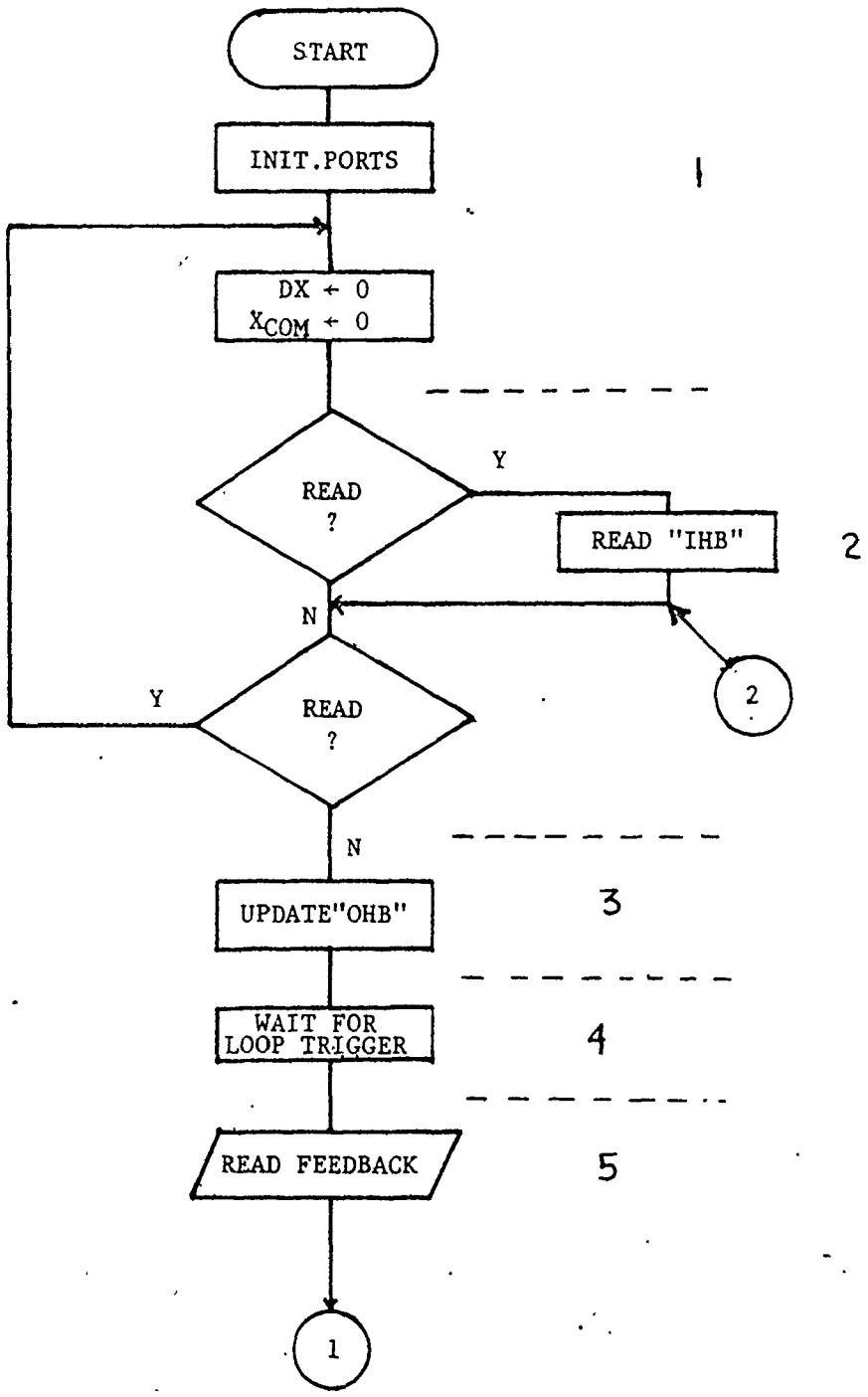


Figure 4.3 Control Program Flowchart

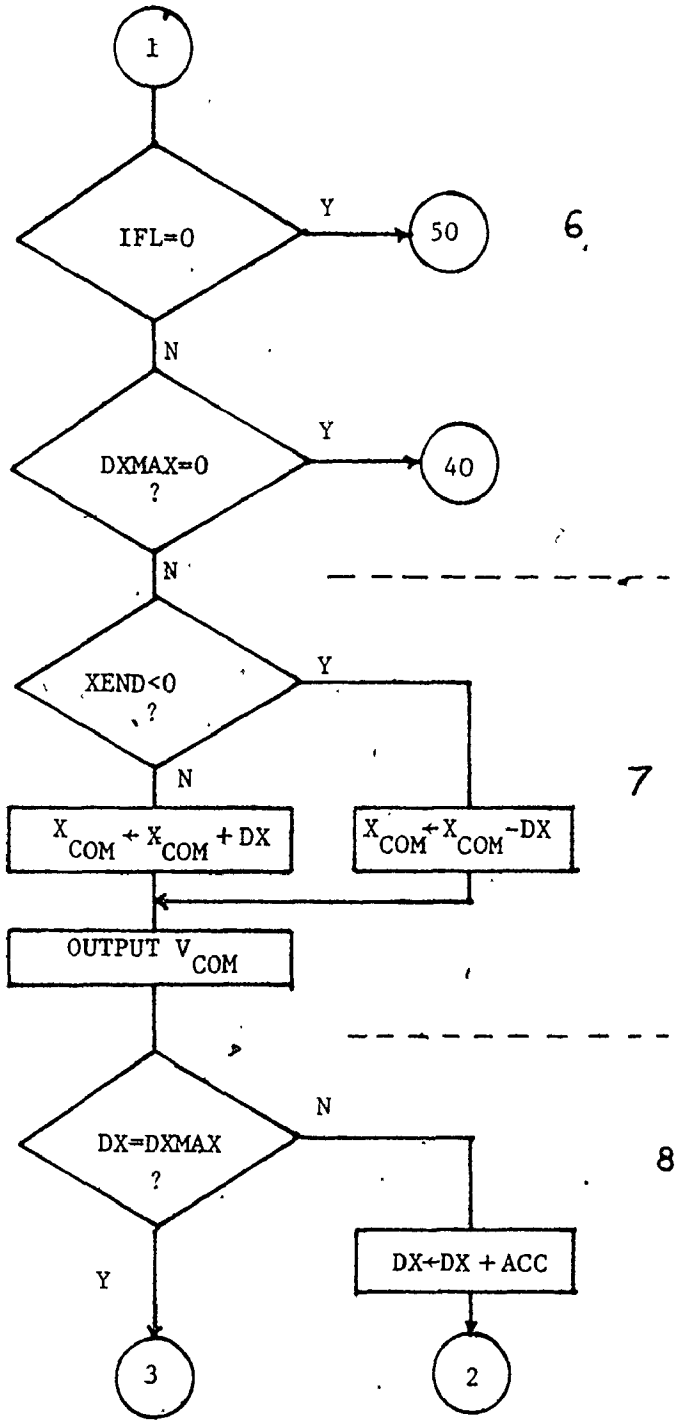


Figure 4.3 (continued)

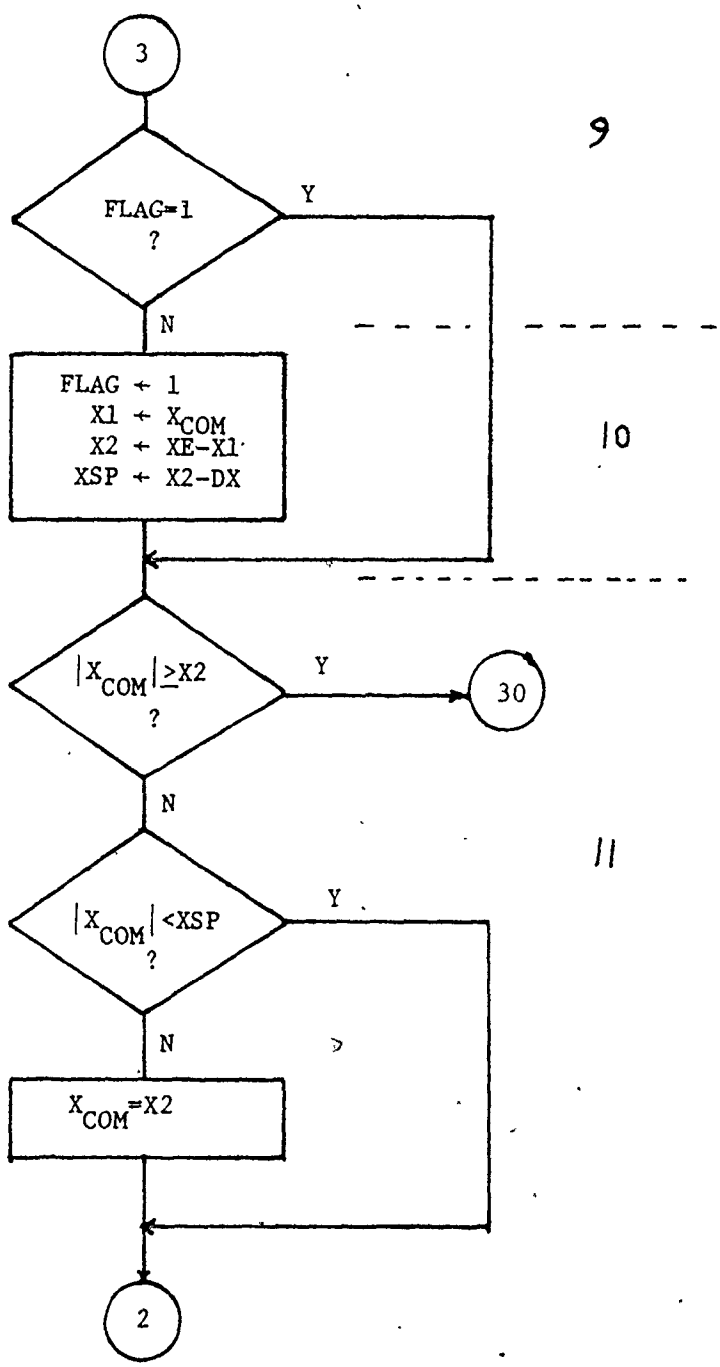


Figure 4.3 (continued)

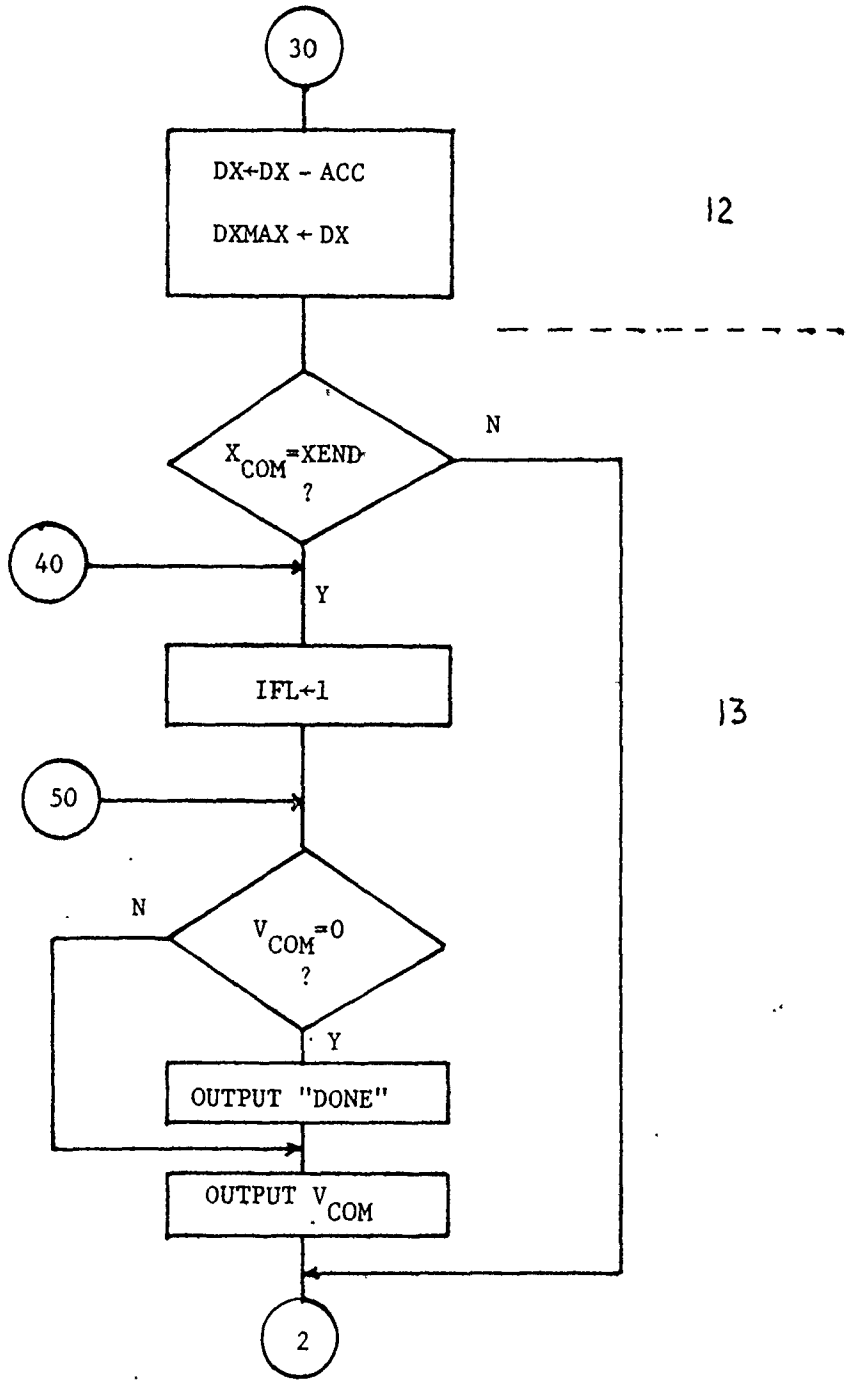


Figure 4.3 (continued)

CHAPTER 5

CONCLUSION

Industrial robots can be either servo or non-servo controlled. Those which are servo controlled are more advanced and can do highly complicated tasks.

The state, position and orientation, of the end-effector can be defined at the two end points in point-to-point control, or at any time in continuous path control. The P-T-P motion is fast and the shape of the path generated is unpredictable. Motion along a straight line is desired for reasons such as to permit a reduction in the inertial load in the rotary menipulator, however, the command generation for this kind of control is more complicated than the one for P-T-P control.

Command generation is carried out by software in a fixed time interval. Cartesian interpolation, joint interpolation, cubic spline interpolation and a constant acceleration/constant velocity technique may be used to smoothen the motion and increase efficiency.

Most industrial robots can be programmed either by the on-line teach method or off-line programming. Teaching the robot may be accomplished by manually controlling a motion from the starting point to the end point, one coordinate after another. Off-line programming can be done on a different computer, keeping the robot running in the field while the joint positions are being generated remotely.

There are advantages and disadvantages in both programming methods.

The control system of the modified UNIMATE 2000 has as its basis the concept of a horizontal decomposition scheme. Each joint has a separate processor and one master computer is responsible for the command generation and joints coordination.

Functional and physical modularity was achieved by the distributed system. The approach taken was to keep as much of the control system as possible independent of the particular robot configuration.

The software is partitioned into a three-level hierarchy. The lowest level is responsible for the interfacing of the servo motors and the control system, while the highest level interfaces the operator to the system.

APPENDIX A

PLM/86 PROGRAM LISTING

```
PL/M-86 COMPILER        MAIN$MODULE                                    01 MAY 81  PAGE   1

ISIS-II PL/M-86 V2.1 COMPILATION OF MODULE MAINMODULE
OBJECT MODULE PLACED IN :F1:MAIN.OBJ
COMPILER INVOKED BY:  PLM86 :F1:MAIN.SRC LARGE OPTIMIZE(3) TITLE('      MAIN$MODULE') &
                      PAGEWIDTH(99) DATE(01 MAY 81)


  1         MAIN$MODULE:DO;

                  /*_____________________________________________
                  |                                             |
                  |        ROBOT;iSBC 86/12A Monitor V1.0       |
                  |        Dec. 1980                            |
                  |        Programmer R.Rafauli                 |
                  |                                             |
                  |                  M A I N                    |
                  |                                             |
                  |          This main module  initializ the    |
                  |         hardware; and contains the command  |
                  |        dispatcher.                          |
                  |                                             |
                  |          Monitor commands:                  |
                  |             S - Substitute memory           |
                  |             E - Examine registers           |
                  |             D - Display memory              |
                  |             M - Move memory                 |
                  |             G - Go to a given address       |
                  |                 and start execution         |
                  |                                             |
                  |           Robot commands:                   |
                  |             C - Close the positional        |
                  |                 loop; no motion             |
                  |             X - Test actual position        |
                  |             F - TEACH Forward test          |
                  |             W - Forward repeatability       |
                  |                 test                        |
                  |             R - TEACH Reverse test          |
                  |             V - Reverse repeatability       |
                  |                 test                        |
                  |                                             |
                  |_____________________________________________*/
```

```
$EJECT
        /*_____________________________________________
        |                                             |
        |        L O C A L   D E C L A R A T I O N S  |
        |_____________________________________________*/

2  1        DECLARE
                CHAR        BYTE EXTERNAL,
                I           BYTE,
                TRUE        LITERALLY'0FFH',
                FALSE       LITERALLY'000H',
                USER$INT$SP LITERALLY'100H',
                STATP       LITERALLY'100H';
3  1        DECLARE
                MAC$MESS1(*)    BYTE DATA(0DH,0AH,'SBC86/12, ROBOT.',0),
                MAC$MESS2(*)    BYTE DATA(0DH,0AH,07H,'    Error',0),
                MAC$CMND(*) BYTE DATA('SEDMGCXMFVR');
4  1        DECLARE
                REGSAV(14)  WORD    PUBLIC,
                SP      LITERALLY'REGSAV(4)',
                BP      LITERALLY'REGSAV(5)',
                CS      LITERALLY'REGSAV(8)',
                DS      LITERALLY'REGSAV(9)',
                SS      LITERALLY'REGSAV(10)',
                ES      LITERALLY'REGSAV(11)',
                IP      LITERALLY'REGSAV(12)',
                FL      LITERALLY'REGSAV(13)';
5  1        DECLARE ERROR LABEL PUBLIC;
6  1        DECLARE HERE  LABEL PUBLIC;
```

```
$EJECT
        /*_____________________________________________
        |                                             |
        |        EXTERNAL PROCEDURE DECLARATIONS      |
        |_____________________________________________*/

 7  1    MAC$USART$INIT: PROCEDURE   EXTERNAL;
 8  2    END MAC$USART$INIT;

 9  1    MAC$PIT$INIT:   PROCEDURE   EXTERNAL;
10  2    END MAC$PIT$INIT;

11  1    MAC$PPI$INIT:   PROCEDURE   EXTERNAL;
12  2    END MAC$PPI$INIT;

13  1    MAC$PIC$INIT:   PROCEDURE   EXTERNAL;
14  2    END MAC$PIC$INIT;

15  1    MAC$STRING$OUT: PROCEDURE (PTR) EXTERNAL;
16  2       DECLARE PTR POINTER;
17  2    END MAC$STRING$OUT;

18  1    MAC$CRLF:   PROCEDURE   EXTERNAL;
19  2    END MAC$CRLF;

20  1    MAC$CHAR$OUT:   PROCEDURE (C)   EXTERNAL;
21  2       DECLARE C BYTE;
22  2    END MAC$CHAR$OUT;

23  1    MAC$CHAR$IN:    PROCEDURE   EXTERNAL;
24  2    END MAC$CHAR$IN;

25  1    MAC$EXAM$MEMO:  PROCEDURE   EXTERNAL;
26  2    END MAC$EXAM$MEMO;

27  1    MAC$EXAM$REG:   PROCEDURE   EXTERNAL;
28  2    END MAC$EXAM$REG;

29  1    MAC$DUMP:   PROCEDURE   EXTERNAL;
30  2    END MAC$DUMP;

31  1    MAC$MOVE:   PROCEDURE   EXTERNAL;
32  2    END MAC$MOVE;

33  1    MAC$GO:     PROCEDURE   EXTERNAL;
34  2    END MAC$GO;
```

```
            $EJECT
 35   1     RESET$8748: PROCEDURE   EXTERNAL;
 36   2     END RESET$8748;

 37   1     READ$FROM$BOARD:PROCEDURE   EXTERNAL;
 38   2     END READ$FROM$BOARD;

 39   1     FORWARD$TEST$RUN:PROCEDURE  EXTERNAL;
 40   2     END FORWARD$TEST$RUN;

 41   1     FORWARD$TEST$LOAD:PROCEDURE EXTERNAL;
 42   2     END FORWARD$TEST$LOAD;

 43   1     REVERSE$TEST$RUN:PROCEDURE  EXTERNAL;
 44   2     END REVERSE$TEST$RUN;

 45   1     REVERSE$TEST$LOAD:PROCEDURE EXTERNAL;
 46   2     END REVERSE$TEST$LOAD;
```

```
            $EJECT

 47   1     HERE:
            DISABLE;
 48   1        CALL    MAC$USART$INIT;
 49   1        CALL    MAC$PIT$INIT;
 50   1        CALL    MAC$PPI$INIT;
 51   1        CALL    MAC$PIC$INIT;
 52   1        CALL TIME(255);
 53   1        CALL TIME(255);
 54   1        CALL MAC$STRING$OUT(@MAC$MESS1);
 55   1        CS,SS,DS,ES,FL,IP=0;
 56   1        SP=USER$INT$SP;
 57   1     NEXT$CMND:
            CALL MAC$CRLF;
 58   1        CALL MAC$CHAR$OUT(0);
 59   1        CALL MAC$CHAR$OUT('.');
 60   1        CALL MAC$CHAR$IN;
 61   1        DO I=0 TO LAST(MAC$CMND);
 62   2           IF CHAR=MAC$CMND(I) THEN GOTO DISTRIBUTE;
 64   2        END;
 65   1        GOTO ERROR;
 66   1     DISTRIBUTE:        /* COMMAND DISPATCHER*/
            DO CASE I;
 67   2           CALL MAC$EXAM$MEMO;
 68   2           CALL MAC$EXAM$REG;
 69   2           CALL MAC$DUMP;
 70   2           CALL MAC$MOVE;
 71   2           CALL MAC$GO;
 72   2           CALL RESET$8748;
 73   2           CALL READ$FROM$BOARD;
 74   2           CALL FORWARD$TEST$RUN;
 75   2           CALL FORWARD$TEST$LOAD;
 76   2           CALL REVERSE$TEST$RUN;
 77   2           CALL REVERSE$TEST$LOAD;
 78   2        END;
 79   1        GOTO NEXT$CMND;
 80   1     ERROR:
            CALL MAC$STRING$OUT(@MAC$MESS2);
 81   1        GOTO NEXT$CMND;
 82   1     END MAIN$MODULE;
```

```
PL/M-86 COMPILER        ROBOT$COMMAND$MODULE                    01 MAY 81  PAGE   1

ISIS-II PL/M-86 V2.1 COMPILATION OF MODULE ROBOTCOMMANDMODULE
OBJECT MODULE PLACED IN :F1:ROBOT.OBJ
COMPILER INVOKED BY:  PLM86 :F1:ROBOT.SRC LARGE OPTIMIZE(3) TITLE('      ROBOT$COMMAND$MODULE') &
                      PAGEWIDTH(99) DATE(01 MAY 81)


  1          ROBOT$COMMAND$MODULE:DO;

                /*____________________________________________________
                :                                                    :
                :        ROBOT,iSBC 86/12A MONITOR V1.0              :
                :        DEC. 1980                                   :
                :        PROGRAMMER R.RAFAULI                        :
                :                                                    :
                :          R O B O T   C O M M A N D                 :
                :                                                    :
                :          THIS MODULE TESTS THE CONTROL             :
                :        SYSTEM USED WITH THE UNIMATE 2000           /
                :        MARK II ROBOT.                              :
                :                                                    :
                :          ROBOT COMMANDS:                           :
                :            C - CLOSE THE POSITIONAL                :
                :                LOOP, NO MOTION                     :
                :            X - TEST ACTUAL POSITION                :
                :            F - TEACH FORWARD TEST                  :
                :            W - FORWARD REPEATABILITY               :
                :                TEST                                :
                :            R - TEACH REVERSE TEST                  :
                :            V - REVERSE REPEATABILITY               :
                :                TEST                                :
                :                                                    :
                :____________________________________________________*/
```

```
$EJECT
        /*_____________________________________________
        |                                             |
        |        L O C A L   D E C L A R A T I O N S  |
        |_____________________________________________*/

 2  1      DECLARE
              I BYTE;
 3  1      DECLARE
              BOARD$BUFFER$0   WORD AT(700H),
              BOARD$BUFFER$1   WORD AT(702H),
              FORWARD$BUFFER$0 WORD AT(704H),
              FORWARD$BUFFER$1 WORD AT(706H),
              REVERSE$BUFFER$0 WORD AT(708H),
              REVERSE$BUFFER$1 WORD AT(70AH),
              BOARD$ADDRESS  WORD AT(10000H);
 4  1      DECLARE
              PORT$C$ADDRESS  LITERALLY'0CCH';
 5  1      DECLARE
              PIC$INIT    LITERALLY'0C0H',
              PIC$OCW3    LITERALLY'00AH';
 6  1      DECLARE
              MESS1(*) BYTE DATA     (0DH,0AH,'Update =',0),
              MESS2(*) BYTE DATA (0DH,0AH,'Forward Test',0),
              MESS3(*) BYTE DATA(0DH,0AH,'Input Word 1:',0),
              MESS4(*) BYTE DATA(0DH,0AH,'Input Word 2:',0),
              MESS5(*) BYTE DATA (0DH,0AH,'Reverse Test',0);
```

```
$EJECT
        /*_____________________________________________
        |                                             |
        |        EXTERNAL PROCEDURE DECLARATIONS      |
        |_____________________________________________*/

 7  1    MAC$STRING$OUT: PROCEDURE (PTR) EXTERNAL;
 8  2       DECLARE PTR POINTER;
 9  2    END MAC$STRING$OUT;

10  1    MAC$DLIX$SCAN:  PROCEDURE   EXTERNAL;
11  2    END MAC$DLIX$SCAN;

12  1    MAC$WORD$IN:    PROCEDURE WORD EXTERNAL;
13  2    END MAC$WORD$IN;

14  1    MAC$WORD$OUT:   PROCEDURE (W) EXTERNAL;
15  2       DECLARE W WORD;
16  2    END MAC$WORD$OUT;
```

```
          $EJECT
                  /*_____________________________________________
                  |                                             |
                  |          COMMAND PROCEDURES SECTION         |
                  |_____________________________________________*/


          /* THIS PROCEDURE IS USED TO INITIALIZE THE SERVO CARDS AND
             PUT ALL SERVO MOTORS IN CLOSED POSITIONAL LOOP ********/

17   1    RESET$8748:
             PROCEDURE   PUBLIC;
18   2       OUTPUT(PORT$C$ADDRESS)=INPUT(PORT$C$ADDRESS) AND    0FDH;
19   2       OUTPUT(PORT$C$ADDRESS)=INPUT(PORT$C$ADDRESS) OR 02H;
20   2       DO I=1 TO 4;
21   3           CALL TIME(250);
22   3       END;
23   2    END RESET$8748;


          /* THIS PROCEDURE IS USED TO READ ONE WORD FROM THE SERVO
             CARD. IT IS USED IN TEACH MODE AND WHEN UPDATING THE
             ACTUAL POSITION ON THE CRT ****************************/

24   1    READ$FROM$BOARD:
          PROCEDURE   PUBLIC;
25   2       CALL MAC$STRING$OUT(CMESS1);
26   2       BOARD$BUFFER$0=NOT(BOARD$ADDRESS);
27   2       CALL MAC$WORD$OUT(BOARD$BUFFER$0);
28   2    END READ$FROM$BOARD;
```

```
                $EJECT

                /* THIS PROCEDURE IS USED TO WRITE TWO WORDS TO THE SERVO CARD */

 29   1         WRITE$TO$BOARD:
                    PROCEDURE;
 30   2             OUTPUT(PIC$INIT)=PIC$OCW3;
 31   2             DO WHILE(INPUT(PIC$INIT))<> 1H;
 32   3             END;
 33   2             BOARD$ADDRESS=NOT(BOARD$BUFFER$0);
 34   2             OUTPUT(PORT$C$ADDRESS)=INPUT(PORT$C$ADDRESS) AND 0FEH;
 35   2             OUTPUT(PORT$C$ADDRESS)=INPUT(PORT$C$ADDRESS) OR 01H;
 36   2             OUTPUT(PIC$INIT)=PIC$OCW3;
 37   2             DO WHILE(INPUT(PIC$INIT))<> 1H;
 38   3             END;
 39   2             BOARD$ADDRESS=NOT(BOARD$BUFFER$1);
 40   2             OUTPUT(PORT$C$ADDRESS)=INPUT(PORT$C$ADDRESS) AND 0FEH;
 41   2             OUTPUT(PORT$C$ADDRESS)=INPUT(PORT$C$ADDRESS) OR 01H;
 42   2             OUTPUT(PIC$INIT)=PIC$OCW3;
 43   2             DO WHILE(INPUT(PIC$INIT))<> 01H;
 44   3             END;
 45   2         END WRITE$TO$BOARD;
```

```
            $EJECT

            /* THIS PROCEDURE IS USED TO TEST THE REPEATABILITY OF THE
               MOTION COMMAND ****************************************/

46    1     FORWARD$TEST$RUN:
                PROCEDURE   PUBLIC;
47    2         CALL MAC$STRING$OUT(@MESS2);
48    2         BOARD$BUFFER$0=FORWARD$BUFFER$0;
49    2         BOARD$BUFFER$1=FORWARD$BUFFER$1;
50    2         CALL WRITE$TO$BOARD;
51    2     END FORWARD$TEST$RUN;


            /* THIS PROCEDURE IS USED TO TEACH THE ROBOT A GIVEN DISTANCE
               AND MOVE IT BY THIS DISTANCE ****************************/

52    1     FORWARD$TEST$LOAD:
                PROCEDURE   PUBLIC;
53    2         CALL MAC$STRING$OUT(@MESS3);
54    2         CALL MAC$BLOCK$SCAN;
55    2         FORWARD$BUFFER$0=MAC$WORD$IN;
56    2         CALL MAC$STRING$OUT(@MESS4);
57    2         CALL MAC$BLOCK$SCAN;
58    2         FORWARD$BUFFER$1=MAC$WORD$IN;
59    2         CALL FORWARD$TEST$RUN;
60    2     END FORWARD$TEST$LOAD;
```

```
                $EJECT

                /* THIS PROCEDURE IS USED TO TEST THE REPEATABLITY IN THE
                   REVERSE DIRECTION ***********************************/

 61   1         REVERSE$TEST$RUN:
                    PROCEDURE   PUBLIC;
 62   2             CALL MAC$STRING$OUT(@MESS5);
 63   2             BOARD$BUFFER$0=REVERSE$BUFFER$0;
 64   2             BOARD$BUFFER$1=REVERSE$BUFFER$1;
 65   2             CALL WRITE$TO$BOARD;
 66   2         END REVERSE$TEST$RUN;


                /* THIS PROCEDURE IS USED TO TEACH THE ROBOT A REVERSE MOTION */

 67   1         REVERSE$TEST$LOAD:
                    PROCEDURE   PUBLIC;
 68   2             CALL MAC$STRING$OUT(@MESS3);
 69   2             CALL MAC$BLINK$SCAN;
 70   2             REVERSE$BUFFER$0=MAC$WORD$IN;
 71   2             CALL MAC$STRING$OUT(@MESS4);
 72   2             CALL MAC$BLINK$SCAN;
 73   2             REVERSE$BUFFER$1=MAC$WORD$IN;
 74   2             CALL REVERSE$TEST$RUN;
 75   2         END REVERSE$TEST$LOAD;
 76   1         END ROBOT$COMMAND$MODULE;
```

```
PL/M-86 COMPILER          INPUT$OUTPUT$MODULE                          01 MAY 81  PAGE   1

ISIS-II PL/M-86 V2.1 COMPILATION OF MODULE INPUTOUTPUTMODULE
OBJECT MODULE PLACED IN :F1:INOUT.OBJ
COMPILER INVOKED BY:  PLM86 :F1:INOUT.SRC LARGE OPTIMIZE(3) TITLE('      INPUT$OUTPUT$MODULE') &
                      PAGEWIDTH(99) DATE(01 MAY 81)


  1          INPUT$OUTPUT$MODULE:DO;

                    /*_____________________________________________
                    |                                             |
                    |        RODOT,iSBC 86/12A Monitor V1.0       |
                    |        Dec. 1980                            |
                    |        Programmer R.Rafauli                 |
                    |                                             |
                    |           I N P U T / O U T P U T           |
                    |                                             |
                    |         This module supports the INPUT/     |
                    |        OUTPUT via the keyboard and CRT.     |
                    |        terminal.                            |
                    |                                             |
                    |        Contained are the following procedures:|
                    |                                             |
                    |        MAC$USART$INITIALIZATION             |
                    |        MAC$PIT$INITIALIZTION                |
                    |        MAC$PPI$INITIALIZTION                |
                    |        MAC$PIC$INITIALIZATION               |
                    |        MAC$CARACTER$READY                   |
                    |        MAC$CONTROL$CHARACTER                |
                    |        MAC$CHARACTER$OUT                    |
                    |        MAC$CHARACTER$IN                     |
                    |        MAC$BYTE$OUT                         |
                    |        MAC$WORD$OUT                         |
                    |        MAC$STRING$OUT                       |
                    |        MAC$CRLF                             |
                    |        MAC$ADDRESS$IN                       |
                    |                                             |
                    |_____________________________________________*/
```

```
$EJECT
        /*_____________________________________________
        :                                             :
        :        L O C A L   D E C L A R A T I O N S  :
        :_____________________________________________*/

2  1        DECLARE
               CHAR        BYTE PUBLIC,
               I           BYTE,
               TRUE        LITERALLY'0FFH',
               FALSE       LITERALLY'000H',
               ASCII(*)    BYTE DATA('0123456789ABCDEF');
3  1        DECLARE
               PIT$MODE        LITERALLY'09EH',
               PIT$CONTROL LITERALLY'0D6H',
               PIT$COUNT2  LITERALLY'0D4H',
               PIT$COUNTVALUE  LITERALLY'008H';
4  1        DECLARE
               USART$STAT  LITERALLY'0DAH',
               USART$DATA  LITERALLY'0D8H',
               USART$RSET  LITERALLY'040H',
               USART$MODE  LITERALLY'04EH',
               USART$CMND  LITERALLY'037H',
               USART$RX$RDY    LITERALLY'002H',
               USART$TX$RDY    LITERALLY'001H';
5  1        DECLARE
               PPI$CONTROL LITERALLY'0CEH',
               PPI$RESET   LITERALLY'003H',
               PORS$C&ADDRESS   LITERALLY'0CCH',
               PPI$INIT$WORD   LITERALLY'092H';
6  1        DECLARE
               PIC$INIT    LITERALLY'0C0H',
               PIC$INIT1   LITERALLY'0C2H',
               PIC$ICW1    LITERALLY'013H',
               PIC$ICW2    LITERALLY'000H',
               PIC$ICW4    LITERALLY'00FH',
               PIC$OCW3    LITERALLY'00AH';
7  1        DECLARE
               ASCR        LITERALLY'00DH',
               ASLF        LITERALLY'00AH',
               ASBL        LITERALLY'020H';
```

```
          $EJECT

  8  1        DECLARE
                 REGINDX     WORD    EXTERNAL,
                 REGSAV(14)  WORD    EXTERNAL,
                 REGORD(*)   BYTE DATA(7,6,1,3,2,0,9,11,12,8,13),
                 SP      LITERALLY'REGSAV(4)',
                 BP      LITERALLY'REGSAV(5)',
                 CS      LITERALLY'REGSAV(8)',
                 DS      LITERALLY'REGSAV(9)',
                 SS      LITERALLY'REGSAV(10)',
                 ES      LITERALLY'REGSAV(11)',
                 IP      LITERALLY'REGSAV(12)',
                 FL      LITERALLY'REGSAV(13)';
```

```
                $EJECT
                        /*_____________________________________________
                        |                                             |
                        |        EXTERNAL PROCEDURE DECLARATIONS      |
                        |_____________________________________________*/

 9   1      DECLARE ERROR   LABEL   EXTERNAL;

10   1      MAC$VALD$HEX:   PROCEDURE (H) BYTE EXTERNAL;
11   2          DECLARE H BYTE;
12   2      END MAC$VALD$HEX;

13   1      MAC$HEX:PROCEDURE (C) WORD EXTERNAL;
14   2          DECLARE C BYTE;
15   2      END MAC$HEX;

16   1      MAC$VALD$REG$FIRST:PROCEDURE BYTE EXTERNAL;
17   2      END MAC$VALD$REG$FIRST;

18   1      MAC$VALD$REG:   PROCEDURE (C1,C2) BYTE EXTERNAL;
19   2          DECLARE (C1,C2) BYTE;
20   2      END MAC$VALD$REG;
```

```
          $EJECT
                /*____________________________________________
                |                                            |
                |          I/O PROCEDURES SECTION            |
                |                                            |
                |____________________________________________*/


          /* THIS PROCEDURE INITIALIZES THE 8251A PROGRAMMABLE
             COMMUNICATION INTERFACE ******************************
             ************************************************/

 21   1    MAC$USART$INIT:
              PROCEDURE   PUBLIC;
 22   2        OUTPUT(USART$STAT)=0H;
 23   2        CHAR=0;
 24   2        OUTPUT(USART$STAT)=0H;
 25   2        CHAR=0;
 26   2        OUTPUT(USART$STAT)=0H;
 27   2        CHAR=0;
 28   2        OUTPUT(USART$STAT)=USART$RESET;
 29   2        OUTPUT(USART$STAT)=USART$MODE;
 30   2        OUTPUT(USART$STAT)=USART$CMDD;
 31   2    END MAC$USART$INIT;

          /* THIS PROCEDURE INITIALIZES THE 8253 PROGRAMMABLE
             INTERVAL TIMER ***************************************
             ************************************************/

 32   1    MAC$PIT$INIT:
              PROCEDURE   PUBLIC;
 33   2        OUTPUT(PIT$CONTROL)=PIT$MODE;
 34   2        OUTPUT(PIT$COUNT2)=PIT$COUNTVALUE;
 35   2    END MAC$PIT$INIT;

          /* THIS PROCEDURE INITIALIZES THE 8255A PROGRAMMABLE
             PERIPHERAL INTERFACE *********************************
             ************************************************/

 36   1    MAC$PPI$INIT:
              PROCEDURE   PUBLIC;
 37   2        OUTPUT(PPI$CONTROL)=PPI$INIT$WORD;
 38   2        OUTPUT(PDR$C$ADDRESS)=PPI$RESET;
 39   2    END MAC$PPI$INIT;
```

```
            $EJECT

            /* THIS PROCEDURE INITIALIZES THE 8259A PROGRAMMABLE
               INTERRUPT CONTROLLERT ******************************
               ******************************************************/

 40   1     MAC$PIC$INIT:
                PROCEDURE   PUBLIC;
 41   2          OUTPUT(PIC$INIT)=PIC$ICW1;
 42   2          OUTPUT(PIC$INIT1)=PIC$ICW2;
 43   2          OUTPUT(PIC$INIT1)=PIC$ICW4;
 44   2     END MAC$PIC$INIT;

            /* THIS PROCEDURE CHECKS IF A CHARACTER FROM THE KEY
               BOARD IS READY TO BE READ ***************************
               ******************************************************/

 45   1     MAC$CHAR$RDY:   /* IS CHAR. RADY */
            PROCEDURE BYTE  PUBLIC;
 46   2        IF (INPUT(USART$STAT) AND USART$RX$RDY)=0 THEN RETURN FALSE;
 48   2        RETURN TRUE;
 49   2     END MAC$CHAR$RDY;

            /* THIS PROCEDURE CHECKS FOR CONTROL CHARACTERS, CNTRL S
               STOPS OUTPUT TO THE CRT., CNTRL Q CONTINUES OUTPUT, CNTRL C
               RETURNS CONTROL TO COMMAND MODE *************************
               ******************************************************/

 50   1     MAC$CONTROL$CHAR:
            PROCEDURE   PUBLIC;
 51   2        CHAR=INPUT(USART$DATA) AND 07FH;
 52   2        IF CHAR=13H THEN
 53   2        DO WHILE CHAR()11H;
 54   3           IF MAC$CHAR$RDY THEN
 55   3           DO;
 56   4              CHAR=INPUT(USART$DATA) AND 07FH;
 57   4              IF CHAR=003H THEN GOTO ERROR;
 59   4           END;
 60   3        END;
 61   2        ELSE IF CHAR=003H THEN GOTO ERROR;
            END MAC$CONTROL$CHAR;
```

```
                $EJECT

                /* THIS PROCEDURE OUTPUTS ONE CHARACTER TO THE CRT.  **
                   ****************************************************/

 64   1         MAC$CHAR$OUT:
                PROCEDURE(C)    PUBLIC;
 65   2            DECLARE C   BYTE;
 66   2            IF MAC$CHAR$RDY THEN CALL MAC$CONTROL$CHAR;
 68   2            DO WHILE (INPUT(USART$STAT) AND USART$TX$RDY)=0;
 69   3            END;
 70   2            OUTPUT(USART$DATA)=C;
 71   2         END MAC$CHAR$OUT;

                /* THIS PROCEDURE READS ONE CHARACTER FROM KEYBD.**
                   ****************************************************/

 72   1         MAC$CHAR$IN:
                PROCEDURE   PUBLIC;
 73   2            DO WHILE (INPUT(USART$STAT) AND USART$RX$RDY)=0;
 74   3            END;
 75   2            CHAR=INPUT(USART$DATA) AND 07FH;
 76   2            IF CHAR)=ASBL THEN CALL MAC$CHAR$OUT(CHAR);
 78   2         END MAC$CHAR$IN;

                /* THIS PROCEDURE OUTPUTS ONE BYTE TO CRT.  *******
                   ****************************************************/

 79   1         MAC$BYTE$OUT:
                PROCEDURE(B)    PUBLIC;
 80   2            DECLARE B   BYTE;
 81   2            CALL MAC$CHAR$OUT(ASCII(SHR(B,4) AND 00FH));
 82   2            CALL MAC$CHAR$OUT(ASCII(B AND 00FH));
 83   2         END MAC$BYTE$OUT;

                /* THIS PROCEDURE OUTPUTS ONE WORD TO CRT.   ******
                   ****************************************************/

 84   1         MAC$WORD$OUT:
                PROCEDURE(W)    PUBLIC;
 85   2            DECLARE W   WORD;
 86   2            CALL MAC$BYTE$OUT(HIGH(W));
 87   2            CALL MAC$BYTE$OUT(LOW(W));
 88   2         END MAC$WORD$OUT;
```

```
            $EJECT

            /* THIS PROCEDURE OUTPUTS A STRING OF CHARACTERS TO
               CRT. ***************************************************
               ***************************************************/

 89   1     MAC$STRING$OUT:
            PROCEDURE(PTR)  PUBLIC;
 90   2         DECLARE PTR        POINTER,
                   STR BASED PTR(1)    BYTE;
 91   2         I=0;
 92   2         DO WHILE STR(I)<>0;
 93   3             CALL MAC$CHAR$OUT(STR(I));
 94   3             I=I+1;
 95   3         END;
 96   2     END MAC$STRING$OUT;

            /* THIS PROCEDURE OUTPUTS A CARRIAGE RETURN AND LINE
               FEED TO THE CRT. ****************************************
               ***************************************************/

 97   1     MAC$CRLF:
            PROCEDURE   PUBLIC;
 98   2         CALL MAC$CHAR$OUT(ASCR);
 99   2         CALL MAC$CHAR$OUT(ASLF);
100   2     END MAC$CRLF;

            /* THIS PROCEDURE  READS A WORD FROM THE CRT.  *****
               ***************************************************/

101   1     MAC$WORD$IN:        /* GET WORD */
            PROCEDURE WORD  PUBLIC;
102   2         DECLARE (SAVE,W)    WORD,
                   (T)     BYTE;
103   2         W=0;
104   2         DO WHILE TRUE;
105   3             T=CHAR;
106   3             SAVE=0;
107   3             IF MAC$VALD$REG$FIRST THEN
108   3             DO;
109   4                 CALL MAC$CHAR$IN;
110   4                 IF MAC$VALDREG(T,CHAR) THEN
111   4                 DO;
112   5                     SAVE=REGSAV(REGINDX);
113   5                     CALL MAC$CHAR$IN;
114   5                     GOTO CONT;
```

```
                $EJECT

115   5                 END;
                        ELSE
116   4                     SAVE=MAC$HEX(T);
117   4                 END;
118   3             IF NOT(MAC$VALDHEX(T)) THEN GOTO ERROR;
120   3             DO WHILE MAC$VALDHEX(CHAR);
121   4                 SAVE=SHL(SAVE,4)+MAC$HEX(CHAR);
122   4                 CALL MAC$CHAR$IN;
123   4             END;
124   3       CONT:     W=SAVE;
125   3             IF CHAR=ASCR OR CHAR=':' OR CHAR=ASBL THEN
126   3             RETURN W;
                    ELSE
127   3             GOTO ERROR;
128   3         END;
129   2       END MAC$WORD$IN;

              /* THIS PROCEDURE  READS AN ADDRESS FROM THE CRT  ******
                 ***************************************************/

130   1       MAC$ADDRES$IN:
              PROCEDURE(PTR,DEFULTBASE)   PUBLIC;
131   2           DECLARE PTR     POINTER,
                      DEFULTBASE  WORD,
                      ARG BASED PTR STRUCTURE (OFF WORD,SEG WORD);
132   2           ARG.SEG=DEFULTBASE;
133   2           ARG.OFF=MAC$WORD$IN;
134   2           IF CHAR=':' THEN
135   2           DO;
136   3               CALL MAC$CHAR$IN;
137   3               ARG.SEG=ARG.OFF;
138   3               ARG.OFF=MAC$WORD$IN;
139   3               IF CHAR=':' THEN GOTO ERROR;
141   3           END;
142   2       END MAC$ADDRES$IN;
143   1       END INPUT$OUTPUT$MODULE;
```

```
PL/M-86 COMPILER        UTILITY$MODULE                                  01 MAY 81  PAGE   1

ISIS-II PL/M-86 V2.1 COMPILATION OF MODULE UTILITYMODULE
OBJECT MODULE PLACED IN :F1:UTILTY.OBJ
COMPILER INVOKED BY:  PLM86 :F1:UTILTY.SRC LARGE OPTIMIZE(3) TITLE('      UTILITY$MODULE') &
                      PAGEWIDTH(99) DATE(01 MAY 81)


  1          UTILITY$MODULE:DO;

                    /*_____________________________________________
                    |                                             |
                    |        ROBOT,iSBC 86/12A Monitor V1.0       |
                    |        Dec. 1980                            |
                    |        Programmer R.Rafauli                 |
                    |                                             |
                    |              U T I L I T Y                  |
                    |                                             |
                    |          This module is used by all the mig-|
                    |        her level procedures in the program. |
                    |                                             |
                    |          It includes the following functions:|
                    |        MAC$VALID$HEX                        |
                    |        MAC$HEX                              |
                    |        MAC$VALID$REG$FIRST                  |
                    |        MAC$VALID$REG                        |
                    |        MAC$TEST$WORD$MODE                   |
                    |        MAC$BLANK$SCAN                       |
                    |_____________________________________________*/
```

```
$EJECT

                /*_____________________________________________
                ¦                                             ¦
                ¦        L O C A L   D E C L A R A T I O N S  ¦
                ¦_____________________________________________*/

2  1        DECLARE
               CHAR        BYTE EXTERNAL,
               I           BYTE,
               WORD$MODE   BYTE PUBLIC,
               TRUE        LITERALLY'0FFH',
               FALSE       LITERALLY'000H',
               ASCII(*)    BYTE DATA('01234567B9ABCDEF'),
               ASBL        LITERALLY'020H';
3  1        DECLARE
               REG(*)      BYTE DATA('AXBXCXDXSPBPSIDICSDSSSESIPFL'),
               REGINDX     WORD    EXTERNAL,
               REGSAV(14)  WORD    EXTERNAL,
               REGORD(*)   BYTE DATA(7,6,1,3,2,0,9,11,12,8,13),
               SP      LITERALLY'REGSAV(4)',
               BP      LITERALLY'REGSAV(5)',
               CS      LITERALLY'REGSAV(B)',
               DS      LITERALLY'REGSAV(9)',
               SS      LITERALLY'REGSAV(10)',
               ES      LITERALLY'REGSAV(11)',
               IP      LITERALLY'REGSAV(12)',
               FL      LITERALLY'REGSAV(13)';


                /*_____________________________________________
                ¦                                             ¦
                ¦     EXTERNAL PROCEDURE DECLARATIONS         ¦
                ¦_____________________________________________*/

4  1     MAC$CHAR$IN:    PROCEDURE   EXTERNAL;
5  2     END MAC$CHAR$IN;
```

```
                $EJECT

                /* TO CHECK IF THE CHARACTER IS A VALID HEXADECIMAL DIGIT */

  6   1         MAC$VALD$HEX:   PROCEDURE (H)   BYTE    PUBLIC;
  7   2             DECLARE H   BYTE;
  8   2             DO I=0 TO LAST(ASCII);
  9   3                 IF H=ASCII(I) THEN RETURN TRUE;
 11   3             END;
 12   2             RETURN FALSE;
 13   2         END MAC$VALD$HEX;

                /*CONVERT FROM ASCII TO HEX */

 14   1         MAC$HEX:    PROCEDURE(C)    WORD    PUBLIC;
 15   2             DECLARE C   BYTE;
 16   2             IF C<='9' THEN RETURN DOUBLE(C-30H);
 18   2             ELSE RETURN DOUBLE(C-37H);
 19   2         END MAC$HEX;

                /* CHECK IF THE CHARACTER IS A VALID FIRST LETTER OF A REGESTER NAME*/

 20   1         MAC$VALD$REG$FIRST: PROCEDURE BYTE  PUBLIC;
 21   2             DO I=0 TO 26 BY 2;
 22   3                 IF CHAR=REG(I) THEN RETURN TRUE;
 24   3             END;
 25   2             RETURN FALSE;
 26   2         END MAC$VALD$REG$FIRST;

                /* CHECK IF THE TWO CHARACTERS ARE A VALID REGESTER NAME */

 27   1         MAC$VALD$REG:   PROCEDURE(C1,C2)    BYTE    PUBLIC;
 28   2             DECLARE (C1,C2) BYTE;
 29   2             DO REGINDX=0 TO 13;
 30   3                 IF C1=REG(REGINDX*2) AND C2=REG(REGINDX*2+1) THEN
 31   3                 RETURN TRUE;
 32   3             END;
 33   2             RETURN FALSE;
 34   2         END MAC$VALD$REG;

                /*  CHECK IF THE COMMAND IS FOLLOWED BY 'W' */

 35   1         MAC$TST$WORD$MOD:   PROCEDURE   PUBLIC;
 36   2             WORD$MODE=FALSE;
 37   2             CALL MAC$CHAR$IN;
 38   2             IF CHAR='W' THEN
```

```
                $EJECT
 39   2             DO;
 40   3                 WORD$MODE=TRUE;
 41   3                 CALL MAC$CHAR$IN;
 42   3             END;
 43   2             IF CHAR=ASBL THEN
 44   2             CALL MAC$CHAR$IN;
 45   2         END MAC$TST$WORD$MOD;

                /* CHECK FOR OPTIONAL BLANK IN THE COMMAND */

 46   1         MAC$BLNK$SCAN:  PROCEDURE   PUBLIC;
 47   2         LOOP:   CALL MAC$CHAR$IN;
 48   2             IF CHAR=ASBL THEN
 49   2             GOTO LOOP;
 50   2         END MAC$BLNKSCAN;
 51   1         END UTILITY$MODULE;
```

APPENDIX B

ASM48 PROGRAM LISTING

VARIABLE DESCRIPTIONS:

DT: The loop time (equal to 4 msec.)

DX: The increment in the command distance per DT.

X_{com}: The command distance per DT.

IFL: Flag, set when motion is completed.

DXMAS: The maximum increment in the command distance.

XEND: The total command distance.

FLG: Flag, set in the RUN part of the trapezoidal profiles.

X_1: The distance travelled during acceleration time.

X_2: The distance travelled until deceleration time.

V_{com}: The command velocity.

```
ASM48 :F1:CONTRO.A48 TITLE('      SERVO&CONTROLLER')


ISIS-II MCS-48/UPI-41 MACRO ASSEMBLER, V4.0            PAGE    1
     SERVO&CONTROLLER

 LOC  OBJ        LINE        SOURCE STATEMENT

                   1
                   2
                   3
                   4
                   5
                   6      ;_____________________________________;
                   7      ;                                     ;
                   8      ;        SERVO CARD SOFTWARE V1.0     ;
                   9      ;        JAN. 1981                    ;
                  10      ;        PROGRAMMER R.RAFAULI         ;
                  11      ;                                     ;
                  12      ;                                     ;
                  13      ;        S E R V O   C O N T R O L    ;
                  14      ;                                     ;
                  15      ;                                     ;
                  16      ;    THIS CONTROL PROGRAM IS RESIDENT ;
                  17      ;   IN THE PROGRAM MEMORY OF THE INTEL ;
                  18      ;   8748 MICROPROCESSOR. IT GENERATES ;
                  19      ;   THE TRAPEZOIDAL VELOCITY PROFILE. ;
                  20      ;   IT IS WRITTEN IN ASSAMBLY LANGUAGE, ;
                  21      ;   (ASM 48)                          ;
                  22      ;                                     ;
                  23      ;_____________________________________;
                  24 $EJECT
```

```
ISIS-II MCS-48/UPI-41 MACRO ASSEMBLER, V4.0            PAGE    2
     SERVO&CONTROLLER

 LOC  OBJ        LINE        SOURCE STATEMENT

                  25       ;_____________________________________;
                  26       ;                                     ;
                  27       ;  DEFINE NAMES TO THE WORKING REGISTERS ;
                  28       ;                                     ;
                  29       ;_____________________________________;
 0000             30          POINR1  EQU     R0
 0001             31          POINR2  EQU     R1
 0002             32          TMP     EQU     R2
 0003             33          DX      EQU     R3
 0004             34          X1L     EQU     R4
 0005             35          X1H     EQU     R5
 0006             36          X1S     EQU     R6
 0007             37          DXMAX   EQU     R7
 0020             38          XENDL   EQU     20H
 0021             39          XENDH   EQU     21H
 0022             40          XENDS   EQU     22H
 0023             41          X2L     EQU     23H
 0024             42          X2H     EQU     24H
 0025             43          X2S     EQU     25H
 0026             44          FLAG    EQU     26H
 0027             45          XCOML   EQU     27H
 0028             46          XCOMH   EQU     28H
 0029             47          XCOMS   EQU     29H
 002A             48          XACTL   EQU     2AH
 002B             49          XACTH   EQU     2BH
 002C             50          XACTS   EQU     2CH
 002D             51          VCOML   EQU     2DH
 002E             52          VCOMH   EQU     2EH
 002F             53          VCOMS   EQU     2FH
 0030             54          XSPL    EQU     30H
 0031             55          XSPH    EQU     31H
 0032             56          XSPS    EQU     32H
                  57 $EJECT
```

```
ISIS-II MCS-48/UPI-41 MACRO ASSEMBLER, V4.0            PAGE    3
     SERVO$CONTROLLER

 LOC  OBJ        LINE        SOURCE STATEMENT

                   58       ;_____________________________________;
                   59       ;                                     ;
                   60       ;          DEFINE THE CONTROL PORTS   ;
                   61       ;                                     ;
                   62       ;_____________________________________;
 0000              63          RSTALL  EQU     00
 0003              64          STP1    EQU     003H
 0064              65          STP2    EQU     064H
 00BF              66          CCACK   EQU     0BFH
 0040              67          ECACK   EQU     040H
 00FD              68          CRFBK   EQU     0FDH
 0042              69          ERFBK   EQU     042H
 00DF              70          DIHBL   EQU     0DFH
 0080              71          EIHBH   EQU     080H
 007F              72          DIHBH   EQU     07FH
 0004              73          EOHBL   EQU     004H
 00FB              74          DOHBL   EQU     0FBH
 0008              75          EOHBH   EQU     008H
 00F7              76          DOHBH   EQU     0F7H
 0020              77          ELSDAL  EQU     020H
 00DF              78          DLSDAL  EQU     0DFH
 0010              79          ELSDAH  EQU     010H
 00EF              80          DLSDAH  EQU     0EFH
 00DF              81          DONE    EQU     0DFH
 0020              82          NOPDONE EQU     020H
                   83 $EJECT
```

```
ISIS-II MCS-48/UPI-41 MACRO ASSEMBLER, V4.0            PAGE    4
     SERVO&CONTROLLER

 LOC  OBJ        LINE        SOURCE STATEMENT

 0000            84          ORG     0H;  ...... .. MEMORY PAGE 0 ........
                 85
 0000 040C       86          JMP     START
                 87        ;_____________________________________
                 88        ;                                     ;
                 89        ;    INDIRECT JUMPS FROM PAGE 0       ;
                 90        ;_____________________________________;
                 91
                 92
 0002 04CE       93 TGI3:   JMP     TG3
 0004 243C       94 TGI20:  JMP     TG20
 0006 04C5       95 ADIDX:  JMP     ADDX
 0008 2450       96 TGI40:  JMP     TG40
 000A 2456       97 TGI50:  JMP     TG50
 000C D5         98 START:  SEL     RB1;  ............Use registers 24 to 51 only
 000D 9A00       99         ANL     P2,#RSTALL;  .....Initialize the hardware
 000F 8A64      100         ORL     P2,#STP2
 0011 27        101         CLR     A
 0012 02        102         OUTL    BUS,A
 0013 8803      103         ORL     BUS,#STP1
 0015 27        104 NEXT:   CLR     A; ...............Clear the working registers
 0016 AB        105         MOV     R3,A
 0017 B807      106         MOV     POINR1,#7H
 0019 B92C      107         MOV     POINR2,#XACTS
 001B A1        108 CLEAR:  MOV     @POINR2,A
 001C C9        109         DEC     POINR2
 001D EB1B      110         DJNZ    POINR1,CLEAR
 001F 85        111         CLR     F0
 0020 5624      112         JT1     LOOP;  ...........Is "INB" full?
 0022 5455      113         CALL    READ;  ...........Yes, read it
 0024 4615      114 LOOP:   JNT1    NEXT;  ...........And start new motion
 0026 5471      115         CALL    WRITE; ......... .No, update actual position
 0028 2628      116 CLOC:   JNT0    CLOC;  ...........to CRT and wait for loop trigger
                117 $EJECT
```

```
ISIS-II MCS-48/UPI-41 MACRO ASSEMBLER, V4.0            PAGE    5
     SERVO&CONTROLLER

 LOC  OBJ        LINE        SOURCE STATEMENT

                  118
                  119      ;_________________________________;
                  120      ;                                 ;
                  121      ;  THE BEGINING OF THE TIMING LOOP ;
                  122      ;_________________________________;
                  123
                  124
 002A 9ABF        125          ANL     P2,#CCACK
 002C 8A40        126          ORL     P2,#ECACK
 002E 98FE        127          ANL     BUS,#0FEH
 0030 09          128          IN      A,P1
 0031 8B01        129          ORL     BUS,#01H
 0033 B82A        130          MOV     POINR1,#XACTL
 0035 F23B        131          JB7     DOWN;  ...........Is IT FORWARD MOTION?
 0037 B900        132 UP:      MOV     POINR2,#0H;  .....Yes, ADD FEEDBACK
 0039 043D        133          JMP     UPDN
 003B B9FF        134 DOWN:    MOV     POINR2,#0FFH;  ...No, SUBTRACT FEEDBACK
 003D 60          135 UPDN:    ADD     A,@POINR1
 003E A0          136          MOV     @POINR1,A
 003F 18          137          INC     POINR1
 0040 F0          138          MOV     A,@POINR1
 0041 79          139          ADDC    A,POINR2
 0042 A0          140          MOV     @POINR1,A
 0043 18          141          INC     POINR1
 0044 F0          142          MOV     A,@POINR1
 0045 79          143          ADDC    A,POINR2
 0046 A0          144          MOV     @POINR1,A
                  145
                  146 ; 1 IF(FLAG.EQ.1)GOTO 50  ........Is MOTION COMPLETED?
                  147
 0047 B826        148 CONO:    MOV     POINR1,#FLAG
 0049 F0          149          MOV     A,@POINR1
 004A 120A        150          JB0     TG150;  ..........Yes, KEEP THE POSITIONAL LOOP CLOSED
                  151
                  152 ; IF(DXMAX.EQ.0)GOTO 40  .........NO, IS COMMANDED VELOCITY EQUAL TO  ZERO?
                  153
 004C FF          154          MOV     A,DXMAX
 004D C608        155          JZ      TG140;  ..........Yes, CLOSED POSITIONAL LOOP
                  156 ;                                        AND STOP MOTION
                  157 ; IF(XEND.LT.0)GOTO 2  ...........No, EXECUTE THE TRAPEZOIDAL
                  158 ;                                       VELOCITY PROFILE
 004F A5          159          CLR     F1
 0050 B822        160          MOV     POINR1,#XENDS
 0052 F0          161          MOV     A,@POINR1
 0053 1261        162          JB0     TG2
                  163 $EJECT
```

```
ISIS-II MCS-48/UPI-41 MACRO ASSEMBLER, V4.0            PAGE    6
     SERVO&CONTROLLER

 LOC  OBJ        LINE        SOURCE STATEMENT

                 164 ; XCOM=XCOM+DX  ..................THE ACCELERATION PART.
                 165
 0055 B827       166         MOV     POINR1,#XCOML
 0057 FB         167         MOV     A,DX
 0058 60         168         ADD     A,@POINR1
 0059 A0         169         MOV     @POINR1,A
 005A 18         170         INC     POINR1
 005B F0         171         MOV     A,@POINR1
 005C 1300       172         ADDC    A,#0
 005E A0         173         MOV     @POINR1,A
                 174
                 175 ; GOTO 5
                 176
 005F 0475       177         JMP     TG5
                 178
                 179 ; 2 XCOM=XCOM-DX
                 180
 0061 B5         181 TG2:    CPL     F1;  .............SET F1 IF XEND IS -VE.
 0062 B827       182         MOV     POINR1,#XCOML
 0064 FB         183         MOV     A,DX
 0065 C675       184         JZ      TG5
 0067 37         185         CPL     A
 0068 17         186         INC     A
 0069 60         187         ADD     A,@POINR1
 006A A0         188         MOV     @POINR1,A
 006B 18         189         INC     POINR1
 006C F0         190         MOV     A,@POINR1
 006D 13FF       191         ADDC    A,#0FFH
 006F A0         192         MOV     @POINR1,A
 0070 18         193         INC     POINR1
 0071 F0         194         MOV     A,@POINR1
 0072 13FF       195         ADDC    A,#0FFH
 0074 A0         196         MOV     @POINR1,A
 0075 5400       197  TG5:   CALL    DAC;  ............OUTPUT TO SERVO VCOM=XCOM-XACT
                 198
                 199 ; IF(DX.NE.DXMAX)GOTO 20
                 200
 0077 FF         201         MOV     A,DXMAX
 0078 DB         202         XRL     A,DX
 0079 9604       203         JNZ     TG120
                 204
                 205 ; IF(FLAG0.EQ.1)GOTO 3
                 206
 007B B602       207         JF0     TG13
                 208 $EJECT
```

```
ISIS-II MCS-48/UPI-41 MACRO ASSEMBLER, V4.0            PAGE    7
     SERVO&CONTROLLER

 LOC  OBJ        LINE        SOURCE STATEMENT

                  209 ; SET FLAG0 ....................END OF THE ACCELERATION PART,
                  210 ;                                X1,X2 ARE KNOWN AT THIS POINT
                  211
 007D 95          212          CPL     F0
                  213
                  214 ; 25 X1=XCOM
                  215
 007E B827        216          MOV     POINR1,#XCOML
 0080 F0          217          MOV     A,@POINR1
 0081 AC          218          MOV     X1L,A
 0082 18          219          INC     POINR1
 0083 F0          220          MOV     A,@POINR1
 0084 AD          221          MOV     X1H,A
 0085 18          222          INC     POINR1
 0086 F0          223          MOV     A,@POINR1
 0087 AE          224          MOV     X1S,A
                  225
                  226 ; X2=XEND-X1
                  227
 0088 B823        228          MOV     POINR1,#X2L
 008A FC          229          MOV     A,X1L
 008B 37          230          CPL     A
 008C 0301        231          ADD     A,#001H
 008E A0          232          MOV     @POINR1,A
 008F 18          233          INC     POINR1
 0090 FD          234          MOV     A,X1H
 0091 37          235          CPL     A
 0092 1300        236          ADDC    A,#0
 0094 A0          237          MOV     @POINR1,A
 0095 18          238          INC     POINR1
 0096 FE          239          MOV     A,X1S
 0097 37          240          CPL     A
 0098 1300        241          ADDC    A,#0
 009A A0          242          MOV     @POINR1,A
 009B B823        243          MOV     POINR1,#X2L
 009D B920        244          MOV     POINR2,#XENDL
 009F F0          245          MOV     A,@POINR1
 00A0 61          246          ADD     A,@POINR2
 00A1 A0          247          MOV     @POINR1,A
 00A2 18          248          INC     POINR1
 00A3 19          249          INC     POINR2
 00A4 F0          250          MOV     A,@POINR1
 00A5 71          251          ADDC    A,@POINR2
 00A6 A0          252          MOV     @POINR1,A
 00A7 18          253          INC     POINR1
 00A8 19          254          INC     POINR2
 00A9 F0          255          MOV     A,@POINR1
 00AA 71          256          ADDC    A,@POINR2
 00AB A0          257          MOV     @POINR1,A
                  258 $EJECT
```

```
ISIS-II MCS-48/UPI-41 MACRO ASSEMBLER, V4.0            PAGE    8
     SERVO&CONTROLLER

 LOC  OBJ         LINE        SOURCE STATEMENT

                   259 ; XSP=X2-DX
                   260
 00AC B925         261          MOV     POINR2,#X2S
 00AE B832         262          MOV     POINR1,#XSPS
 00B0 F1           263          MOV     A,@POINR2
 00B1 A0           264          MOV     @POINR1,A
 00B2 B823         265          MOV     POINR1,#X2L
 00B4 B930         266          MOV     POINR2,#XSPL
 00B6 1206         267          JB0     ADIDX
 00B8 FB           268 SUBDX:   MOV     A,BX
 00B9 37           269          CPL     A
 00BA 17           270          INC     A
 00BB 60           271          ADD     A,@POINR1
 00BC A1           272          MOV     @POINR2,A
 00BD 18           273          INC     POINR1
 00BE 19           274          INC     POINR2
 00BF F0           275          MOV     A,@POINR1
 00C0-13FF         276          ADDC    A,#0FFH
 00C2 A1           277          MOV     @POINR2,A
 00C3 04CE         278          JMP     TG3
 00C5 FB           279 ADDX:    MOV     A,DX
 00C6 60           280          ADD     A,@POINR1
 00C7 A1           281          MOV     @POINR2,A
 00C8 18           282          INC     POINR1
 00C9 19           283          INC     POINR2
 00CA F0           284          MOV     A,@POINR1
 00CB 1300         285          ADDC    A,#0
 00CD A1           286          MOV     @POINR2,A
                   287
                   288 ; 3 IF(ABS(XCOM).GE.ABS(X2))GOTO 30  ....THE BEGINNING OF THE
                   289 ;                                        RUN PART.
                   290
 00CE B827         291 TG3:     MOV     POINR1,#XCOML
 00D0 B923         292          MOV     POINR2,#X2L
 00D2 F0           293          MOV     A,@POINR1
 00D3 37           294          CPL     A
 00D4 61           295          ADD     A,@POINR2
 00D5 37           296          CPL     A
 00D6 AA           297          MOV     TMP,A
 00D7 18           298          INC     POINR1
 00D8 19           299          INC     POINR2
 00D9 F0           300          MOV     A,@POINR1
 00DA 2400         301          JMP     PAGE1
                   302 $EJECT
```

```
ISIS-II MCS-48/UPI-41 MACRO ASSEMBLER, V4.0            PAGE    9
     SERVO&CONTROLLER

 LOC  OBJ        LINE        SOURCE STATEMENT

 0100            303         ORG     100H; ......... MEMORY PAGE 1  ...
 0100 37         304 PAGE1:  CPL     A
 0101 71         305         ADDC    A,@POINR2
 0102 37         306         CPL     A
 0103 7609       307         JF1     NXCNX2
 0105 F610       308         JC      TG10; ...........XCOM<X2
 0107 243F       309         JMP     TG30
 0109 F63F       310 NXCNX2: JC      TG30
 010B 9610       311         JNZ     TG10
 010D FA         312         MOV     A,TMP
 010E 9610       313         JNZ     TG10
                 314 ;       JMP     TG10
                 315
                 316 ; 10  IF(ABS(XCOM).LT.ABS(XSP))GOTO 1
                 317
 0110 B827       318 TG10:   MOV     POINR1,#XCOML
 0112 B930       319         MOV     POINR2,#XSPL
 0114 F0         320         MOV     A,@POINR1
 0115 37         321         CPL     A
 0116 61         322         ADD     A,@POINR2
 0117 37         323         CPL     A
 011B AA         324         MOV     TMP,A
 0119 18         325         INC     POINR1
 011A 19         326         INC     POINR2
 011B F0         327         MOV     A,@POINR1
 011C 37         328         CPL     A
 011D 71         329         ADDC    A,@POINR2
 011E 37         330         CPL     A
 011F 7625       331         JF1     NXCNXS
 0121 F669       332         JC      TG11; ...........XCOM>XSP
 0123 242C       333         JMP     TG15
 0125 F62C       334 NXCNXS: JC      TG15
 0127 9669       335         JNZ     TG11
 0129 FA         336         MOV     A,TMP
 012A 9669       337         JNZ     TG11
                 338 ;       JMP     TG15
                 339
                 340
                 341 ; 15  XCOM=X2  ..................BEGINNING OF THE DECELERATION PART
                 342
 012C B827       343 TG15:   MOV     POINR1,#XCOML
 012E B923       344         MOV     POINR2,#X2L
 0130 F1         345         MOV     A,@POINR2
 0131 A0         346         MOV     @POINR1,A
 0132 18         347         INC     POINR1
                 348 $EJECT
```

```
ISIS-II MCS-48/UPI-41 MACRO ASSEMBLER, V4.0            PAGE   10
     SERVO&CONTROLLER

 LOC  OBJ        LINE        SOURCE STATEMENT

 0133 19          349          INC     POINR2
 0134 F1          350          MOV     A,@POINR2
 0135 A0          351          MOV     @POINR1,A
 0136 18          352          INC     POINR1
 0137 19          353          INC     POINR2
 0138 F1          354          MOV     A,@POINR2
 0139 A0          355          MOV     @POINR1,A
                  356
                  357 ; GOTO 1
                  358
 013A 0424        359          JMP     LOOP
                  360
                  361 ; 20     DX=DX+1
                  362
 013C 1B          363 TG20:    INC     DX
                  364
                  365 ; GOTO 1
                  366
 013D 0424        367          JMP     LOOP
                  368
                  369 ; 30     DX=DX-1
                  370
 013F CB          371 TG30:    DEC     DX
                  372
                  373 ; DXMAX=DX
                  374
 0140 FB          375          MOV     A,DX
 0141 AF          376          MOV     DXMAX,A
                  377
                  378 ; IF(XCOM.NE.XEND)GOTO 1
                  379
 0142 B82B        380          MOV     POINR1,#XCOMH
 0144 B921        381          MOV     POINR2,#XENDH
 0146 F1          382          MOV     A,@POINR2
 0147 D0          383          XRL     A,@POINR1
 0148 9669        384          JNZ     TG11;............XCOM()XEND
 014A C8          385          DEC     POINR1
 014B C9          386          DEC     POINR2
 014C F1          387          MOV     A,@POINR2
 014D D0          388          XRL     A,@POINR1
 014E 9669        389          JNZ     TG11;  ..........XCOM()XEND
                  390 ;        JMP     TG40;  ..........XCOM=XEND
                  391 $EJECT
```

```
ISIS-II MCS-48/UPI-41 MACRO ASSEMBLER, V4.0            PAGE   11
     SERVO&CONTROLLER

 LOC  OBJ        LINE        SOURCE STATEMENT

                  392 ; 40    SET FLAG ...............MOTION IS COMPLETED
                  393
 0150 B826        394 TG40:   MOV     POINR1,#FLAG
 0152 F0          395         MOV     A,@POINR1
 0153 4301        396         ORL     A,#001H
 0155 A0          397         MOV     @POINR1,A
                  398
                  399 ; 50    IF(VCOM.EQ.0)OUTPUT DONE
                  400
 0156 BB2F        401 TG50:   MOV     POINR1,#VCOMS
 0158 F0          402         MOV     A,@POINR1
 0159 9667        403         JNZ     CLDAC
 015B C8          404         DEC     POINR1
 015C F0          405         MOV     A,@POINR1
 015D 9667        406         JNZ     CLDAC
 015F C8          407         DEC     POINR1
 0160 F0          408         MOV     A,@POINR1
 0161 9667        409         JNZ     CLDAC
 0163 9ADF        410 ZERO1:  ANL     P2,#DONE
 0165 BA20        411         ORL     P2,#NODONE
 0167 5400        412 CLDAC:  CALL    DAC
 0169 0424        413 TG11:   JMP     LOOP
                  414 $EJECT
```

```
LOC  OBJ        LINE        SOURCE STATEMENT

                415
                416
0200            417          ORG     200H;  ......... MEMORY PAGE 2 ...
                418        ;_____________________________________________;
                419        ;                                             ;
                420        ;    THIS SUBROUTINE IS USED TO CALCULATE     ;
                421        ;   THE POSITIONAL ERROR AND CONVERT IT TO    ;
                422        ;   A VELOCITY COMMAND TO DRIVE THE SERVO      ;
                423        ;   MOTOR IN A DIRECTION THAT WILL MINIM-     ;
                424        ;   IZE THIS ERROR.                           ;
                425        ;_____________________________________________;
                426
                427
0200 B82D       428 BAC:     MOV     POINR1,#VCOML
0202 B92A       429          MOV     POINR2,#XACTL
0204 F1         430          MOV     A,@POINR2
0205 37         431          CPL     A
0206 0301       432          ADD     A,#001H
0208 A0         433          MOV     @POINR1,A
0209 18         434          INC     POINR1
020A 19         435          INC     POINR2
020B F1         436          MOV     A,@POINR2
020C 37         437          CPL     A
020D 1300       438          ADDC    A,#0
020F A0         439          MOV     @POINR1,A
0210 18         440          INC     POINR1
0211 19         441          INC     POINR2
0212 F1         442          MOV     A,@POINR2
0213 37         443          CPL     A
0214 1300       444          ADDC    A,#0
0216 A0         445          MOV     @POINR1,A
0217 B82D       446          MOV     POINR1,#VCOML
0219 B927       447          MOV     POINR2,#XCOML
021B F0         448          MOV     A,@POINR1
021C 61         449          ADD     A,@POINR2
021D A0         450          MOV     @POINR1,A
021E 18         451          INC     POINR1
021F 19         452          INC     POINR2
0220 F0         453          MOV     A,@POINR1
0221 71         454          ADDC    A,@POINR2
0222 A0         455          MOV     @POINR1,A
0223 18         456          INC     POINR1
0224 19         457          INC     POINR2
0225 F0         458          MOV     A,@POINR1
0226 71         459          ADDC    A,@POINR2
0227 A0         460          MOV     @POINR1,A
                461 $EJECT
```

```
LOC  OBJ        LINE        SOURCE STATEMENT

0228 B82F        462 CHEK:   MOV     POINR1,#VCOMS
022A F0          463         MOV     A,@POINR1
022B C8          464         DEC     POINR1
022C 123A        465         JB0     NGVCOM
022E F0          466         MOV     A,@POINR1
022F 53FE        467         ANL     A,#0FEH
0231 C645        468         JZ      OUTSRV
0233 B001        469         MOV     @POINR1,#001H
0235 C8          470         DEC     POINR1
0236 B0FF        471         MOV     @POINR1,#0FFH
0238 4445        472         JMP     OUTSRV
023A F0          473 NGVCOM: MOV     A,@POINR1
023B 37          474         CPL     A
023C 53FE        475         ANL     A,#0FEH
023E C645        476         JZ      OUTSRV
0240 B0FE        477         MOV     @POINR1,#0FEH
0242 C8          478         DEC     POINR1
0243 B000        479         MOV     @POINR1,#00H
0245 B82D        480 OUTSRV: MOV     POINR1,#VCOML;  .......OUTPUT VCOM TO SERVO
0247 F0          481         MOV     A,@POINR1
0248 39          482         OUTL    P1,A
0249 8820        483         ORL     BUS,#ELSDAL
024B 98DF        484         ANL     BUS,#DLSDAL
024D 18          485         INC     POINR1
024E F0          486         MOV     A,@POINR1
024F 39          487         OUTL    P1,A
0250 8810        488         ORL     BUS,#ELSDAH
0252 98EF        489         ANL     BUS,#DLSDAH
0254 83          490         RET
                 491 $EJECT
```

```
LOC  OBJ        LINE        SOURCE STATEMENT

                 492
                 493
                 494      ;_____________________________________________;
                 495      ;                                             ;
                 496      ;   THIS SUBROUTINE IS CALLED WHEN THE "IHB"  ;
                 497      ;  IS FULL. IT WILL THEN READ TWO WORDS AND   ;
                 498      ;  RETURN TO THE MAIN PROGRAM.                ;
                 499      ;                                             ;
                 500      ;_____________________________________________;
                 501
                 502
0255 B81F        503 READ:   MOV     POINR1,#1FH
0257 5657        504 WAIT:   JT1     WAIT
0259 98FD        505         ANL     BUS,#CRFBK
025B 8842        506         ORL     BUS,#42H
025D 09          507         IN      A,P1
025E A0          508         MOV     @POINR1,A
025F 18          509         INC     POINR1
0260 98BF        510         ANL     BUS,#DIHBL
0262 8880        511         ORL     BUS,#EIHBH
0264 09          512         IN      A,P1
0265 A0          513         MOV     @POINR1,A
0266 1B          514         INC     POINR1
0267 9ADF        515         ANL     P2,#DONE
0269 987F        516         ANL     BUS,#DIHBH
026B 8A20        517         ORL     P2,#NODONE
026D 95          518         CPL     F0
026E B657        519         JF0     WAIT
0270 83          520         RET
                 521 $EJECT
```

```
ISIS-II MCS-48/UPI-41 MACRO ASSEMBLER, V4.0            PAGE   15
     SERVO$CONTROLLER

 LOC  OBJ        LINE        SOURCE STATEMENT

                  522
                  523
                  524       ;_____________________________________;
                  525       ;                                     ;
                  526       ;    THIS SUBROUTINE IS USED TO WRITE ;
                  527       ;   ONE WORD TO THE "OWB". IT IS CALLED ;
                  528       ;   IN THE TEACH MODE AND WHEN THE    ;
                  529       ;   ACTUAL POSITION IS UPDATED ON THE ;
                  530       ;   CRT.                              ;
                  531       ;                                     ;
                  532       ;_____________________________________;
                  533
                  534
 0271 B82B        535 WRITE:  MOV     POINR1,#XACTH
 0273 F0          536         MOV     A,@POINR1
 0274 39          537         OUTL    P1,A
 0275 8808        538         ORL     BUS,#E0HBH
 0277 98F7        539         ANL     BUS,#D0HBH
 0279 C8          540         DEC     POINR1
 027A F0          541         MOV     A,@POINR1
 027B 39          542         OUTL    P1,A
 027C 8804        543         ORL     BUS,#E0HBL
 027E 98FB        544         ANL     BUS,#D0HBL
 0280 83          545         RET
 000C             546 END     START
```

APPENDIX C

The following pages contain the components' locations and the schematics of the Servo Control Board. Also, the schematic for the manual controller is included.

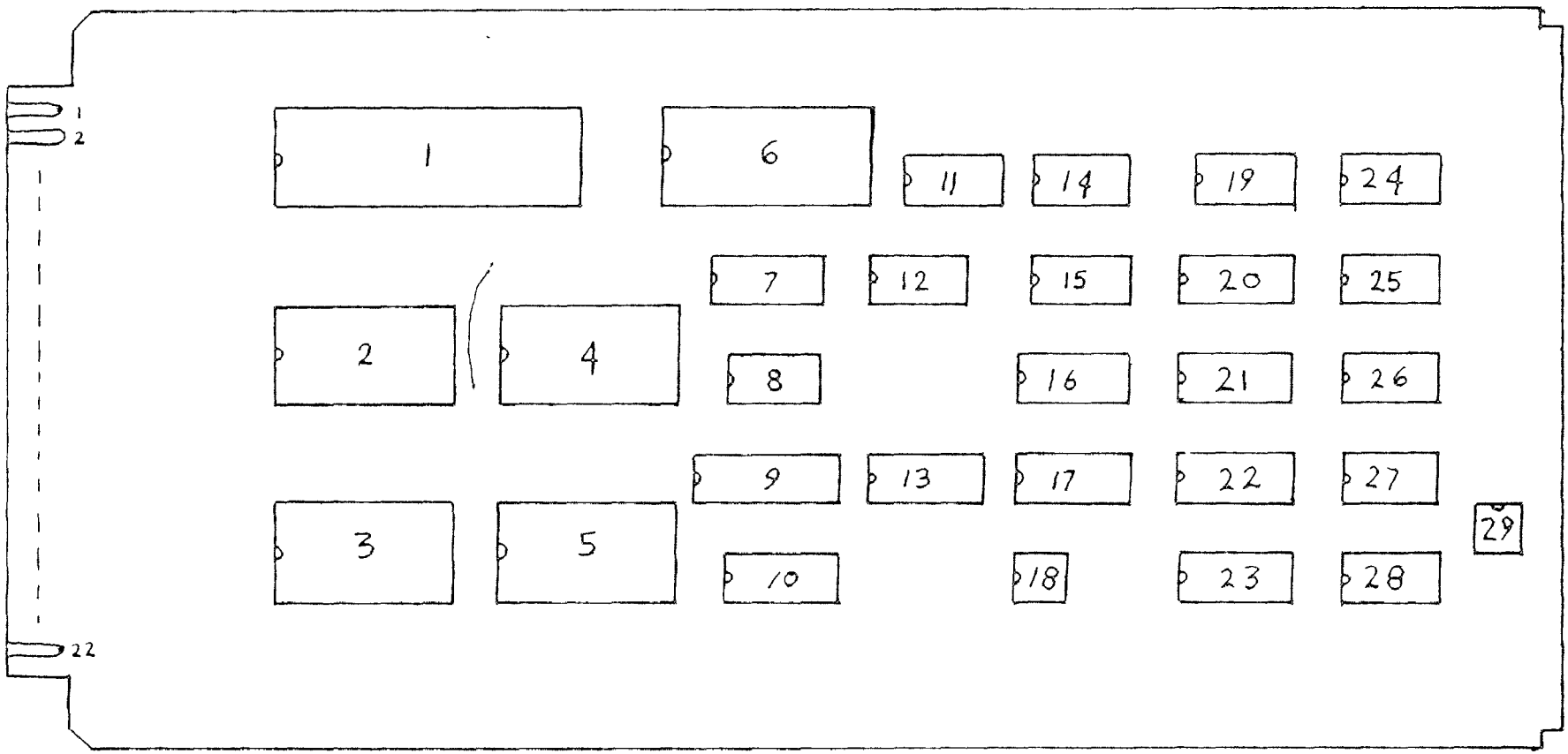


Figure C-1 Control Board Parts Location Diagram

Socket No.	Chip No.	No.of Pins	GND	+5V	+15V	-15V
1	8748	40	20	40	--	--
2	8212	24	12	24	--	--
3	8212	24	12	24	--	--
4	8212	24	12	24	--	--
5	8212	24	12	24	--	--
6	7522	28	28	27	1	--
7	0015D	16	3	14	4	13
8	74126	14	7	14	--	--
9	8282	20	10	20	--	--
10	74193	16	8	16	--	--
11	747C	14	--	--	9,13	4
12	7474	14	7	14	--	--
13	74193	16	8	16	--	--
14	747C	14	--	--	9,13	4
15	RC	14	--	--	--	--
16	RDC	16	--	--	--	--
17	74123	16	8	16	--	--
18	RC	8/14	--	--	--	--
19	7404	14	7	14	--	--
20	74123	16	8	16	--	--
21	74123	16	8	16	--	--
22	7483	16	8	16	--	--
23	R	16	--	--	--	--
24	7408	14	7	14	--	--
25	7474	14	7	14	--	--
26	7474	14	7	14	--	--
27		14				
28	319N	14	6	11	--	--
29	DIP SW.	8	--	--	--	--

RC: Resistor/Capacitor Bank
RDC: Resistor/Diode/Capacitor Bank
R: Resistors Bank

TABLE C-1 Device Power Supply

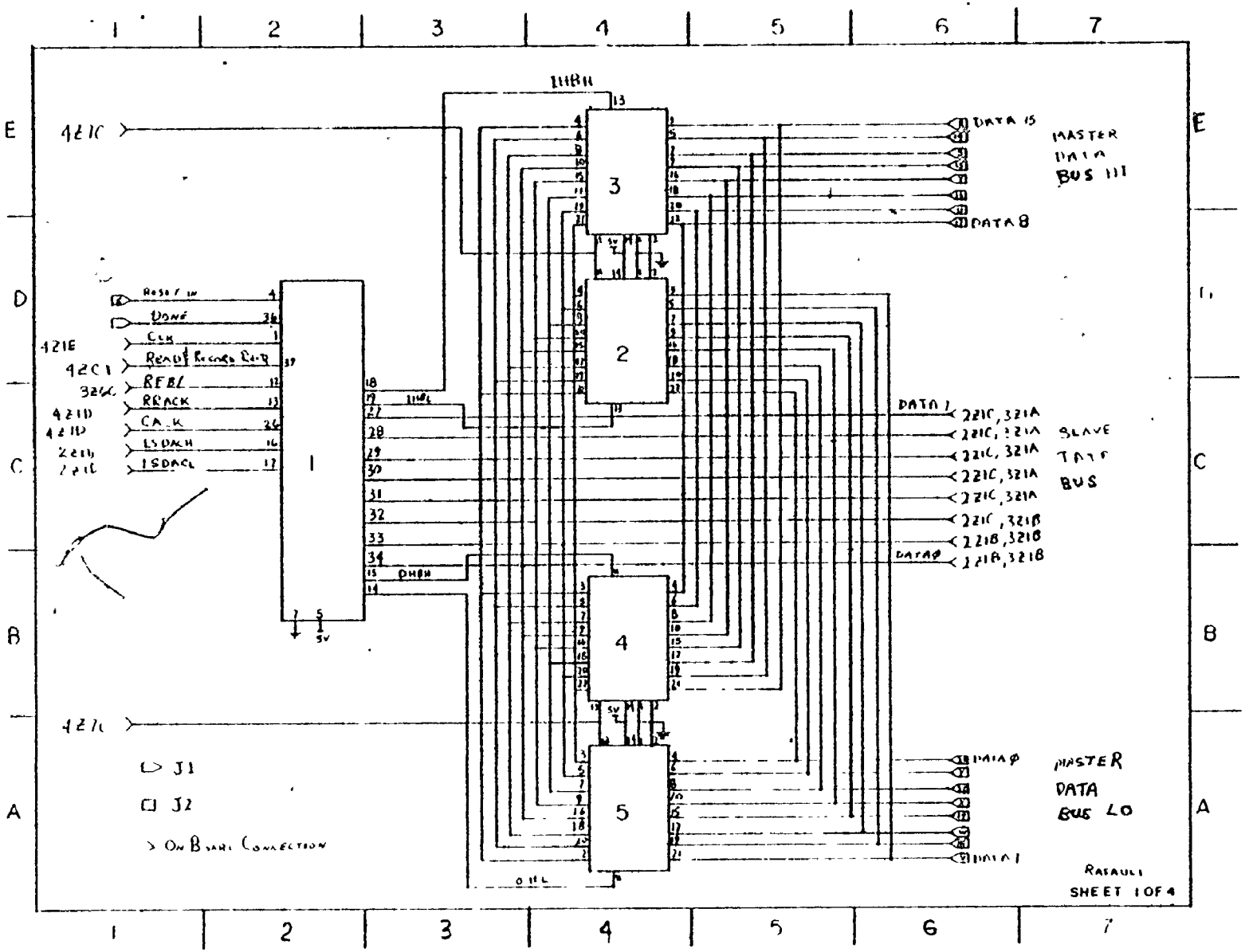


Figure C-2 Control Board Schematic Diagram

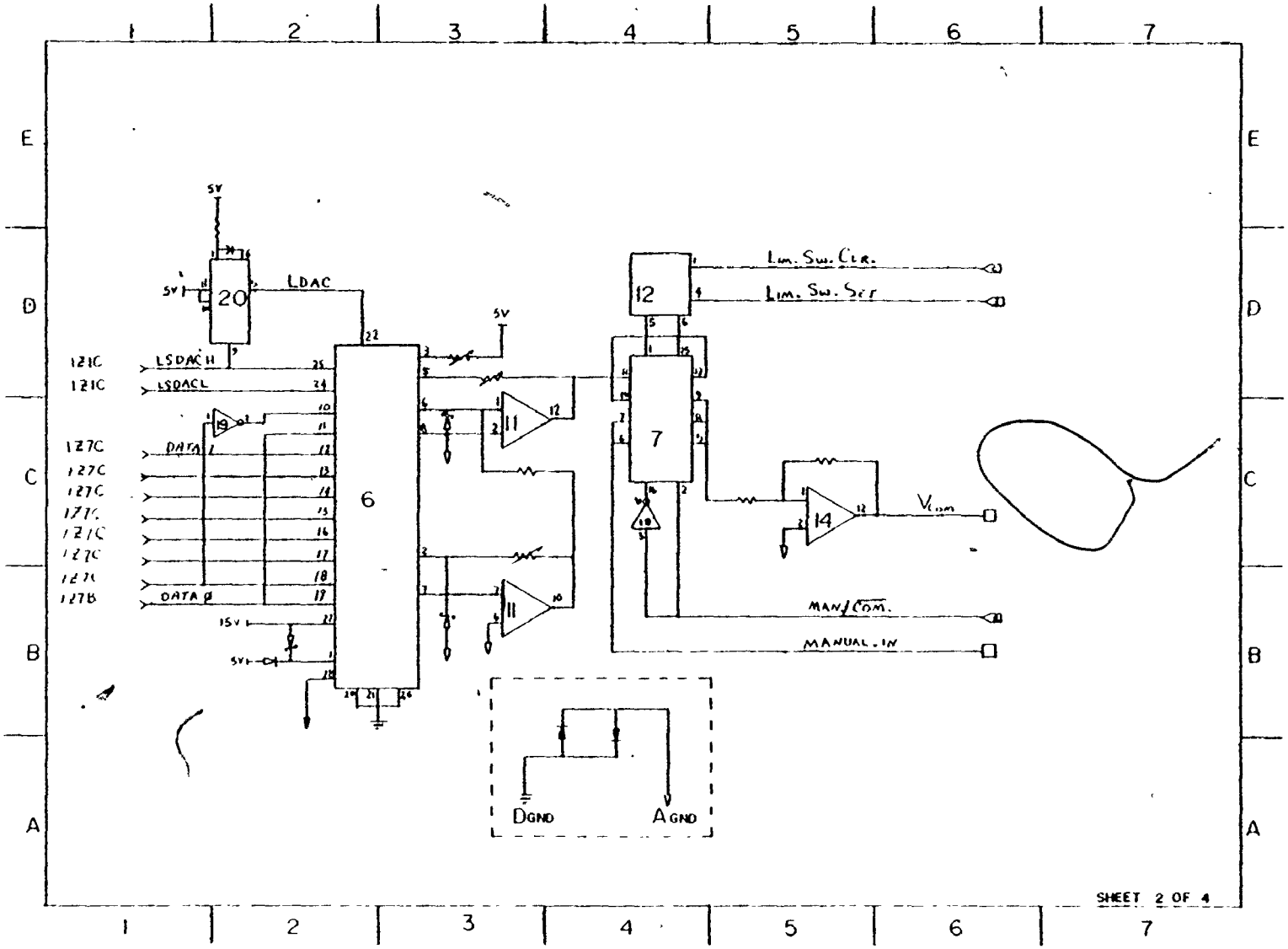


Figure C-2 Control Board Schematic Diagram

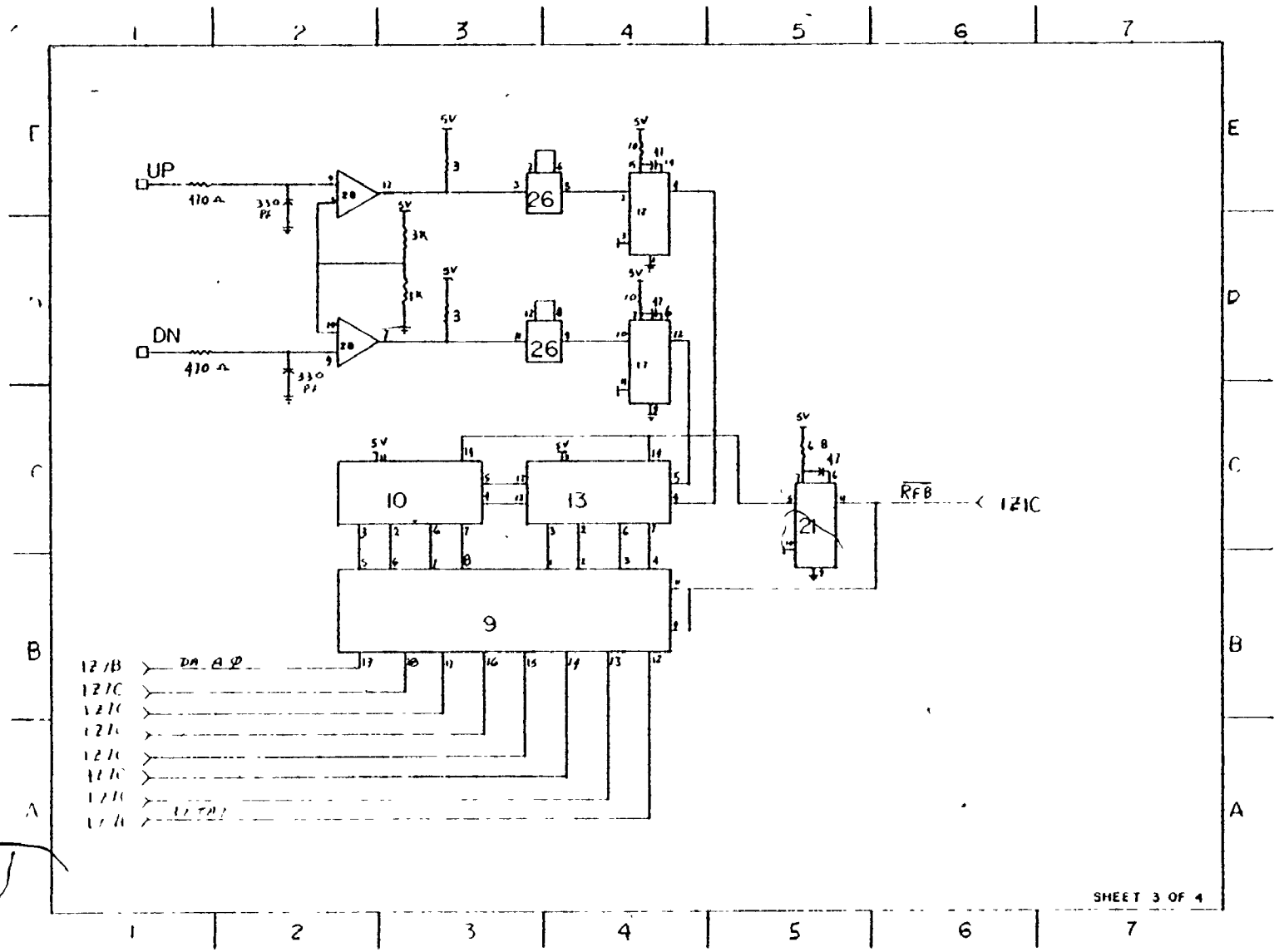


Figure C-2 Control Board Schematic Diagram

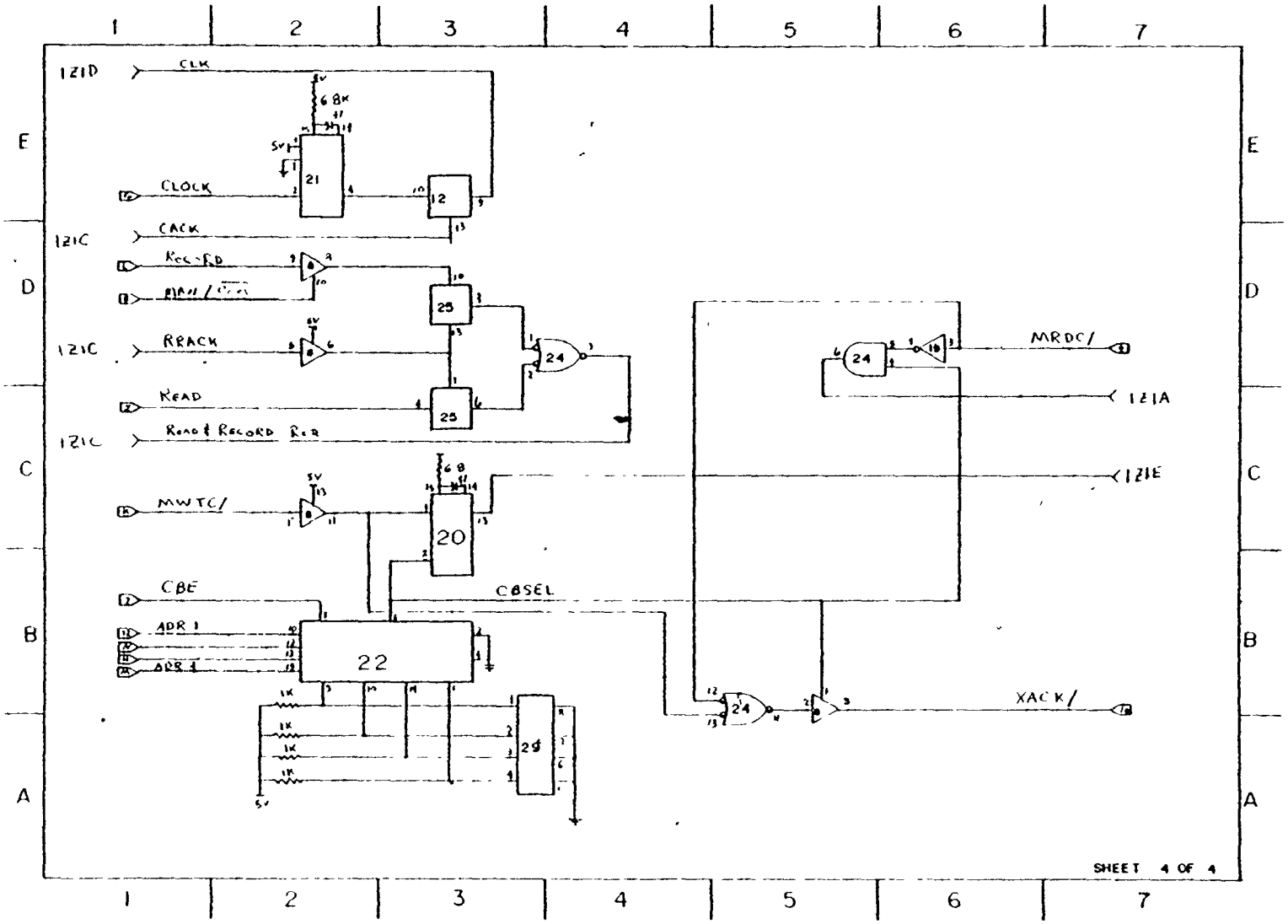


Figure C-2 Control Board Schematic Diagram

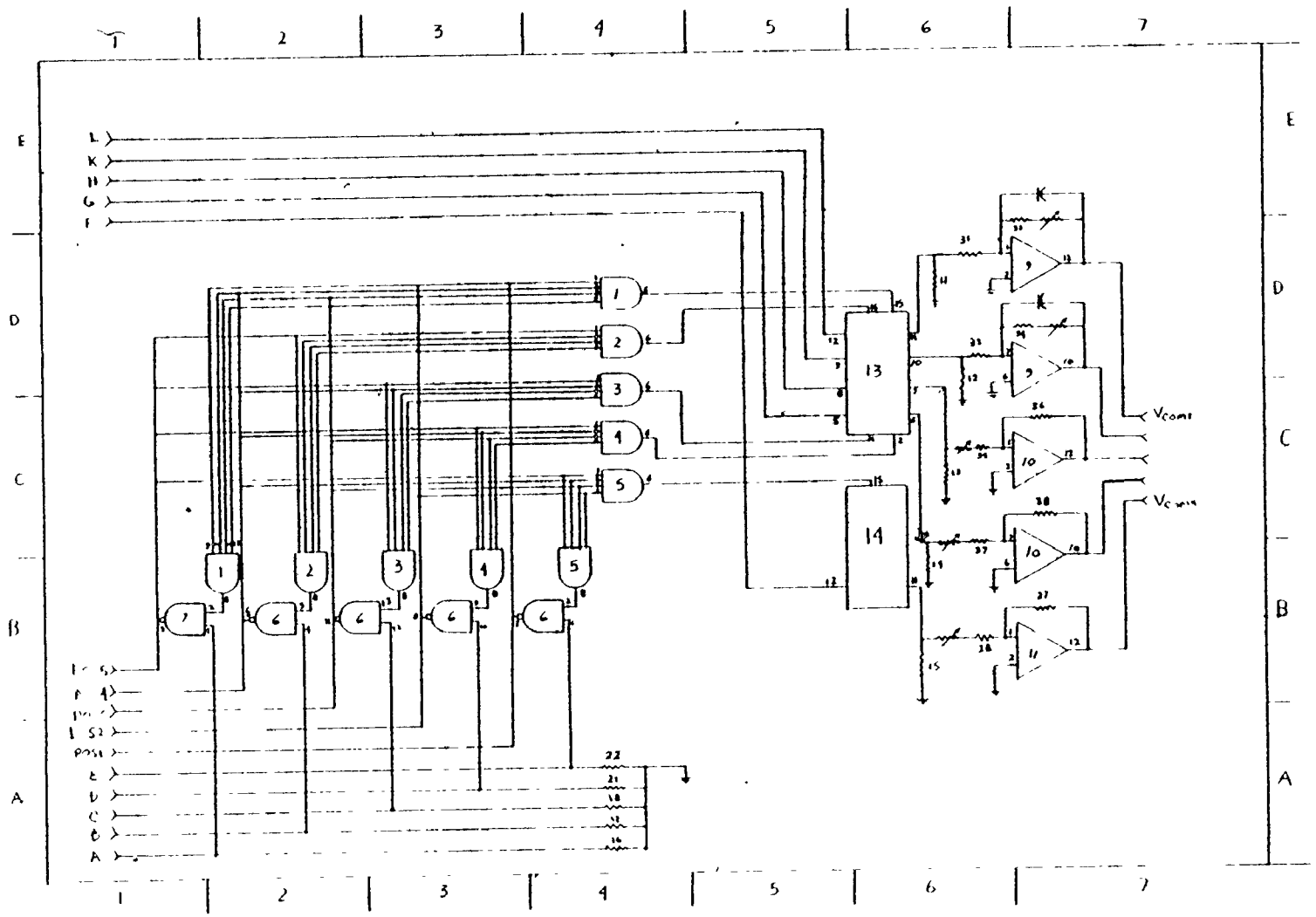


Figure C-3 Manual Control Board Schematic Diagram

APPENDIX D

iSBC86/12A Monitor Commands

The monitor is capable of executing five individual commands. Each command is described within the sections which follow. The following command syntax is used:

[A]	indicates that "A" is optional
[A]*	indicates one or more optional occurrences of "A"
<cr>	indicates a carriage return is entered
<sp>	indicates a space is entered

(S) Examine and Substitute Memory:

Memory locations "seg:off" are displayed on the CRT and are open for modification through the keyboard. These can be displayed either in bytes or words of "w" is typed after the command character "s".

SYNTAX

S [W] <addr> <sp> [[<new contents>] <sp>]* <cr>

(E) Examine and Modify Registers:

Internal register pairs are displayed in sequence if the command character "E" is followed by the carriage return key. Any register pair can be displayed and modified if it's name follows the command character "E".

SYNTAX

E [<reg>][[<new contents>] <sp>]* <cr>

(D) Display Memory:

Memory locations "seg:off" to the new "seg:off" are displayed on the CRT. If "w" follows the command character "D", then memory is displayed in word format.

SYNTAX

D [W] <start addr> [<sp> <end addr] <cr>

(M) Move Memory:

This command moves a block of memory between starting "seg:off" and end "seg:off" to the new address "seg:off".

SYNTAX

M <start addr> <sp> <end addr> <sp> <destination addr> <cr>

(G) GO Command:

The GO command is followed by starting address "seg:off" and control is transferred to the given address.

SYNTAX

G <addr> <cr>

```
PL/M-86 COMPILER          MONITOR$COMMAND$MODULE                    01 MAY 81  PAGE   1

ISIS-II PL/M-86 V2.1 COMPILATION OF MODULE MONITORCOMMANDMODULE
OBJECT MODULE PLACED IN :F1:MONCMD.OBJ
COMPILER INVOKED BY:  PLM86 :F1:MONCMD.SRC LARGE OPTIMIZE(3) TITLE('      MONITOR$COMMAND$MODULE')
                  -&PAGEWIDTH(99) DATE(01 MAY 81)


  1         MONITOR$COMMAND$MODULE:DO;

                  /*____________________________________________
                  |                                            |
                  |        ROBOT,iSBC 86/12A Monitor V1.0      |
                  |        Dec. 1980                           |
                  |        Programmer R.Rafauli                |
                  |                                            |
                  |         M O N I T O R   C O M M A N D      |
                  |                                            |
                  |          This module supports the iSBC 86/12A
                  |        hardware                            |
                  |                                            |
                  |                                            |
                  |          Monitor commands:                 |
                  |             S - Substitute memory          |
                  |             E - Examine registers          |
                  |             D - Display memory             |
                  |             M - Move memory                |
                  |             G - Go to a given address      |
                  |                 and start execution        |
                  |____________________________________________*/
```

```
$EJECT
        /*_____________________________________________
        |                                             |
        |        L O C A L   D E C L A R A T I O N S  |
        |_____________________________________________*/

2  1        DECLARE
               CHAR        BYTE EXTERNAL,
               I           BYTE,
               END$OFF     WORD,
               WORD$MODE   BYTE EXTERNAL,
               TRUE        LITERALLY'0FFH',
               FALSE       LITERALLY'000H',
               STATP       LITERALLY'100H',
               ASCR        LITERALLY'00DH',
               ASBL        LITERALLY'020H';
3  1        DECLARE
               MEMOARG1PTR POINTER,
               ARG1 STRUCTURE  (OFF WORD,SEG WORD)
               AT(@MEMOARG1PTR),
               MEMOARG1 BASED MEMOARG1PTR  BYTE,
               MEMOWORDARG1 BASED MEMOARG1PTR  WORD,
               MEMOARG3PTR POINTER,
               ARG3 STRUCTURE  (OFF WORD,SEG WORD)
               AT(@MEMOARG3PTR),
               MEMOARG3 BASED MEMOARG3PTR  BYTE,
               MEMORYCSIPPTR POINTER,
               CSIP STRUCTURE (OFF WORD,SEG WORD)
               AT(@MEMORYCSIPPTR),
               MEMORYCSIP BASED MEMORYCSIPPTR  BYTE,
               MEMORYUSERSTACKPTR POINTER,
               USERSTACK STRUCTURE (OFF WORD,SEG WORD)
               AT(@MEMORYUSERSTACKPTR),
               MEMORYUSERSTACK BASED MEMORYUSERSTACKPTR WORD;
4  1        DECLARE
               REG(*)      BYTE DATA('AXBXCXDXSPBPSIDICSDSSSESIPFL'),
               REGINDX     WORD    PUBLIC,
               REGSAV(14)  WORD    EXTERNAL,
               REGORD(*)   BYTE DATA(7,6,1,3,2,0,9,11,12,8,13),
               SP      LITERALLY'REGSAV(4)',
               BP      LITERALLY'REGSAV(5)',
               CS      LITERALLY'REGSAV(8)',
               DS      LITERALLY'REGSAV(9)',
               SS      LITERALLY'REGSAV(10)',
               ES      LITERALLY'REGSAV(11)',
               IP      LITERALLY'REGSAV(12)',
               FL      LITERALLY'REGSAV(13)';
```

```
                $EJECT
                        /*_____________________________________________
                        |                                             |
                        |        EXTERNAL PROCEDURE DECLARATIONS      |
                        |_____________________________________________*/

  5   1     DECLARE     ERROR   LABEL   EXTERNAL;

  6   1     MAC$BLNK$SCAN:  PROCEDURE   EXTERNAL;
  7   2     END MAC$BLNK$SCAN;

  8   1     MAC$ADDRES$IN:PROCEDURE (PTR,DEFULTBASE)    EXTERNAL;
  9   2         DECLARE PTR POINTER,
                    DEFULTBASE WORD;
 10   2     END MAC$ADDRES$IN;

 11   1     MAC$TST$WORD$MOD:PROCEDURE  EXTERNAL;
 12   2     END MAC$TST$WORD$MOD;

 13   1     MAC$CHAR$OUT:   PROCEDURE (C)   EXTERNAL;
 14   2         DECLARE C BYTE;
 15   2     END MAC$CHAR$OUT;

 16   1     MAC$WORD$OUT:   PROCEDURE (W)   EXTERNAL;
 17   2         DECLARE W WORD;
 18   2     END MAC$WORD$OUT;

 19   1     MAC$BYTE$OUT:   PROCEDURE (B)   EXTERNAL;
 20   2         DECLARE B BYTE;
 21   2     END MAC$BYTE$OUT;

 22   1     MAC$CHAR$IN:    PROCEDURE   EXTERNAL;
 23   2     END MAC$CHAR$IN;

 24   1     MAC$WORD$IN:    PROCEDURE WORD  EXTERNAL;
 25   2     END MAC$WORD$IN;

 26   1     MAC$CRLF:   PROCEDURE   EXTERNAL;
 27   2     END MAC$CRLF;

 28   1     MAC$VALD$REG$FIRST:PROCEDURE BYTE   EXTERNAL;
 29   2     END MAC$VALD$REG$FIRST;

 30   1     MAC$VALD$REG:PROCEDURE (C1,C2) BYTE EXTERNAL;
 31   2         DECLARE (C1,C2) BYTE;
 32   2     END MAC$VALD$REG;
```

```
$EJECT
                /*_____________________________________________
                :                                            :
                :          COMMAND PROCEDURES SECTION        :
                :____________________________________________*/

/* THIS ROUTINE RESTORES THE MACHINE STATUS AND PASSES CONTROL TO
   THE USER PROGRAM. IT CONTAINS A MACHINE LANGUAGE SUBROUTINE TO
   PERFORM THE POPPING OF THE USER REGISTERS AND TO EXECUTE AN
   'IRET' TO THE USER PROGRAM.*********************************/

33  1   MAC$START$IT:
        PROCEDURE;
34  2       DECLARE START$IT$CODE(*) BYTE DATA
            (08BH,0ECH, /* MOV BP,SP          */
             08BH,46H,2H,   /* MOV AX,/BP/.PARM2  */
             8BH,5EH,04H,   /* MOV BX,/BP/.PARM1  */
             8EH,0D0H,      /* MOV SS,AX          */
             8BH,0E3H,      /* MOV SP,BX          */
             5DH,           /* POP BP             */
             5FH,           /* POP DI             */
             5EH,           /* POP SI             */
             5BH,           /* POP BX             */
             5AH,           /* POP DX             */
             59H,           /* POP CX             */
             58H,           /* POP AX             */
             1FH,           /* POP DS             */
             07H,           /* POP ES             */
             0CFH),         /* IRET               */

                START$IT$CODE$PTR WORD DATA(.START$IT$CODE);
35  2       USERSTACK.SEG=SS;
36  2       USERSTACK.OFF=SP;
37  2       DO I=0 TO 10;
38  3           USERSTACK.OFF=USERSTACK.OFF - 2;
39  3           MEMORYUSERSTACK=REGSAV(REGORD(10-I));
40  3       END;
41  2       USERSTACK.OFF=USERSTACK.OFF - 2;
42  2       MEMORYUSERSTACK=BP;
43  2       CALL    START$IT$CODE$PTR(USERSTACK.OFF,USERSTACK.SEG);
44  2   END MAC$START$IT;
```

```
            $EJECT


            /* THIS ROUTINE TRANSFERS CONTROL TO A GIVEN ADDRESS */

 45   1     MAC$GO:
            PROCEDURE   PUBLIC;
 46   2         CALL    MAC$BLNK$SCAN;
 47   2         CALL    MAC$ADDRES$IN(@CSIP,CS);
 48   2         IP=CSIP.OFF;
 49   2         CS=CSIP.SEG;
 50   2         FL=FL AND (NOT STATP);
 51   2         CALL    MAC$START$IT;
 52   2     END MAC$GO;
```

```
                $EJECT


                /* THIS ROUTINE EXAMINES AND/OR MODIFIES THE MEMORY LOCATIONS */

 53   1         MAC$EXAM$MEMO:
                PROCEDURE   PUBLIC;
 54   2            DECLARE W WORD;
 55   2            CALL MAC$TSTWORDMOD;
 56   2            CALL MAC$ADDRES$IN(@ARG1,CS);
 57   2            IF CHAR()ASBL THEN GOTO ERROR;
 59   2            DO WHILE TRUE;
 60   3                CALL MAC$CHAR$OUT(ASBL);
 61   3                IF WORD$MODE THEN
 62   3                CALL MAC$WORD$OUT(MEMOWORDARG1);
                       ELSE
 63   3                CALL MAC$BYTE$OUT(MEMOARG1);
 64   3                CALL MAC$CHAR$OUT('_');
 65   3                CALL MAC$CHAR$OUT(ASBL);
 66   3                CALL MAC$CHAR$IN;
 67   3                IF CHAR=ASCR THEN RETURN;
 69   3                IF CHAR()ASBL THEN
 70   3                DO;
 71   4                    W=MAC$WORD$IN;
 72   4                    IF(CHAR()ASBL) AND (CHAR()ASCR) THEN GOTO ERROR;
 74   4                    IF WORD$MODE THEN
 75   4                        MEMOWORDARG1=W;
                           ELSE
 76   4                        MEMOARG1=LOW(W);
 77   4                    END;
 78   3                IF CHAR =ASCR THEN RETURN;
 80   3                IF WORD$MODE THEN
 81   3                    ARG1.OFF=ARG1.OFF+2;
                       ELSE
 82   3                    ARG1.OFF=ARG1.OFF+1;
 83   3                CALL MAC$CRLF;
 84   3                CALL MAC$WORD$OUT(ARG1.OFF);
 85   3            END;
 86   2        END MAC$EXAMMEMO;
```

```
                $EJECT


                /* THIS ROUTINE EXAMINES AND/OR MODIFIES THE REGISTERS */

 87   1         MAC$EXAM$REG:
                PROCEDURE   PUBLIC;
 88   2            DECLARE (T,I)   BYTE,
                      SAVE    WORD;
 89   2            CALL MAC$BLNK$SCAN;
 90   2            IF CHAR=ASCR THEN
 91   2            DO;
 92   3            CALL MAC$CRLF;
 93   3            DO I=0 TO 13;
 94   4               CALL MAC$CHAR$OUT(ASBL);
 95   4               CALL MAC$CHAR$OUT (REG(I*2));
 96   4               CALL MAC$CHAR$OUT(REG(I*2+1));
 97   4               CALL MAC$CHAR$OUT('=');
 98   4               CALL MAC$WORD$OUT(REGSAV(I));
 99   4               IF I=6 THEN CALL MAC$CRLF;
101   4            END;
102   3            RETURN;
103   3            END;
104   2            IF NOT(MAC$VALDREGFIRST) THEN GOTO ERROR;
106   2            I=CHAR;
107   2            CALL MAC$CHAR$IN;
108   2            IF NOT(MAC$VALDREG(T,CHAR)) THEN GOTO ERROR;
110   2            I=REGINDX;
111   2            DO WHILE TRUE;
112   3               CALL MAC$CHAR$OUT('=');
113   3               CALL MAC$WORD$OUT(REGSAV(I));
114   3               CALL MAC$CHAR$OUT('_');
115   3               CALL MAC$CHAR$OUT(ASBL);
116   3               CALL MAC$CHAR$IN;
117   3               IF CHAR()ASBL AND CHAR()ASCR THEN
118   3               DO;
119   4                  SAVE=MAC$WORD$IN;
120   4                  IF CHAR()ASBL AND (CHAR()ASCR) THEN GOTO ERROR;
122   4                  REGSAV(I)=SAVE;
123   4               END;
124   3               IF CHAR =ASCR OR I=13 THEN RETURN;
126   3               I=I+1;
127   3               CALL MAC$CRLF;
128   3               CALL MAC$CHAR$OUT(REG(I*2));
129   3               CALL MAC$CHAR$OUT(REG(I*2+1));
130   3            END;
131   2         END MAC$EXAMREG;
```

```
                $EJECT

                /* THIS ROUTINE MOVES A BYTE OR A BLOCK OF DATA IN MEMORY */

 132   1        MAC$MOVE:
                PROCEDURE   PUBLIC;
 133   2           CALL MAC$BLNK$SCAN;
 134   2           CALL MAC$ADDRES$IN(@ARG1,CS);
 135   2           IF CHAR()ASBL THEN GOTO ERROR;
 137   2           CALL MAC$CHAR$IN;
 138   2           END$OFF=MAC$WORD$IN;
 139   2           IF END$OFF<ARG1.OFF THEN GOTO ERROR;
 141   2           IF CHAR()ASBL THEN GOTO ERROR;
 143   2           CALL MAC$CHAR$IN;
 144   2           CALL MAC$ADDRES$IN(@ARG3,CS);
 145   2           IF CHAR()ASCR THEN GOTO ERROR;
 147   2           CALL MAC$CRLF;
 148   2        LOOP:   MEMOARG3=MEMOARG1;
 149   2           IF MEMOARG3()MEMOARG1 THEN GOTO ERROR;
 151   2           IF ARG1.OFF=END$OFF THEN RETURN;
 153   2           ARG1.OFF=ARG1.OFF+1;
 154   2           ARG3.OFF=ARG3.OFF+1;
 155   2           GOTO LOOP;
 156   2        END MAC$MOVE;
```

```
                $EJECT


                /* THIS ROUTINE DISPLAYS A BLOCK OF MEMORY */

157   1         MAC$DUMP:
                PROCEDURE   PUBLIC;
158   2             DECLARE T   BYTE;
159   2             CALL MAC$TSTWORDMOD;
160   2             CALL MAC$ADDRES$IN(@ARG1,CS);
161   2             IF CHAR=ASCR THEN
162   2                 END$OFF=ARG1.OFF;
                    ELSE
163   2                 DO;
164   3                     IF CHAR()ASBL THEN GOTO ERROR;
166   3                     CALL MAC$CHAR$IN;
167   3                     END$OFF=MAC$WORD$IN;
168   3                     IF END$OFF<ARG1.OFF THEN GOTO ERROR;
170   3                     IF CHAR()ASCR THEN GOTO ERROR;
172   3                 END;
173   2             NEXTLINE:   CALL MAC$CRLF;
174   2                     CALL MAC$WORD$OUT(ARG1.OFF);
175   2                 LOOP1:  CALL MAC$CHAR$OUT(ASBL);
176   2                     IF WORD$MODE THEN
177   2                     DO;
178   3                         CALL MAC$WORD$OUT(MEMOWORDARG1);
179   3                         IF ARG1.OFF=END$OFF THEN RETURN;
181   3                         ARG1.OFF=ARG1.OFF+1;
182   3                     END;
                            ELSE
183   2                         CALL MAC$BYTE$OUT(MEMOARG1);
184   2                         IF ARG1.OFF)=END$OFF THEN RETURN;
186   2                         ARG1.OFF=ARG1.OFF+1;
187   2                         T=ARG1.OFF AND 000FH;
188   2                         IF T=0 OR(WORD$MODE AND T=1)THEN GOTO NEXTLINE;
190   2                         GOTO LOOP1;
191   2         END MAC$DUMP;
192   1         END MONITOR$COMMAND$MODULE;
```