THE UNIVERSITY OF LIVERPOOL

AN INVESTIGATION INTO THE AUTOMATION OF TIG (Tungsten Inert Gas) WELDING

Thesis submitted in accordance with the requirements of the University of Liverpool for the degree of Doctor in Philosophy by EDWARD MORRIS

Department of Electrical Engineering and Electronics

September, 1984.

A CKNOWLEDGEMENTS

I would like to thank Dr. J. Lucas for his help and guidance as supervisor of this research work. I also wish to express my gratitude to Professor J.H. Leck, Head of the Department of Electrical Engineering and Electronics, and to all the other department members for providing the facilities needed and for their assistance.

The work has been supervised financially by The Welding Institute and the Science and Engineering Research Council under the CASE award scheme. I am greatly indebted to Dr. W. Lucas of The Welding Institute, who acted as Industrial Supervisor, for the equipment and welding expertise he has provided.

CONTENTS

ABSTRACT

The TIM robot has been designed for use with high precision welding using the TIG (tungsten-inert-gas) process. Details are given of the design and construction of a five degrees of freedom machine, having cartesian and wrist movements. Details are also given of the operating features and the microcomputer and electronic hardware necessary to achieve the design. An Intel 8085 8-bit microcomputer was used to control the welding operation. The workpiece position and movement and the time sequence of the welding cycle could be controlled in accordance to a programmed set of parameters stored in memory.

The approach adopted for software development is also described. Emphasis is placed on the use of a high level language to achieve flexibility, with subroutines in assembly language for faster response. Path entry was carried out by moving the torch around the joint and recording by the point to point technique. Complex joint paths, for example, circular, square and triangular paths can be accurately reproduced and such welds have been used as illustrations of the flexibility of the machine.

CHAPTER 1

The subject of this research work is the automation of the Tungsten-Inert-Gas (TIG) welding process. This is a continuation of prelimary work,[1] at Liverpool University, relating to the control of the TIG process. In order to highlight the features which need to be considered, Chapter 1 gives basic background information on the TIG and similar arc welding processes and describes the fundamental features of robots which may be used.

1.1 ARC WELDING

1.1.1 General Arc Welding Considerations

Arc welding is fusion process for joining metals[2], where the heat needed for melting is supplied by an electrical arc. A pool of molten metal is formed under the arc and extends down into the workpiece, as shown in figure 1.1. It is usual to surround the weld pool with an 'inert' gas to prevent oxidation. If the metal thickness is not excessive (say less than 5 mm) then the joint may be achieved by simply melting the workpiece along the join, thus forming a weld seam.

For thicker metal the obtainable penetration may be less than the thickness. In this case it is usual to carry out the weld in several stages (passes) as shown in figure 1.2. The weld preparation, figure 1.2(a), allows the first pass, figure 1.2(b), to fully penetrate. Subsequent passes, figures 1.2(c) and 1.2(d), require metal to be fed into the weld pool as the weld progresses, in order to fill-in the joint. The filler metal is usually a similar material to that of the workpiece and takes the form of a wire or

rod. There are many variations of the multipass technique, but the above illustrates the basic procedure for welding thicker materials.

The heat input to the weld pool is an important factor, since it has a major influence on the resulting weld. A higher heat input rate gives deeper penetration and a smaller weld pool, and the subsequent cooling rate will be lower. Too low a cooling rate gives rise to a coarse metallurgical grain structure, and hence a less ductile seam. Too high a cooling rate may give rise to cracking. The heat input rate is governed by the setting of the welding power source and the rate at which the arc is traversed along the weld.

1.1.2 The Metal-Inert-Gas (MIG) Process

This process[3] is illustrated in figure 1.3(a). An arc is struck between a metal electrode and the workpiece. Usually the electrode is in the form of a continuous wire, of material similar to that of the workpiece. The heat of the arc causes the electrode to melt and the molten metal to transfer across to the weld pool. At the same time the electrode is fed towards the weld. Under normal conditions the rate of melting (burn-off) and feeding balance. A shielding gas, usually carbon dioxide, is supplied through the torch. Figure 1.4(a) shows the V/I curve for a typical MIG arc. Figure 1.5(a) shows a MIG power source's V/I curve, with long and short arc characteristics superimposed. Regarding figure 1.5(a), when the wire feed rate is increased the arc length will initially decrease. This shorter arc length causes an increase in current and hence an increase in burn-off rate, tending to lengthen the arc. In this manner the arc length adjusts itself to balance feed and burn-off rates.

The manner in which metal transfers across the arc is an important factor, since it determines the quality of the resulting weld. Three modes of transfer can be characterised, i.e. short-circuit, droplet and spray transfer.

In the short-circuited mode the operating current is relatively low. Large droplets of metal form on the tip of the electrode, eventually short-circuiting the electrode and workpiece. When this occurs the current rises rapidly, causing very rapid melting and the drop to detach.

At higher operating currents the droplet, although still relatively large, detaches before causing a short-circuit. This is the droplet transfer mode.

At higher currents still the droplets detach from the electrode before reaching an appreciable size and this is known as spray transfer.

An improved method, Synergic welding[4], allows the transfer mode to be controlled. Here instead of using a constant supply setting the current is pulsed between two levels, i.e. peak current (I_p) and background current (I_b) (where I_p is greater than I_b). The higher current pulse I_p is chosen so as to detach a metal droplet every time it occurs. Thus, by choosing I_p and I_b and their timings the metal transfer mode may be selected for a given feed rate (mean current).

The MIG process is widely used for general fabrication work, due mainly to its high level deposition rate, the low cost of the equipment used and the relative simplicity of the process. It is mainly used for welding steels and accounts for 23% of welding process[5].

1.1.3 The Tungsten-Inert-Gas (TIG) Process

This process[6] is illustrated in figure 1.3(b). An arc is struck between a tungsten electrode and the workpiece. The heat generated forms a weld pool, in the workpiece, but is not sufficient to melt the tungsten. It is usual for the polarity of the electrode to be negative, with respect to the workpiece, since about twice as much heat is generated in the anode region as in the cathode region. A shielding gas, usually argon, is supplied through the welding torch. If filler wire is required it is added externally.

Figure 1.4(a) compares a typical TIG arc V/I characteristic to that of an MIG arc. Figure 1.4(b) shows the variation in arc voltage with arc length. Figure 1.5(b) shows long and short arc characteristics superimposed onto the V/I curve of a TIG welding power source. It can be seen that the 'constant current' characteristic of the supply allows variation in the arc length without causing drastic changes in arc current and thus allows manual control of arc stability.

Pulsed TIG[1] is a development of the basic (D.C.) TIG process. Here the welding current is pulsed between two values I_p and I_b (see figure 1.6). I_b is chosen to maintain a stable arc but without causing any melting, whilst I_p is chosen to cause enough melting for the required penetration. The resulting seam takes the form of a series of overlapping beads where the bead size and degree of overlap is determined by the pulse parameters. This method allows greater control over weld penetration than does D.C. TIG welding.

The TIG process is used where precision, high integrity welds with good finish are required. It can be used on a wide variety of metals, e.g. steels, aluminium, copper, and is widely used in the nuclear and

aircraft industries. TIG welding accounts for 17% of welding process with a predicted usage of 20% by 1985[5].

1.1.4 The Plasma Arc Welding Process

This process[7] is illustrated in figure 1.3(c). In many respects this is similar to TIG welding. The arc is however constricted by an orifice in the torch, and an additional stream of inert gas is passed through this same orifice. The constriction and increased velocity of the arc plasma causes higher temperatures to be achieved, and thus gives increased heat input rates.

This process is used where very high heat input rates are required, e.g. for welding very thin metals and for 'flame cutting'.

1.2 ROBOTS FOR ARC WELDING

Before considering the suitability of particular robots for arc welding, let us review the general characteristics of robots.

1.2.1 Robot Configurations

For the purpose of arc welding a robot is a programmable device for manipulating a welding torch. Robots may be grouped into four categories, according to how this manipulation is achieved, i.e. XYZ, Cylindrical, Polar and Jointed-arm. Figure 1.7 illustrates these categories and shows the three basic movements (degrees-of-freedom), i.e. travel, elevation and reach, for each type. The welding torch is held in a 'wrist' attached to the end of the reach arm, and figure 1.8 shows the movements of a three degrees-of-freedom wrist.

The portion of space which the torch can reach is known as the

working envelope and depends not only on the size of the robot but the amount of movement allowed on each degree-of-freedom.

1.2.2 Drives

Two types of actuators, (i.e. hydraulic and electric) are used for welding robots. Hydraulic actuators have a much greater power to weight ratio than electric but cannot generally achieve the same positional accuracy. Also the equipment required (hydraulic pump, reservoir, cooler and plumbing) make hydraulics a less attractive proposition. Thus hydraulic robots are used mainly where greater loads (say above 50 kg) are being manipulated or where very high speed is required.

1.2.3 Programming and Path Control

Nearly all robots have a pendant type keyboard to allow the user to enter a position by manually directing the robot towards it. In this manner a path may be programmed as a succession of positions. When commanded to reproduce the path the robot moves from one point to the next in a straight line, at the required speed. This procedure is known as 'point-to-point motion with linear interpolation'. Under this scheme a curved path must be entered as many points at small intervals.

In order to alleviate this problem some robots (e.g. the Kawasaki-Unimate Puma 700) allow circular arcs to be defined as three points (i.e. the starting and ending points of the arc and some arbitrary intermediate point on the arc). When commanded the robot will produce the complete arc, smoothly, at the required speed. This is known as circular interpolation.

1.2.4 Requirements of the TIG Welding Robot

The dimensions of practical weld seams (down to less than 1 mm) and arc lengths (less than 2 mm) require the torch to be positioned with an accuracy greater than 0.1 mm. The Cylindrical, Polar and Jointed arm configurations suffer from the fact that accuracy varies with position (e.g. accuracy is less when arms are extended more), and it is difficult to achieve a required value throughout the working envelope. A given accuracy is much more readily obtained with an XYZ configuration.

TIG welding speeds are in the range 1 mm/s to 10 mm/s and need to be controlled with an accuracy of 1 % of the required value. These speeds are relatively low and easily accomplished, however higher speeds are useful to non-welding motions in order to reduce process time.

Welding torches, with associated cable drag, are normally less than 1 kg in weight and the robot should be capable of handling this load throughout its working envelope.

The control of path trajectory is an important feature of automated welding systems. Speed has to be maintained along the path and provision made for the programming of arbitrary three dimensional space curves.

1.2.5 Practical Welding Robots

Table 1.1 summarises the characteristics of the more notable robots which have been applied to arc welding.[8] Although all have been used for MIG welding the accuracy of the systems does not allow them to be used in all classes of TIG welding.

1.2.6 Project Aim

The project described in the following chapters was set up in

order to research the problems associated with the automation of the TIG welding process, to a degree where it can be used on very small job batches (say a single job). The construction of a practical system was specified in order to assess ideas and to provide a test-bed for further work. The system produced combines the welding current control methods of Sloan [1] with an XYZ robotic manipulator.

Sloan showed how a microprocessor may be interfaced to TIG welding equipment, in order to control the welding process, and his system is outlined in figure 1.9. The controller is based on an Intel 8080 microprocessor with 8 k of RAM and 4 k of EPROM. A keyboard and monitor are provided to facilitate user interaction. A D.C. TIG welding power supply and a simple linear traverse unit are interfaced to the controller and allow current modulation and torch motion to be achieved. The operator may recall sets of welding parameters from memory, display them and if necessary make modifications. A welding sequence may then be initiated. Figure 1.10 shows the variable parameters associated with this sequence.

Figure 1.11 depicts the system resulting from the present project. The controller is based on an Intel 8085 microprocessor with 32 k of RAM and 20 k of EPROM. A VDU allows the user to enter welding parameters and system commands. A D.C. TIG welding power supply provides the welding current and an XYZ robot allows torch manipulation. A pendant keyboard allows the programming of weld paths. Welds are executed at constant speed with point-to-point linear interpolation path control.

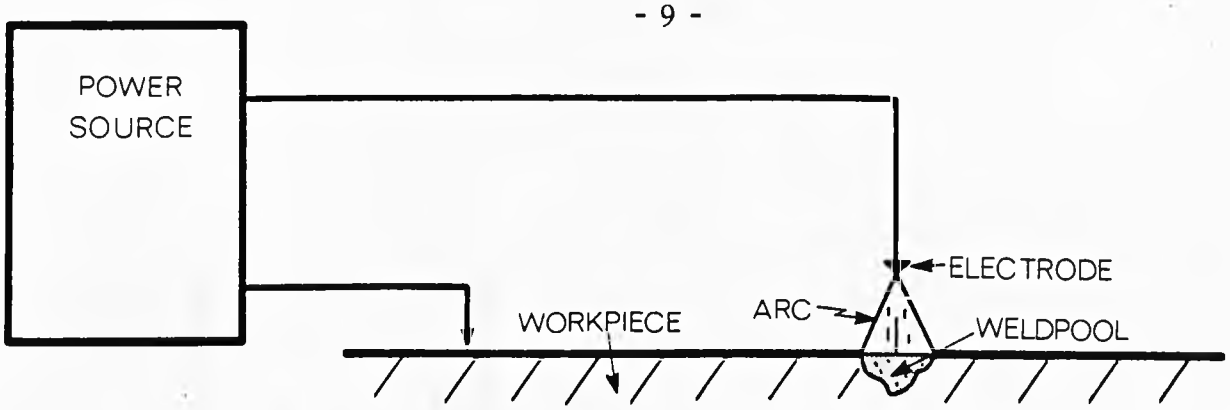


Figure 1.1 WELD-POOL PENERTRATION

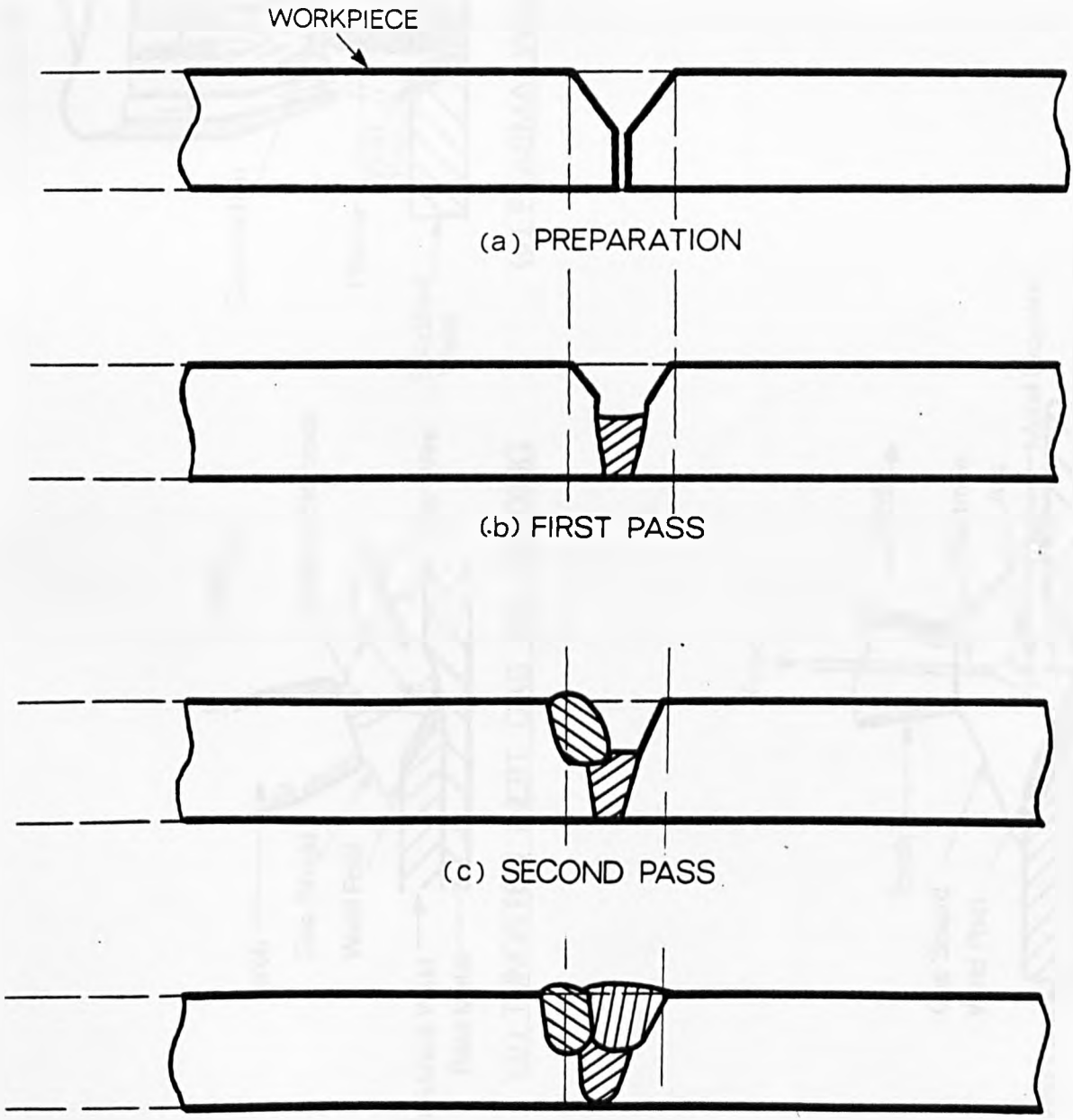
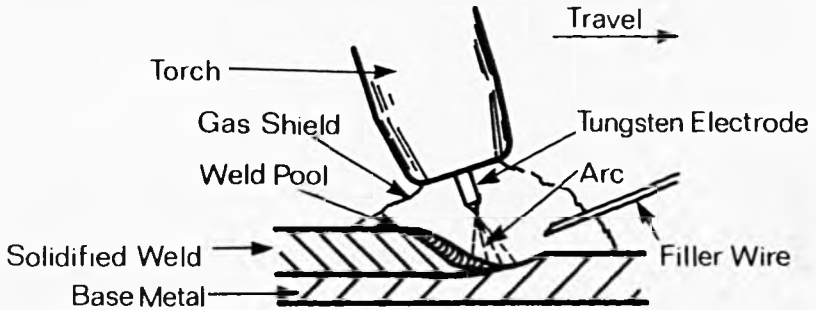
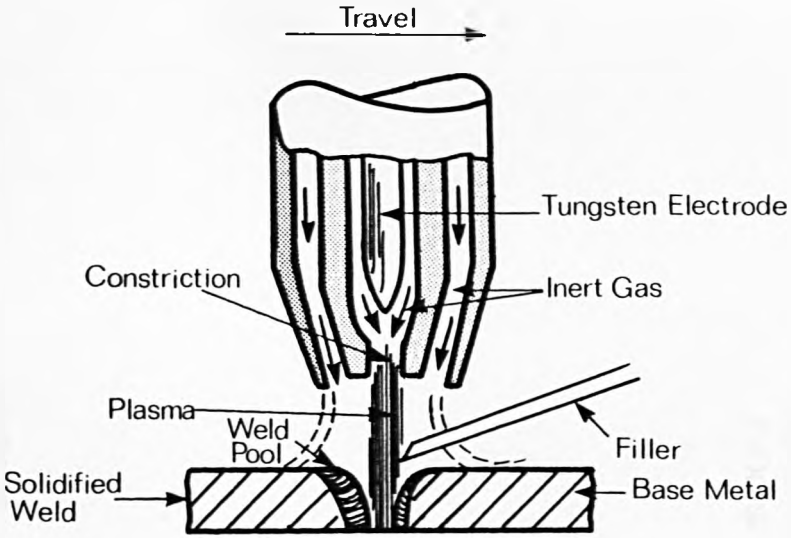


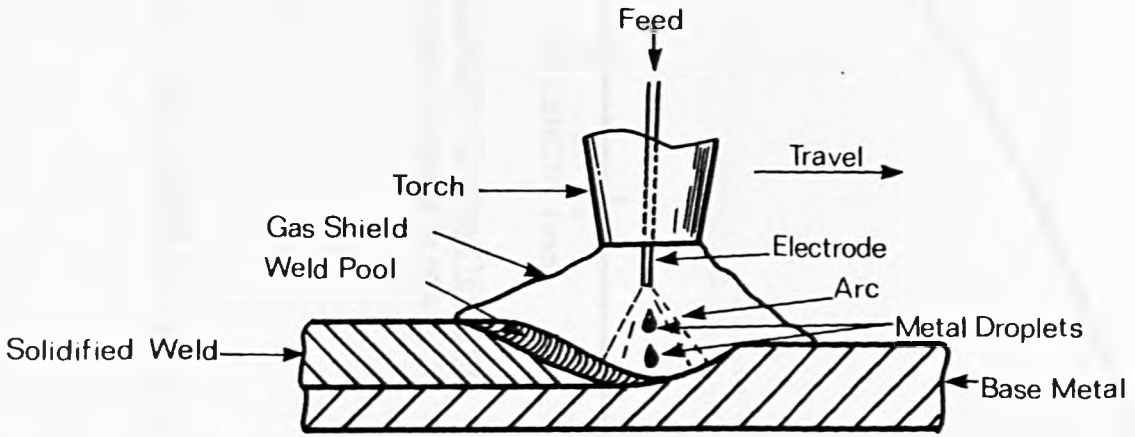
Figure 1.2 WELDING OF 'THICKER' METAL



(b) TUNGSTEN INERT GAS TIG WELDING

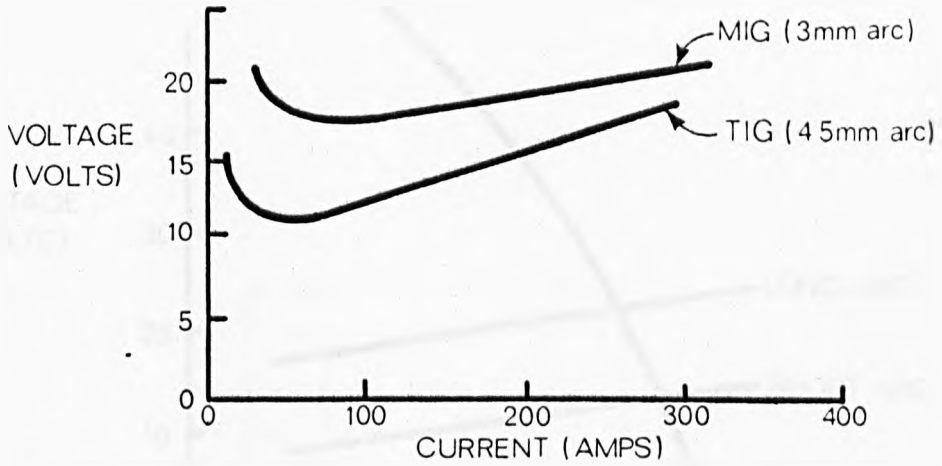


(c) PLASMA WELDING

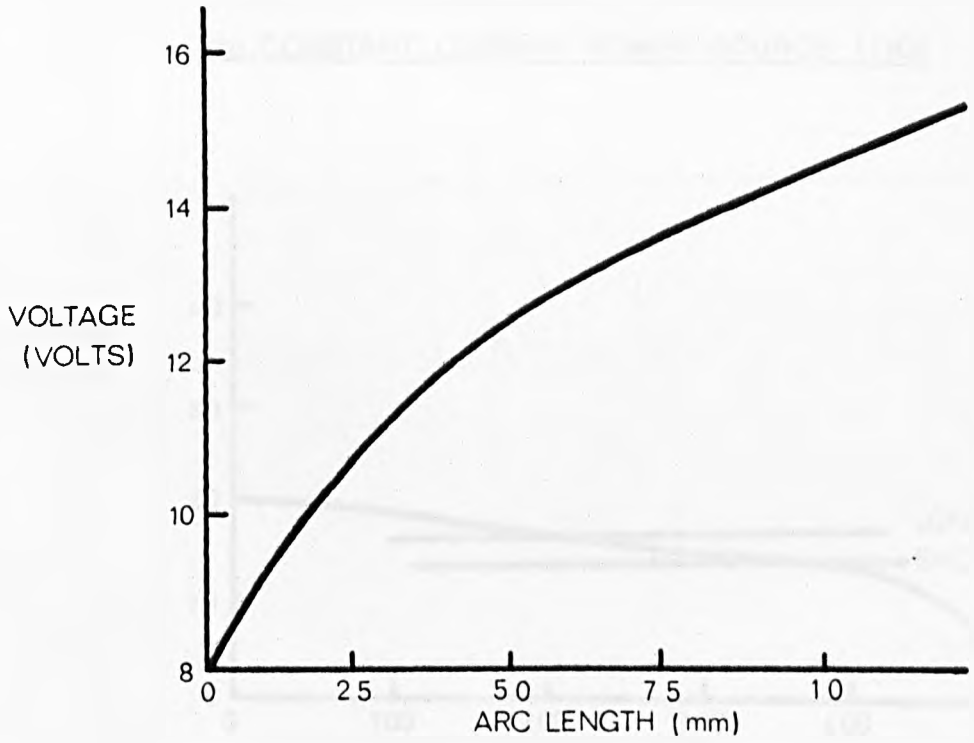


(a) METAL INERT GAS MIG WELDING

Figure 1.3 ARC WELDING PROCESSES

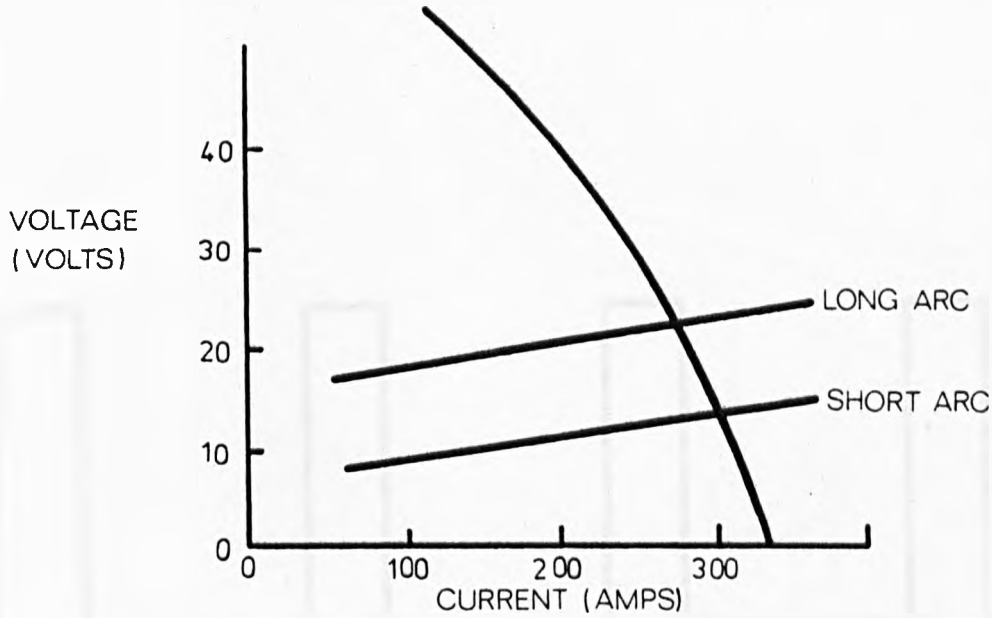


(a) TYPICAL WELDING ARC V/I CHARACTERISTICS

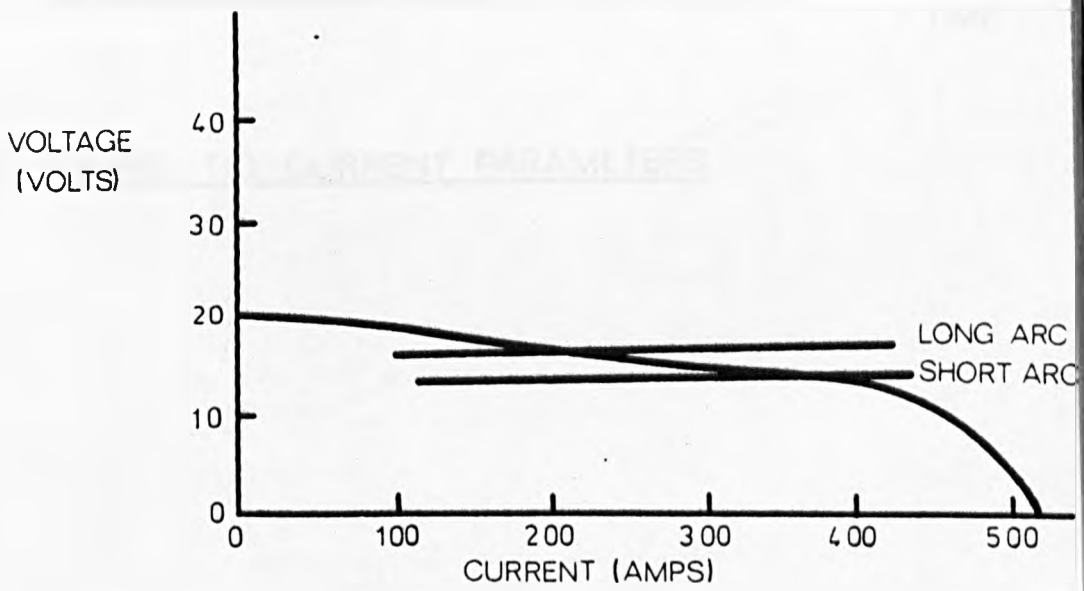


(b) ARC VOLTAGE / LENGTH CHARACTERISTIC
(thoriated tungsten electrode at 50A)

figure 1.4 ARC CHARACTERISTICS



(b) CONSTANT CURRENT POWER SOURCE (TIG)



(a) CONSTANT VOLTAGE POWER SOURCE (MIG)

figure 1.5 POWER SORCE V/I CHARACTERISTICS

figure 1.6 PULSED TIG CURRENT PARAMETERS

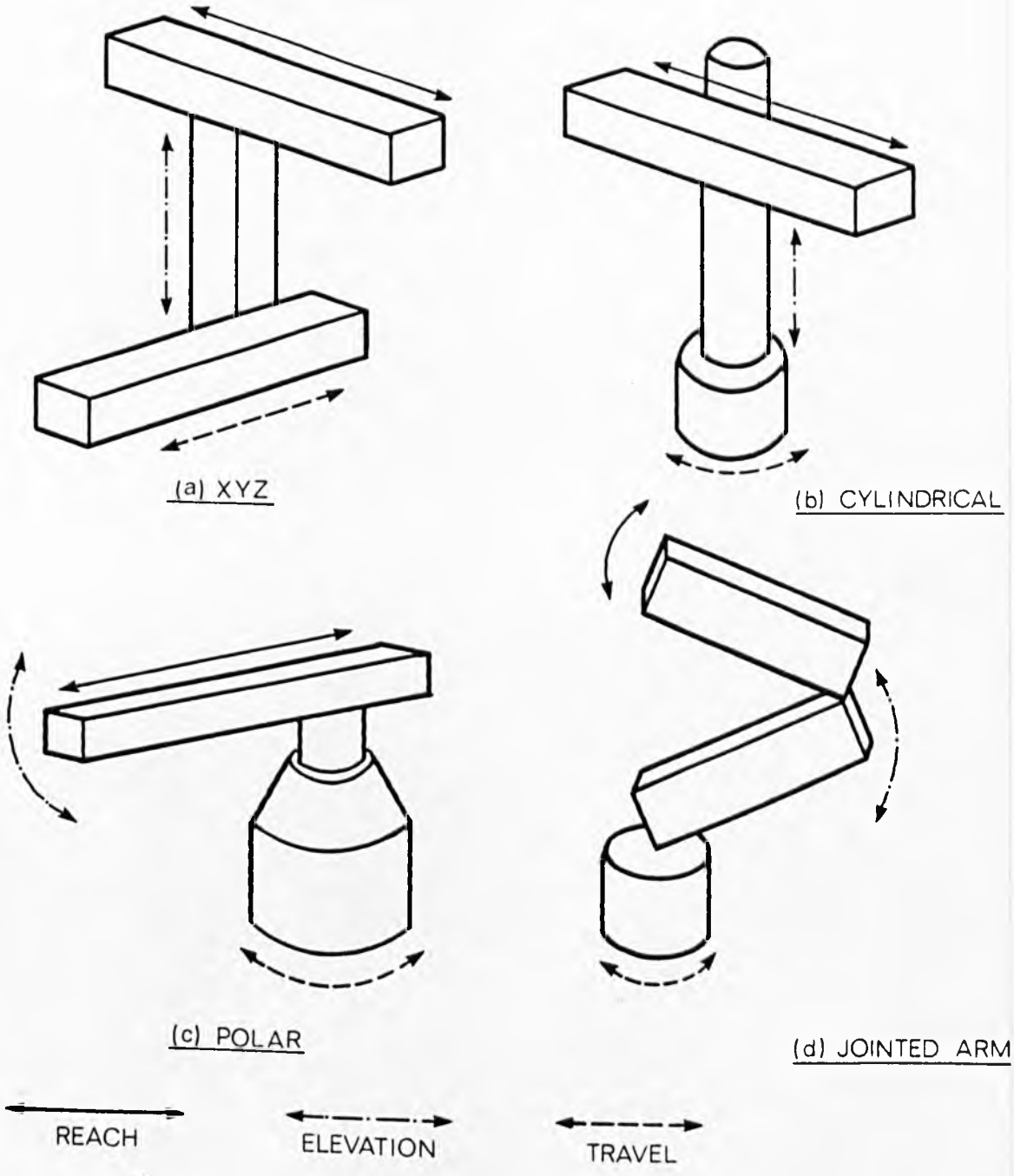


figure 1.7 ROBOT CONFIGURATIONS

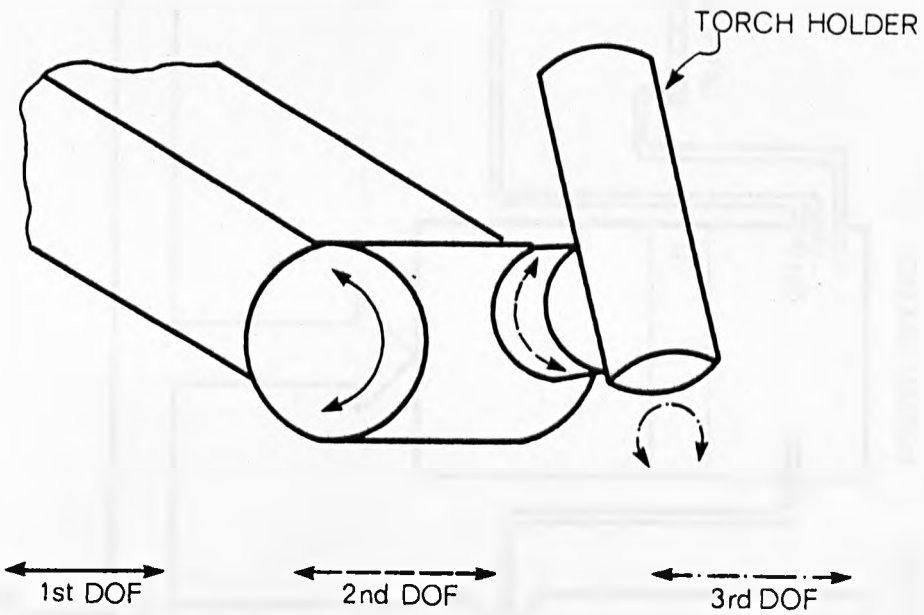


figure 1.8 A THREE DEGREES OF FREEDOM (DOF) WRIST

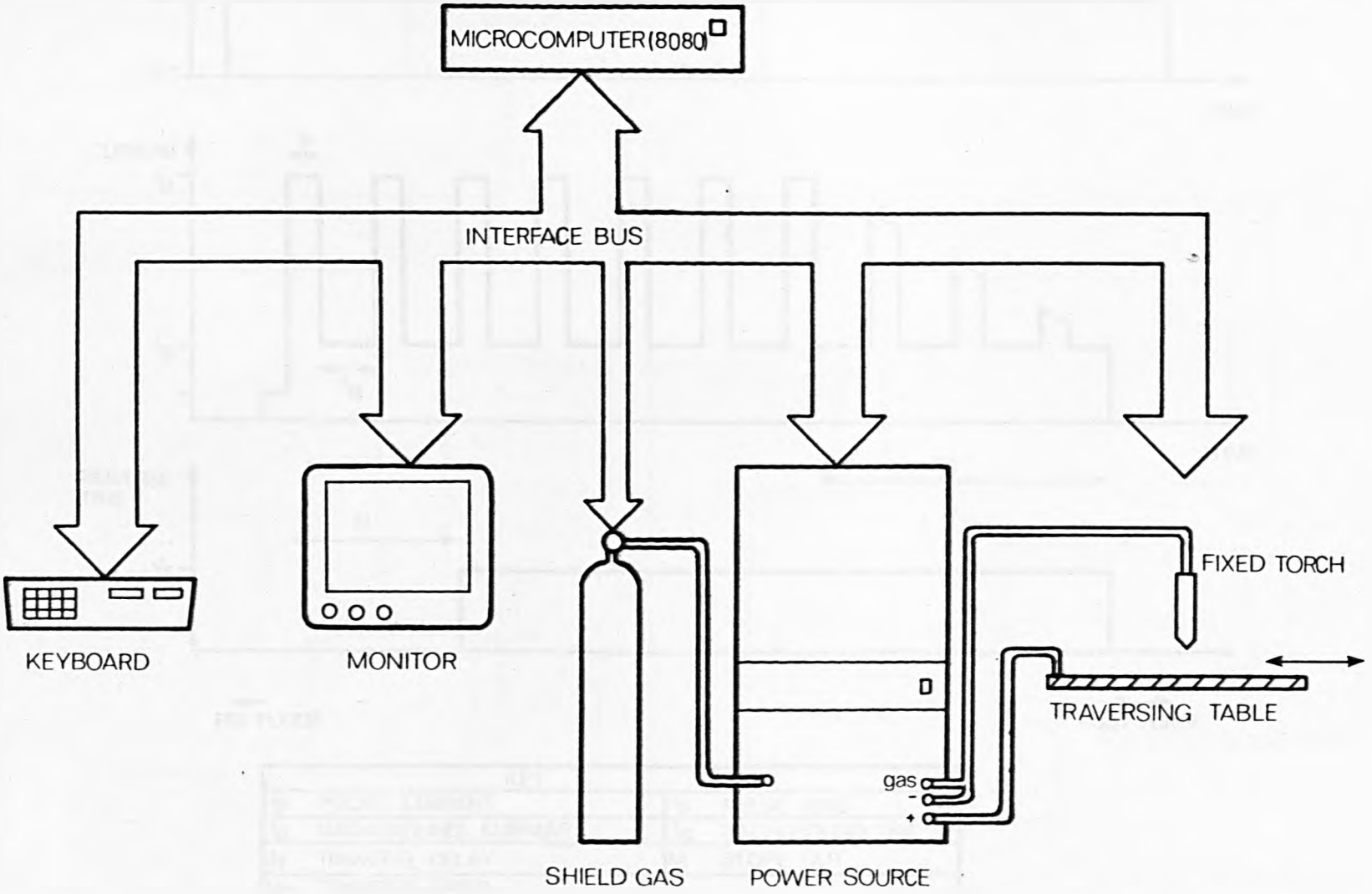
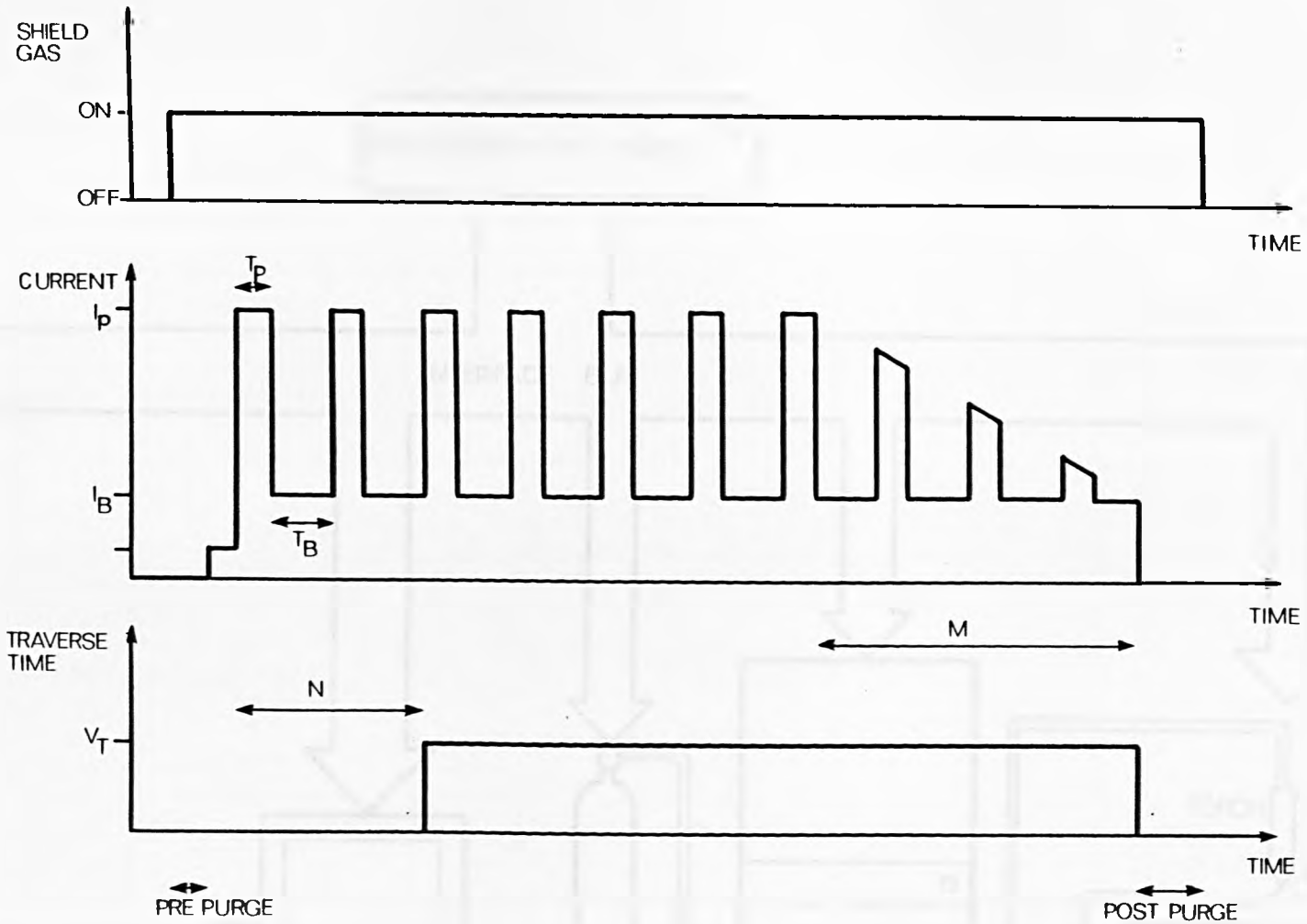


figure 1.9 A TIG PROCESS CONTROL SYSTEM



KEY			
I_P	PULSE CURRENT	T_P	PULSE TIME
I_B	BACKGROUND CURRENT	T_B	BACKGROUND TIME
N	TRAVERSE DELAY	M	SLOPE OUT
V_T	TRAVERSE SPEED		

figure 1.10 PULSED TIG PARAMETERS

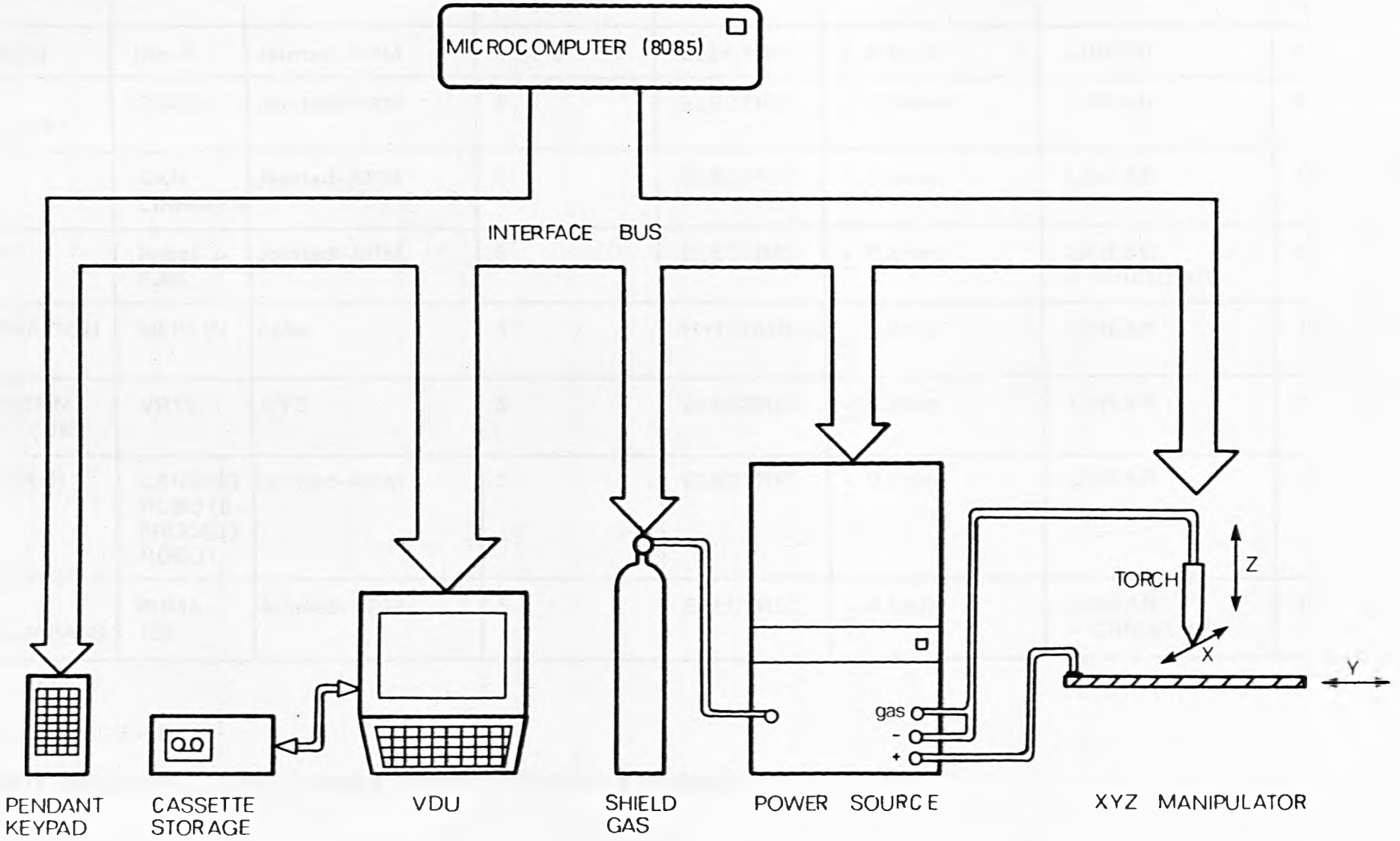


figure 1.11 A TIG WELDING ROBOTIC SYSTEM

MANUFACTURER	MODEL	CONFIGURATION	DEGREES OF FREEDOM	DRIVE	REPEATABILITY	INTERPOLATION	LOAD (kg)	SPEED (mm/s)
ASEA (SWEDEN)	IRb-6	Jointed-ARM	5 or 6	ELECTRIC	± 0.2mm	LINEAR	6	750
CINCINNATI MILACRON (USA)	T3-726	Jointed-ARM	6	ELECTRIC	± 0.15mm	LINEAR	6	1000
YASKAWA (JAPAN)	GKN LINKMAN	Jointed-ARM	5	ELECTRIC	± 0.2mm	LINEAR	10	800
SHIN MEIWA (JAPAN)	Robel J-RJ65	Jointed-ARM	5	ELECTRIC	± 0.2mm	LINEAR + CIRCULAR	10	*
HALL AUTOMATION (UK)	MERLIN	Polar	5	HYDRAULIC	± 1.0mm	LINEAR	13	150
HEAD WRIGHTON MACHINE Co. (UK)	VR15	XYZ	5	ELECTRIC	± 0.2mm	LINEAR	**	400
HITACHI (JAPAN)	LANSING ROBOTS-PROCESS ROBOT	Jointed-ARM	5	ELECTRIC	± 0.2mm	LINEAR	10	1000
KAWASAKI-UNIMATION (JAPAN)	PUMA-750	Jointed-ARM	5	ELECTRIC	± 0.1mm	LINEAR + CIRCULAR	10	1000

NOTES:- *NOT SPECIFIED

**NOT APPLICABLE, UNIT COMPLETE WITH WELDING EQUIPMENT

TABLE 1.1 REVIEW OF ARC WELDING ROBOTS USING THE MIG PROCESS

CHAPTER 2

DESIGN ASPECTS OF A T.I.G. WELDING SYSTEM

2.1 KINEMATICS

A manipulator is a device consisting of one or more actuated joints, connected via links. The links serve to maintain a fixed relationship between one joint and the next. The object to be manipulated is attached to the final link. In order to control and use manipulators we need to understand their kinematics, i.e. the relationships between the various actuator positions and the corresponding object position.

Consider the case of a welding torch manipulator consisting of a cartesian manipulator and 'wrist' manipulator combined. This is shown, in a simplified form, in figure 2.1. The frame (X,Y,Z) is the world co-ordinate frame and it is usual to express the position of objects with respect to this. The frame (X*, Y*, Z*) has been assigned to the wrist manipulator and is referred to as the torch co-ordinate frame.

Using the notation of Paul[9] we can formulate the relationship between the world and torch co-ordinate frames :-

$$\begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} = \text{Trans}(K_x, K_y, K_z)\ \text{Rot}(X,\theta)\ \text{Rot}(Y,\phi) \begin{bmatrix} X^* \\ Y^* \\ X^* \\ 1 \end{bmatrix} \qquad (2.1)$$

The translation term $\text{Trans}(K_x, K_y, K_z)$ is due to the cartesian manipulator. The rotation terms $\text{Rot}(X,\theta)$ and $\text{Rot}(Y,\phi)$ are due to the rotational motions of the wrist manipulator about the X* and Y* axes respectively.

Re-writing (2.1) in full matrix form :-

$$\begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & K_x \\ 0 & 1 & 0 & K_y \\ 0 & 0 & 1 & K_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \text{Cos}\theta & -\text{Sin}\theta & 0 \\ 0 & \text{Sin}\theta & \text{Cos}\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \text{Cos}\phi & 0 & \text{Sin}\phi & 0 \\ 0 & 1 & 0 & 0 \\ -\text{Sin}\phi & 0 & \text{Cos}\phi & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X^* \\ Y^* \\ Z^* \\ 1 \end{bmatrix}$$

$$= \begin{bmatrix} 1 & 0 & 0 & K_x \\ 0 & 1 & 0 & K_y \\ 0 & 0 & 1 & K_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \text{Cos}\phi & 0 & \text{Sin}\phi & 0 \\ \text{Sin}\theta\,\text{Sin}\phi & \text{Cos}\theta & -\text{Sin}\theta\,\text{Cos}\phi & 0 \\ -\text{Cos}\theta\,\text{Sin}\phi & \text{Sin}\theta & \text{Cos}\theta\,\text{Cos}\phi & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X^* \\ Y^* \\ Z^* \\ 1 \end{bmatrix}$$

$$= \begin{bmatrix} \text{Cos}\phi & 0 & \text{Sin}\phi & K_x \\ \text{Sin}\theta\,\text{Sin}\phi & \text{Cos}\theta & -\text{Sin}\theta\,\text{Cos}\phi & K_y \\ -\text{Cos}\theta\,\text{Sin}\phi & \text{Sin}\theta & \text{Cos}\theta\,\text{Cos}\phi & K_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X^* \\ Y^* \\ Z^* \\ 1 \end{bmatrix}$$

$$= \begin{bmatrix} X^*\text{Cos}\phi + Z^*\text{Sin}\phi + K_x \\ X^*\text{Sin}\theta\text{Sin}\phi + Y^*\text{Cos}\theta - Z^*\text{Sin}\theta\text{Cos}\phi + K_y \\ -X^*\text{Cos}\theta\text{Sin}\phi + Y^*\text{Sin}\theta + Z^*\text{Cos}\theta\text{Cos}\phi + K_z \\ 1 \end{bmatrix} \qquad (2.2)$$

Thus if we measure any point in the torch co-ordinate frame we can determine the world co-ordinates of the same point by substituting in (2.2).

Now the position of the torch tip in the torch frame is given by $\text{TORCH}_{tip}(0,\ h, -l)$ and in the world co-ordinate frame as $\text{WORLD}_{tip}(x,y,z)$.

Where :-

$$\begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} -l\,Sin\phi + K_x \\ h\,Cos\theta + l\,Sin\theta\,Cos\phi + K_y \\ h\,Sin\theta - l\,Cos\theta\,Cos\phi + K_z \\ 1 \end{bmatrix} \qquad (2.3)$$

K_x, K_y, K_z represent the settings of the X, Y and Z axis actuators of the cartesian manipulator and θ and ϕ represent the position angles of the wrist manipulator.

Hence (2.3) expresses the position of the torch tip for any given actuator settings.

2.2 PATH REPRESENTATION AND MANIPULATION

In automating the welding process we are concerned with the representation of three dimensional space curves, and with the manipulation of these curves to account for differing manipulator-workpiece orientations and workpiece dimensional tolerance.

2.2.1 Linear Interpolation

Consider figure 2.2. This shows a three dimensional space curve between the points $P_0(X_0, Y_0, Z_0)$ and $P_n(X_n, Y_n, Z_n)$ which we require to represent mathematically. Linear interpolation provides what is possibly the simplest scheme. Here we choose intermediate points $P_1, \ldots, P_{n-1}$ and represent the curve by straight line segments $P_0P_1, \ldots, P_{n-1}P_n$.

Consider the segment P_1P_2 shown in detail in figure 2.3.

$$\text{Let } X_2 - X_1 = \Delta X$$

$$Y_2 - Y_1 = \Delta Y$$

$$Z_2 - Z_1 = \Delta Z$$

Now $\Delta S = (\Delta X^2 + \Delta Y^2 + \Delta Z^2)^{\frac{1}{2}}$

If the welding speed is v ms^{-1} then the time interval Δt from P_1 to P_2 is given by :-

$$\Delta t = \frac{\Delta S}{v}$$

$$\text{Now } \dot{x} = \frac{\Delta X}{\Delta t} = \frac{\Delta X\, v}{\Delta S} \tag{2.4}$$

$$\dot{y} = \frac{\Delta Y}{\Delta t} \quad \frac{\Delta Y\, v}{\Delta S} \tag{2.5}$$

$$\dot{z} = \frac{\Delta Z}{\Delta t} \quad \frac{\Delta Z\, v}{\Delta S} \tag{2.6}$$

(2.4), (2.5) and (2.6) give the mathematical relationship, between position and time, required to trace the space curve. Although these relationships are simple there is a major disadvantage in the fact that many intermediate points ($P_1 .. P_{n-1}$) are needed to represent, with sufficient accuracy, practical weld seams.

2.2.2 Cubic Spline Interpolation

We wish to represent a function f(u) as a continuous mathematical equation by inserting a few values at non-equally spaced positions u_j (j = 0...n). Such a curve is shown in figure 2.4. The function z = f(u) may be approximated, piecewise, over the intervals h_j and h_{j+1} by $\emptyset_j(u)$ and $\emptyset_{j+1}(u)$ respectively, where $\emptyset(u)$ is a cubic equation.

For a smooth, continuous curve the following conditions apply :-

$$\phi_j(u_j) = \phi_{j+1}(u_j)$$

$$\phi'_j(u_j) = \phi'_{j+1}(u_j) \qquad \text{(equating first derivatives)}$$

$$\phi''_j(u_j) = \phi''_{j+1}(u_j) \qquad \text{(equating second derivatives)}$$

Let $S_j = \phi''(u_j)$

Since $\phi_j(u)$ is cubic then $\phi''_j(u)$ will be linear, hence over the interval h_j :-

$$\phi''_j(u) = S_{j-1} + (u - u_{j-1}) \frac{(S_j - S_{j-1})}{(u_j - u_{j-1})}$$

$$= \frac{S_{j-1}(u_j - u_{j-1}) + (u - u_{j-1})(S_j - S_{j-1})}{u_j - u_{j-1}}$$

$$= \frac{S_{j-1}(u_j - u) + S_j(u - u_{j-1})}{h_j}$$

Integrating gives :-

$$\phi'_j(u) = \frac{S_{j-1}(u_j - u)^2 + S_j(u - u_{j-1})^2}{2h_j} + C_1$$

Integrating again gives :-

$$\phi_j(u) = \frac{S_{j-1}(u_j - u)^3 + S_j(u - u_{j-1})^3}{6h_j} + C_1u + C_2 \qquad (2.7)$$

Substituting $\phi_j(u_j) = Z_j$ into (2.7) gives :-

$$Z_j = \frac{S_j(u_j - u_{j-1})^3}{6h_j} + C_1u_j + C_2$$

$$= \frac{S_j h_j^2}{6} + C_1u_j + C_2 \qquad (2.8)$$

Substituting $\emptyset_j(u_{j-1}) = Z_{j-1}$ into (2.7) gives :-

$$Z_{j-1} = \frac{S_{j-1}(u_j - u_{j-1})^3}{6h_j} + C_1 u_{j-1} + C_2$$

$$= \frac{S_{j-1} h_j^2}{6} + C_1 u_{j-1} + C_2 \qquad (2.9)$$

(2.8) - (2.9) gives :-

$$Z_j - Z_{j-1} = \frac{h_j^2}{6}(S_j - S_{j-1}) + C_1(u_j - u_{j-1})$$

$$= \frac{h_j^2}{6}(S_j - S_{j-1}) + C_1 h_j$$

thus $C_1 = \frac{(Z_j - Z_{j-1})}{h_j} + \frac{h_j(S_{j-1} - S_j)}{6}$ (2.10)

(2.8) - $\frac{u_j}{u_{j-1}}$ (2.9) gives :-

$$Z_j - \frac{u_j Z_{j-1}}{u_{j-1}} = \frac{h_j^2}{6}\left(S_j - \frac{u_j S_{j-1}}{u_{j-1}}\right) + C_1\left(u_j - \frac{u_j u_{j-1}}{u_{j-1}}\right) + C_2\left(1 - \frac{u_j}{u_{j-1}}\right)$$

$$= \frac{h_j^2}{6}\left(S_j - \frac{u_j S_{j-1}}{u_{j-1}}\right) + C_2\left(\frac{u_{j-1} - u_j}{u_{j-1}}\right)$$

$$= \frac{h_j^2}{6}\left(S_j - \frac{u_j S_{j-1}}{u_{j-1}}\right) - \frac{C_2 h_j}{u_{j-1}}$$

Thus $C_2 = \frac{h_j}{6}(S_j u_{j-1} - u_j S_{j-1}) + \frac{u_j Z_{j-1} - Z_j u_{j-1}}{h_j}$ (2.11)

Substituting (2.10) and (2.11) into (2.7) gives :-

$$\phi_j(u) = \frac{S_{j-1}(u_j - u)^3}{6h_j} + S_j\frac{(u - u_{j-1})^3}{6h_j}$$

$$+ \frac{(Z_j - Z_{j-1})u}{h_j} + h_j\frac{(S_{j-1} - S_j)\,u}{6}$$

$$+ h_j\frac{(S_j u_{j-1} - u_j S_{j-1})}{6} + \frac{u_j Z_{j-1} - Z_j u_{j-1}}{h_j}$$

Thus :-

$$\phi_j(u) = \frac{S_{j-1}(u_j - u)^3}{6h_j} + \frac{S_j(u - u_{j-1})^3}{6h_j}$$

$$+ \left(\frac{Z_{j-1}}{h_j} - \frac{h_j S_{j-1}}{6}\right)(u_j - u)$$

$$+ \left(\frac{Z_j}{h_j} - \frac{h_j S_j}{6}\right)(u - u_{j-1}) \qquad (2.12)$$

(2.12) is the interpolating cubic, over the span h_j, in terms of the values S_{j-1} and S_j. Thus in order to form our approximation we must solve for S_{j-1} and S_j.

Differentiating (2.12) gives :-

$$\phi'_j(u) = \frac{-S_{j-1}(u_j - u)^2}{2h_j} + \frac{S_j(u - u_{j-1})^2}{2h_j} - \frac{Z_{j-1}}{h_j} + \frac{h_j S_{j-1}}{6} + \frac{Z_j}{h_j} - \frac{h_j S_j}{6}$$

at $u = u_j$:-

$$\phi'_j(u_j) = \frac{S_j(u_j - u_{j-1})^2}{2h_j} - \frac{Z_{j-1}}{h_j} + \frac{h_j S_{j-1}}{6} + \frac{Z_j}{h_j} - \frac{h_j S_j}{6}$$

$$= \frac{S_j h_j}{2} + \frac{Z_j - Z_{j-1}}{h_j} + \frac{h_j S_{j-1}}{6} - \frac{h_j S_j}{6}$$

$$= \frac{Z_j - Z_{j-1}}{h_j} + \frac{h_j S_{j-1}}{6} + \frac{S_j h_j}{3} \qquad (2.13)$$

From (2.12) the interpolating equation $\phi_{j+1}(u)$ for the interval h_{j+1} may be written :-

$$\phi_{j+1}(u) = \frac{S_j(u_{j+1} - u)^3}{6h_{j+1}} + \frac{S_{j+1}(u - u_j)^3}{6h_{j+1}}$$

$$+ \left(\frac{Z_j}{h_{j+1}} - \frac{h_{j+1} S_j}{6}\right)(u_{j+1} - u)$$

$$+ \left(\frac{Z_{j+1}}{h_{j+1}} - \frac{h_{j+1} S_{j+1}}{6}\right)(u - u_j)$$

Differentiating gives :-

$$\phi'_{j+1}(u) = -\frac{S_j(u_{j+1} - u)^2}{2h_{j+1}} + \frac{S_{j+1}(u - u_j)^2}{2h_{j+1}}$$

$$\frac{-Z_j}{h_{j+1}} + \frac{h_{j+1} S_j}{6} + \frac{Z_{j+1}}{h_{j+1}} - \frac{h_{j+1} S_{j+1}}{6}$$

At $u = u_j$:-

$$\phi'_{j+1}(u_j) = -\frac{S_j(u_{j+1} - u_j)^2}{2h_{j+1}} + \frac{Z_{j+1} - Z_j}{h_{j+1}} + \frac{h_{j+1}S_j}{6} - \frac{h_{j+1}S_{j+1}}{6}$$

$$= -\frac{S_j h_{j+1}}{2} + \frac{Z_{j+1} - Z_j}{h_{j+1}} + \frac{h_{j+1}S_j}{6} - \frac{h_{j+1}S_{j+1}}{6}$$

$$= \frac{Z_{j+1} - Z_j}{h_{j+1}} - \frac{S_j h_{j+1}}{3} - \frac{h_{j+1}S_{j+1}}{6} \qquad (2.14)$$

Equating (2.13) and (2.14) :-

$$\frac{Z_j - Z_{j-1}}{h_j} + \frac{h_j S_{j-1}}{6} + \frac{S_j h_j}{3} = \frac{Z_{j+1} - Z_j}{h_{j+1}} - \frac{S_j h_{j+1}}{3} - \frac{h_{j+1}S_{j+1}}{6}$$

Thus :-

$$h_j S_{j-1} + 2(h_j + h_{j+1})S_j + h_{j+1}S_{j+1} = \frac{6(Z_{j+1} - Z_j)}{h_{j+1}} + \frac{6(Z_{j-1} - Z_j)}{h_j} \qquad (2.15)$$

If we have $n+1$ points ($j = 0..n$) then equations such as (2.15) may be written for the $n-1$ internal points ($j = 1...n-1$). This gives $n-1$ equations in $n+1$ unknowns ($S_0...S_n$). In order to obtain solutions for $S_0...S_n$ we need to formulate another two equations. This is achieved by choosing boundary conditions, i.e. specifying properties of the curve at, and beyond, the end points. Three possible boundary conditions are Periodic-Spline, Natural-Spline and Built-in-Ends-Spline, as described below.

Periodic Spline

In this case it is assumed that the curve repeats beyond j = n, i.e. :-

$$S_{-1} = S_n \qquad S_{n+1} = S_0$$

$$Z_{-1} = Z_n \qquad Z_{n+1} = Z_0$$

$$h_0 = h_n \qquad h_{n+1} = h_1$$

Hence the two additional equations given by (2.15) are :-

$$\text{for } j = 0,\quad h_n S_n + 2(h_n + h_1) S_0 + h_1 S_1 + h_1 S_1 = \frac{6(Z_1 - Z_0)}{h_1} + \frac{6(Z_n - Z_0)}{h_n}$$

$$\text{for } j = n,\quad h_n S_{n-1} + 2(h_n + h_1) S_n + h_1 S_0 = \frac{6(Z_0 - Z_n)}{h_1} + \frac{6(Z_{n-1} - Z_n)}{h_n}$$

Figure 2.5 shows the n + 1 equations written in matrix form.

Natural Spline

Here the curve continues beyond the end points as straight lines, with no curvature at u_0 and u_n (i.e. $S_0 = S_n = 0$). Thus we do not have to solve for S_0 and S_n, and now have n - 1 equations in n - 1 unknowns ($S_1 \ldots S_{n-1}$). As shown in figure 2.6.

Built-in-Ends Spline

In this case the curve continues beyond the end points as straight lines but with fixed gradients.

i.e. :- $\quad \phi'(u_0) = g_0$

$$\phi'(u_n) = g_n$$

From (2.14) :- $\quad g_0 = \frac{Z_1 - Z_0}{h_1} - \frac{S_0 h_1}{3} - \frac{h_1 S_1}{6}$

hence :- $\quad 2h_1 S_0 + h_1 S_1 = \frac{6(Z_1 - Z_0)}{h_1} - 6g_0$

From (2.13) :- $$g_n = \frac{Z_n - Z_{n-1}}{h_n} + \frac{h_n S_{n-1}}{6} + \frac{S_n h_n}{3}$$

hence :- $$2h_n S_n + h_n S_{n-1} = \frac{6(Z_{n-1} - Z_n)}{h_n} + 6g_n$$

Thus we now have $n+1$ equations in $n+1$ unknowns ($S_0 .. S_n$).

Figure 2.7 shows these equations in matrix form.

Having chosen appropriate boundary conditions we need to solve the resulting matrix equation for $S_0 .. S_n$. This can be done using Gauss-Jordan elimination.

Supposing we have the equation :-

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{bmatrix} \begin{bmatrix} s_0 \\ s_1 \\ s_2 \\ s_3 \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \end{bmatrix}$$

or in short-hand notation :- $A\,S = B$

If we find a set of operations which act on A to give the identity matrix I and apply corresponding operations to B to give B' then the Result is :- $I\,S = B'$

i.e. B' is the solution matrix for S.

Starting at the top row of A we divide by a_{11} giving :-

$$\begin{bmatrix} 1 & x_1 & x_2 & x_3 \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{bmatrix} \begin{bmatrix} s_0 \\ s_1 \\ s_2 \\ s_3 \end{bmatrix} = \begin{bmatrix} x_4 \\ b_2 \\ b_3 \\ b_4 \end{bmatrix}$$

where :- $x_1 = \frac{a_{12}}{a_{11}}$, $x_2 = \frac{a_{13}}{a_{11}}$, $x_3 = \frac{a_{14}}{a_{11}}$, $x_4 = \frac{b_1}{a_{11}}$

We can now subtract a multiple (a_{21}) of the first row from the second giving :-

$$\begin{bmatrix} 1 & x_1 & x_2 & x_3 \\ 0 & y_1 & y_2 & y_3 \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{bmatrix} \begin{bmatrix} s_0 \\ s_1 \\ s_2 \\ s_3 \end{bmatrix} = \begin{bmatrix} x_4 \\ y_4 \\ b_3 \\ b_4 \end{bmatrix}$$

where :- $y_1 = a_{22} - a_{21}x_1$, $y_2 = a_{23} - a_{21}x_2$

$$y_3 = a_{24} - a_{21}x_3 \text{ , } y_4 = b_2 - a_{21}x_4$$

By repeated application of the two above procedures to successively lower rows we can manipulate the equation into the form :-

$$\begin{bmatrix} 1 & k_1 & k_2 & k_3 \\ 0 & 1 & k_4 & k_5 \\ 0 & 0 & 1 & k_6 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} s_0 \\ s_1 \\ s_2 \\ s_3 \end{bmatrix} = \begin{bmatrix} k_7 \\ k_8 \\ k_9 \\ k_{10} \end{bmatrix}$$

Now by starting at the next to bottom row of this equation and working upwards, then by subtracting multiples of the row below we may achieve:-

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} s_0 \\ s_1 \\ s_2 \\ s_3 \end{bmatrix} = \begin{bmatrix} b'_1 \\ b'_2 \\ b'_3 \\ b'_4 \end{bmatrix}$$

which is the solution for S.

Having solved for $S_0..S_n$, we can substitute into (2.12) and obtain the n interpolating cubics $\phi_1(u)...\phi_n(u)$.

The above presents a method of piecewise approximation of two dimensional curves. This needs to be extended to three dimensions and account taken of the difficulties which arise due to infinite gradients.

Figure 2.8 shows a two dimensional (y,z) closed curve which we wish to approximate using cubic splines based on the points (knots) $P_0...P_3$. Clearly the sections P_1P_2 and P_3P_0 cannot be satisfactorily represented by cubic functions of y, since they have places of infinite gradient. This problem can be overcome by expressing y and z in terms of u, where u is the distance from P_0, measured along the straight line knot to knot path. Spline curves $\phi y(u)$ and $\phi z(u)$ may now be calculated using the values of x, y and u at the knots.

This technique is readily extended to three dimensions giving $\phi x(u)$, $\phi y(u)$ and $\phi z(u)$.

2.2.3 Path Manipulation

Having defined a weld path as a three dimensional space curve and having represented this path by a set of data points (knots) and interpolating functions, we now wish to consider manipulation of the path as an entity in order to cope with practical situations. It is usual to have several similar objects to be welded. The weld path is defined according to some nominal data, this same path being used for each object. There will, however, be some variation both in dimensions and orientation between the nominal and each object. The procedure below is prescribed as a

method of dealing with this situation.

Let three reference points $R_0 .. R_2$ be defined for the nominal object. By measuring the positions of the corresponding points $P_0..P_2$ on the actual object we can correct the path data.

Consider figure 2.9, which shows the case where there is dimensional variation in the object.

Let $S_{R1} = ((X_{R1} - X_{R0})^2 + (Y_{R1} - Y_{R0})^2 + (Z_{R1} - Z_{R0})^2)^{\frac{1}{2}}$

$S_{R2} = ((X_{R2} - X_{R1})^2 + (Y_{R2} - Y_{R1})^2 + (Z_{R2} - Z_{R1})^2)^{\frac{1}{2}}$

$S_{R3} = ((X_{R0} - X_{R2})^2 + (Y_{R0} - Y_{R2})^2 + (Z_{R0} - Z_{R2})^2)^{\frac{1}{2}}$

$S_1 = ((X_1 - X_0)^2 + (Y_1 - Y_0)^2 + (Z_1 - Z_0)^2)^{\frac{1}{2}}$

$S_2 = ((X_2 - X_1)^2 + (Y_2 - Y_1)^2 + (Z_2 - Z_1)^2)^{\frac{1}{2}}$

$S_3 = ((X_0 - X_2)^2 + (Y_0 - Y_2)^2 + (Z_0 - Z_2)^2)^{\frac{1}{2}}$

If k is the relative difference in size between the object and the nominal, then we may form a measure (E) of the error, due to measurement and/or distortion in the object.

$$E = E_1^2 + E_2^2 + E_3^2$$

where :- $E_1 = (S_{R1}k - S_1)$, $E_2 = (S_{R2}k - S_2)$, $E_3 = (S_{R3}k - S_3)$

Then :- $E = (S_{R1}k - S_1)^2 + (S_{R2}k - S_2)^2 + (S_{R3}k - S_3)^2$

Thus :- $\frac{dE}{dK} = 2(S_{R1}k - S_1)S_{R1} + 2(S_{R2}k - S_2)S_{R2} + 2(S_{R3}k - S_3)SR_3$

For minimum error $\frac{dE}{dK} = 0$

then :- $$(S^2_{R1} + S^2_{R2} + S^2_{R3})\, k = S_1 S_{R1} + S_2 S_{R2} + S_3 S_{R3}$$

giving :- $$k = \frac{S_1 S_{R1} + S_2 S_{R2} + S_3 S_{R3}}{S^2_{R1} + S^2_{R2} + S^2_{R3}} \qquad (2.16)$$

(2.16) gives the value of the scaling factor k to be applied to the nominal data for use with the object.

Considering the object triangle $P_0P_1P_2$ of figure 2.9 we can derive a set of manipulations which transform P_0 to the origin, P_1 onto the Y axis and P_2 onto the XY plane, i.e. :-

$$\text{Rot}(Y,\alpha)\ \text{Rot}(Z,\emptyset)\ \text{Rot}(X,\theta)\ \text{Trans}(-x_0, -y_0, -z_0)\ P_0P_1P_2 \qquad (2.17)$$

Similarly, we can derive a set of manipulations which transform R_0 to the origin, R_1 onto the Y axis and R_2 onto the XY plane :-

$$\text{Rot}(Y,\alpha_R)\ \text{Rot}(Z,\emptyset_R)\ \text{Rot}(X,\theta_R)\ \text{Trans}(-x_{R0}, -y_{R0}, -z_{R0})\ R_0R_1R_2 \qquad (2.18)$$

If we now scale the result of (2.18) by the factor k we have the same triangle as given by (2.17).

Hence $P_0P_1P_2$ = Trans (x_0, y_0, z_0) Rot(X, $-\theta$) Rot(Z, $-\emptyset$) Rot(Y, $-\alpha$)

Scale (k) Rot(Y, α_R) Rot (Z, $\emptyset_R$) Rot (X, θ_R)

Trans $(-x_{R0}, -y_{R0}, -z_{R0})$ $R_0R_1R_2$

Where Scale(k) is the scaling transformation, by a factor k.

Hence we can determine the transformation matrix A where :-

$$P_0P_1P_2 = A\ R_0R_1R_2$$

The transformation A is that which when applied to the nominal path gives

the path for the object.

The angles of rotation are given by :-

$$\theta = \tan^{-1} \frac{(z_1 - z_0)}{(y_1 - y_0)} \qquad \theta_R = \tan^{-1} \frac{(z_{R1} - z_{R0})}{(y_{R1} - y_{R0})}$$

$$\phi = \sin^{-1} \frac{(x_1 - x_0)}{S_1} \qquad \phi_R = \sin^{-1} \frac{(x_{R1} - x_{R0})}{S_{R1}}$$

$$\alpha = \tan^{-1} \frac{(z_2 - z_0)\cos\theta + (y_2 - y_o)\sin\theta}{(x_2 - x_0)\cos\phi - (y_2 - y_0)\sin\phi\cos\theta + (z_2 - z_0)\sin\phi\sin\theta}$$

$$\alpha_R = \tan^{-1} \frac{(z_{R2} - z_{R0})\cos\theta_R + (y_{R2} - y_{R0})\sin\theta_R}{(x_{R2} - x_{R0})\cos\phi_R - (y_{R2} - y_{R0})\sin\phi_R\cos\theta_R + (z_{R2} - z_{R0})\sin\phi_R\sin\theta_R}$$

Trajectory Control

In order to produce a required weld path, time varying position demands need to be generated for each of the manipulator axes. Sections 2.2.1 and 2.2.2 have shown how we may represent a path, piecewise, by mathematical functions. We now consider how to use these representations to generate the path with a manipulator.

2.3.1 Linearly Interpolated Trajectory

Here, between successive point (knots), the cartesian velocities are given by (2.4), (2.5) and (2.6). From these we may obtain the position equations :-

$$x = \frac{\Delta x}{\Delta S} v\, t + C_x \qquad (2.19)$$

$$y = \frac{\Delta y}{\Delta S} v\, t + C_y \qquad (2.20)$$

$$z = \frac{\Delta z}{\Delta S} v\ t + C_z \qquad (2.21)$$

where:- t is time

C_x is the X ordinate of the first knot

C_y is the Y ordinate of the first knot

C_z is the Z ordinate of the first knot.

By substituting into (2.19), (2.20) and (2.21) at equal intervals of time (Δt) we can obtain the position demands to be sent to the cartesian actuator/servos.

We can simplify this procedure by using forward-difference techniques to eliminate the need for repeated substitution.

Consider the equation :-

$x = mt + c$

at $t = t_1$, then $x_1 = mt_1 + c$

at $t = t_2$, then $x_2 = mt_2 + c$

thus :- $x_2 - x_1 = m(t_2 - t_1)$

Now, if $(t_2 - t_1) = \Delta t$

then:- $x_2 - x_1 = m\Delta t$

Thus having determined x_i we may evaluate $x_2 \ldots x_n$ by successive additions of the factor $m\Delta t$.

2.3.2 Cubic Spline Interpolated Trajectory

In this case we have a cubic (ϕx (u), ϕy(u), ϕ(u))for each of the cartesian axes, between successive knots.

Thus each ordinate is expressed in terms of the parameter u, where u is the straight line knot to knot distance measured from the first knot. We would like to re-parameterise these equations in terms of time (t).

Consider figure 2.10, this shows a curve $P_0 \ldots P_3$ based on the knots P_0, P_1, P_2 and P_3. Using the parameter u and the techniques described in section 2.2.2 we can obtain a set of interpolating cubics :-

$\phi x_1(u)$ $\phi x_2(u)$ $\phi x_3(u)$

$\phi y_1(u)$ $\phi y_2(u)$ $\phi y_3(u)$

$\phi z_1(u)$ $\phi z_2(u)$ $\phi z_3(u)$

Having obtained these, we can find the true path distance (S) from P_0 to any point on the curve by intergrating :-

$$S = (X^2 + Y^2 + Z^2)^{\frac{1}{2}}$$

Hence we can obtain the values of S (S_0, S_1, S_2, S_3) at the points P_0, P_1, P_2 and P_3 respectively.

Since the welding speed is constant then S is proportional to t. Thus, we may calculate the times t_0, t_1, t_2 and t_3 at which the torch passes through the points P_0, P_1, P_2 and P_3 respectively. These values of t may now be used to form a new set of interpolating cubics, parameterised in terms of t :-

$$\begin{array}{lll} \phi x_1(t) & \phi x_2(t) & \phi x_3(t) \\ \phi y_1(t) & \phi y_2(t) & \phi y_3(t) \\ \phi z_1(t) & \phi z_2(t) & \phi z_3(t) \end{array}$$

In order to reproduce the path on the manipulator we may substitute into the $\phi x(t)$, $\phi y(t)$ and $\phi z(t)$ equations at equal intervals (Δt) of time and feed the results to the actuator/servos.

Forward difference techniques may be used to eliminate the need for repetitive substitution.

Consider the four points x_0, x_1, x_2, and x_3 of the function :-

$$x = at^3 + bt^2 + ct + d$$

Let $(t_1 - t_0) = (t_2 - t_1) = (t_3 - t_2) = \Delta t$

Now

$$x_0 = at_0^3 + bt_0^2 + ct_0 + d \qquad (2.22)$$

$$x_1 = a(t_0 + \Delta t)^3 + b(t_0 + \Delta t)^2 + c(t_0 + \Delta t) + d \qquad (2.23)$$

$$x_2 = a(t_0 + 2\Delta t)^3 + b(t_0 + 2\Delta t)^2 + c(t_0 + 2\Delta t) + d \qquad (2.24)$$

$$x_3 = a(t_0 + 3\Delta t)^3 + b(t_0 + 3\Delta t)^2 + c(t_0 + 3\Delta t) + d \qquad (2.25)$$

writing $t_0 = t$ for clarity then :-

$$(2.23) - (2.22) = a(3t^2\Delta t + 3t\Delta t^2 + \Delta t^3) + b(2t\Delta t + \Delta t^2) + ct \qquad (2.26)$$

$$(2.24) - (2.23) = a(3t^2\Delta t + 9t\Delta t^2 + 7\Delta t^3) + b(2t\Delta t + 3\Delta t^2) + c\Delta t \qquad (2.27)$$

$$(2.25) - (2.24) = a(3t^2\Delta t + 15t\Delta t^2 + 19\Delta t^3) + b(2t\Delta t + 5\Delta t^2) + c\Delta t \qquad (2.28)$$

$$(2.27) - (2.26) = a(6t\Delta t^2 + 6\Delta t^3) + b(2\Delta t^2) \qquad (2.29)$$

$$(2.28) - (2.27) = a(6t\Delta t^2 + 12\Delta t^3) + b(2\Delta t^2) \qquad (2.30)$$

$$(2.30) - (2.29) = a6\Delta t^3 \qquad (2.31)$$

(2.31) shows that the third differences for cubic equations are constant and

determined by the coefficient (a) and the step size (Δ t).

Thus, when interpolating using a cubic equation we need only find the first three interpolated values by direct substitution. Successive values may be obtained by summing the first, second and third differences, the first and second differences being updated with each step. This technique can give significant reduction (greater than an order of magnitude) in the computation time required.

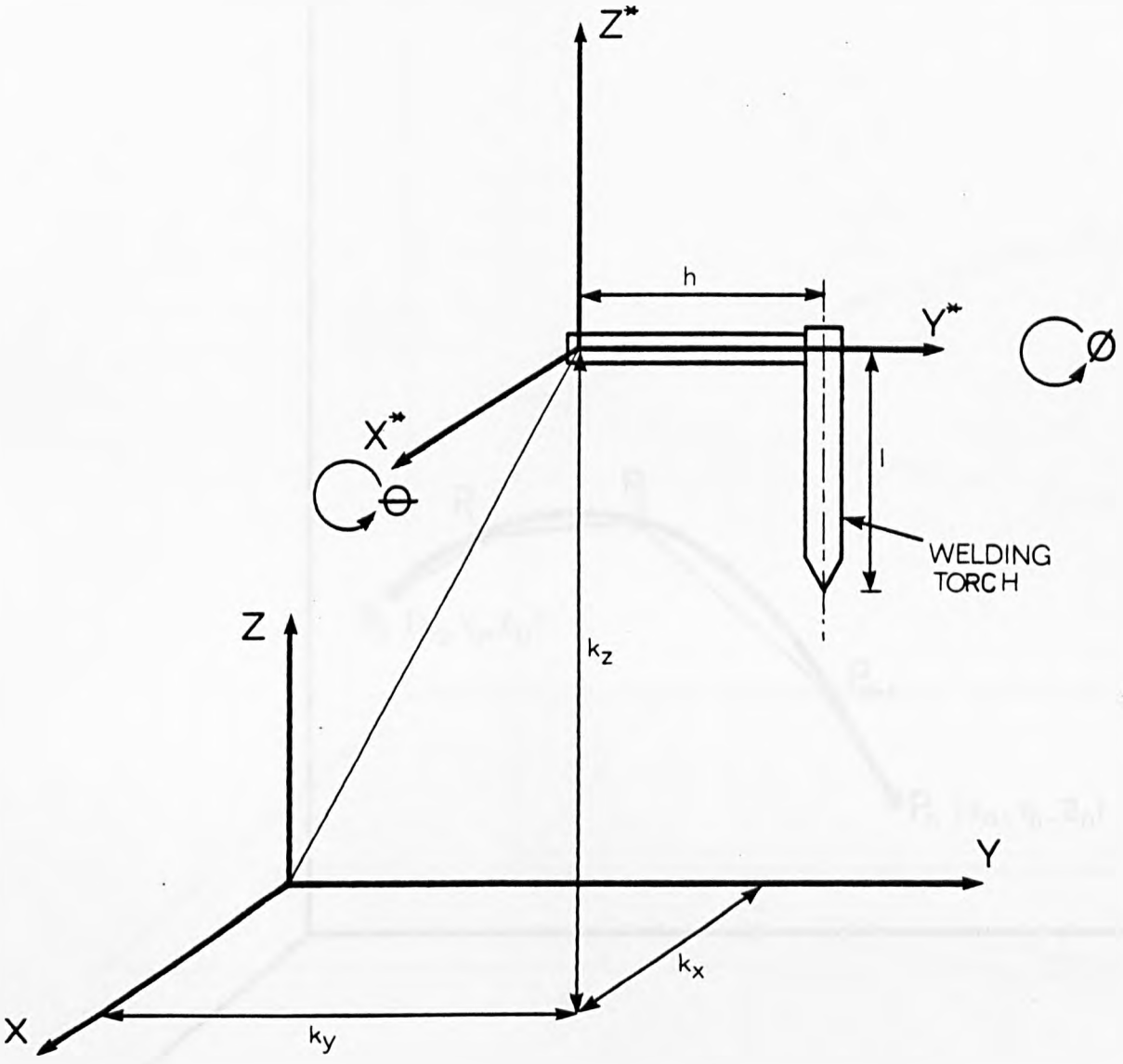


figure 2.1 WRIST KINEMATICS

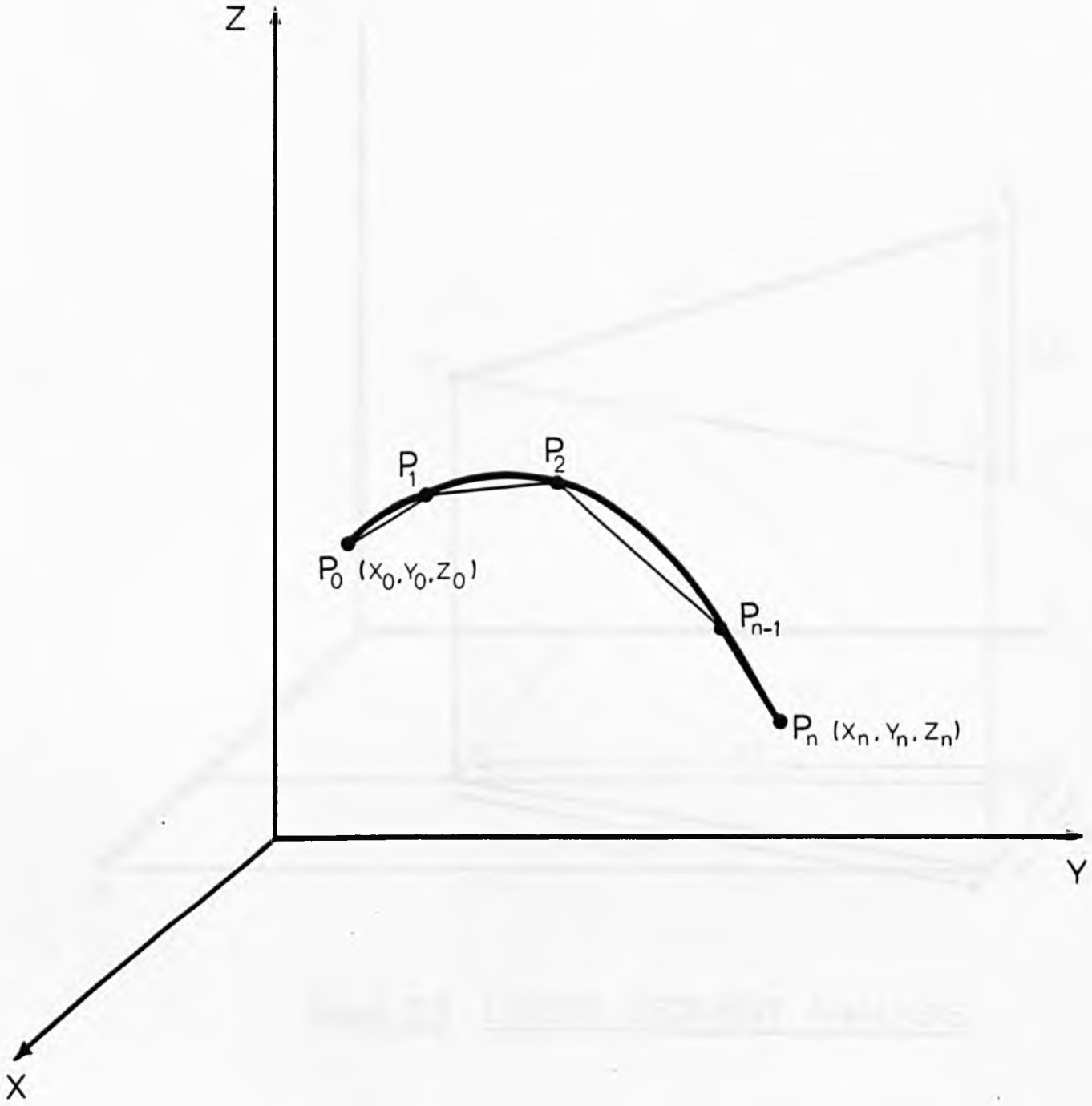


figure 2.2 LINEAR INTERPOLATION

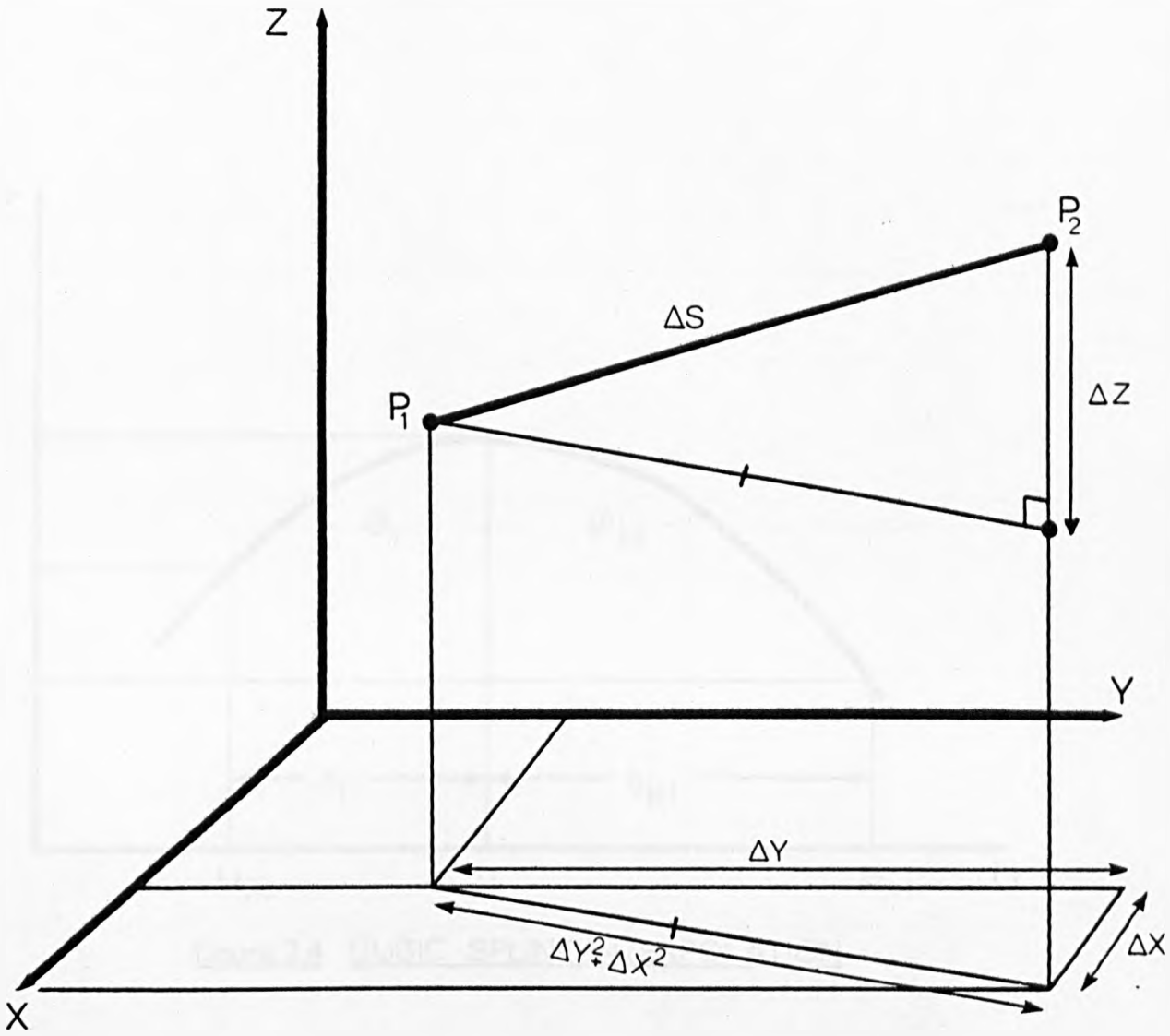


figure 2.3 LINEAR SEGMENT ANALYSIS

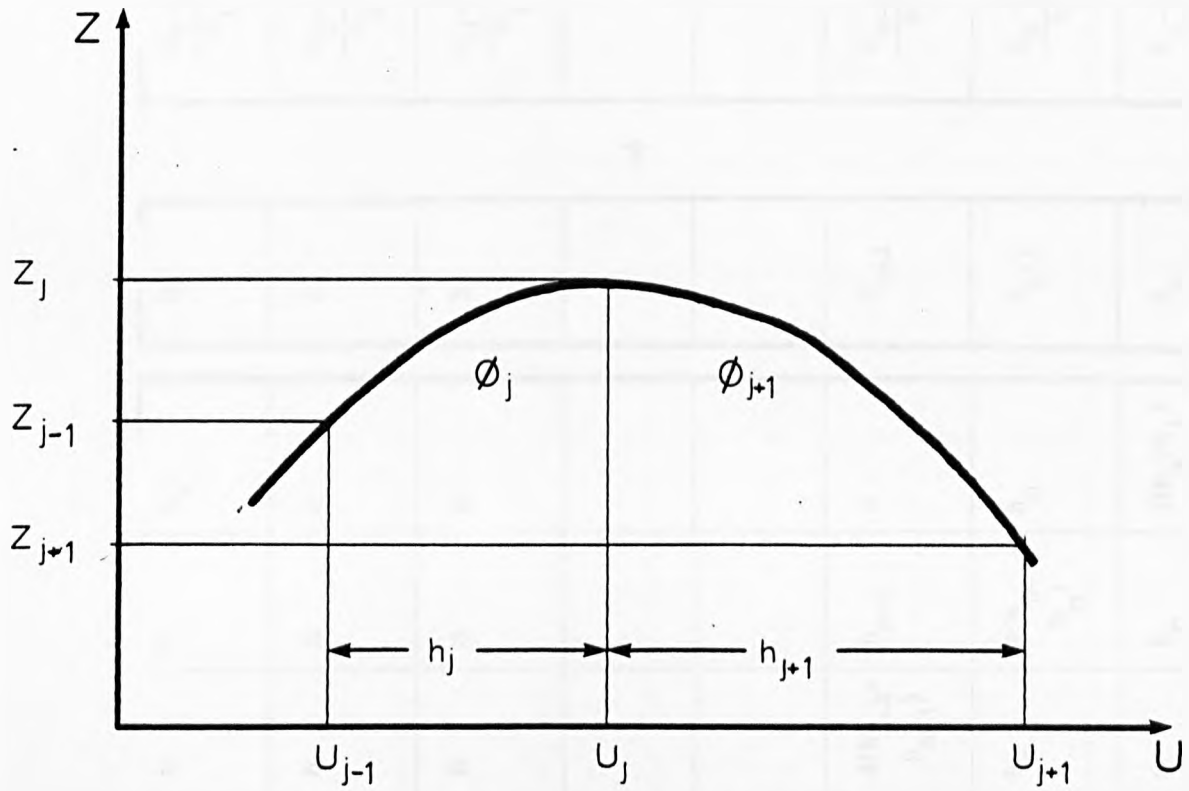


figure 2.4 CUBIC SPLINE INTERPOLATION

$$
\begin{bmatrix}
2(h_n+h_1) & h_1 & 0 & 0 & 0 & 0 & 0 & h_n \\
h_1 & 2(h_1+h_2) & h_2 & 0 & 0 & 0 & 0 & 0 \\
0 & h_2 & 2(h_2+h_3) & h_3 & 0 & 0 & 0 & 0 \\
 & & & & & & & \\
 & & & & & & & \\
0 & 0 & 0 & 0 & h_{n-2} & 2(h_{n-2}+h_{n-1}) & h_{n-1} & 0 \\
0 & 0 & 0 & 0 & 0 & h_{n-1} & 2(h_{n-1}+h_n) & h_n \\
h_1 & 0 & 0 & 0 & 0 & 0 & h_n & 2(h_n+h_1)
\end{bmatrix}
\begin{bmatrix} S_o \\ S_1 \\ S_2 \\ \\ \\ S_{n-2} \\ S_{n-1} \\ S_n \end{bmatrix}
=6
\begin{bmatrix}
\frac{Z_1-Z_o}{h_1} + \frac{Z_n-Z_o}{h_n} \\
\frac{Z_2-Z_1}{h_2} + \frac{Z_o-Z_1}{h_1} \\
\frac{Z_3-Z_2}{h_3} + \frac{Z_1-Z_2}{h_2} \\
 \\
 \\
\frac{Z_{n-1}+Z_{n-2}}{h_{n-1}} + \frac{Z_{n-3}-Z_{n-2}}{h_{n-2}} \\
\frac{Z_n-Z_{n-1}}{h_n} + \frac{Z_{n-2}-Z_{n-1}}{h_{n-1}} \\
\frac{Z_o-Z_n}{h_1} + \frac{Z_{n-1}-Z_n}{h_n}
\end{bmatrix}
$$

FIGURE 2.5 PERIODIC SPLINE MATRICIES

$$
\begin{bmatrix}
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 2(h_1+h_2) & h_2 & 0 & 0 & 0 & 0 & 0 \\
0 & h_2 & 2(h_2+h_3) & h_3 & 0 & 0 & 0 & 0 \\
 & & & & & & & \\
 & & & & & & & \\
0 & 0 & 0 & 0 & h_{n-2} & 2(h_{n-2}+h_{n-1}) & h_{n-1} & 0 \\
0 & 0 & 0 & 0 & 0 & h_{n-1} & 2(h_{n-1}+h_n) & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{bmatrix}
\begin{bmatrix} S_o \\ S_1 \\ S_2 \\ \\ \\ S_{n-2} \\ S_{n-1} \\ S_n \end{bmatrix}
=
6
\begin{bmatrix}
0 \\
\frac{Z_2-Z_1}{h_2} + \frac{Z_o-Z_1}{h_1} \\
\frac{Z_3-Z_2}{h_3} + \frac{Z_1-Z_2}{h_2} \\
 \\
 \\
\frac{Z_{n-1}+Z_{n-2}}{h_{n-1}} + \frac{Z_{n-3}-Z_{n-2}}{h_{n-2}} \\
\frac{Z_n-Z_{n-1}}{h_n} + \frac{Z_{n-2}-Z_{n-1}}{h_{n-1}} \\
0
\end{bmatrix}
$$

FIGURE 2.6 NATURAL SPLINE MATRICIES

$$
\begin{bmatrix}
2h_1 & h_1 & 0 & 0 & 0 & 0 & 0 & 0 \\
h_1 & 2(h_1+h_2) & h_2 & 0 & 0 & 0 & 0 & 0 \\
0 & h_2 & 2(h_2+h_3) & h_3 & 0 & 0 & 0 & 0 \\
 & & & & & & & \\
 & & & & & & & \\
0 & 0 & 0 & 0 & h_{n-2} & 2(h_{n-2}+h_{n-1}) & h_{n-1} & 0 \\
0 & 0 & 0 & 0 & 0 & h_{n-1} & 2(h_{n-1}+h_n) & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & h_n & 2h_n
\end{bmatrix}
\begin{bmatrix} S_o \\ S_1 \\ S_2 \\ \\ \\ S_{n-2} \\ S_{n-1} \\ S_n \end{bmatrix}
=6
\begin{bmatrix}
\frac{Z_1-Z_o}{h_1} - g_o \\
\frac{Z_2-Z_1}{h_2} + \frac{Z_o-Z_1}{h_1} \\
\frac{Z_3-Z_2}{h_3} + \frac{Z_1-Z_2}{h_2} \\
 \\
 \\
\frac{Z_{n-1}-Z_{n-2}}{h_{n-1}} + \frac{Z_{n-3}-Z_{n-2}}{h_{n-2}} \\
\frac{Z_n-Z_{n-1}}{h_n} + \frac{Z_{n-2}-Z_{n-1}}{h_{n-1}} \\
\frac{Z_{n-1}-Z_n}{h_n} + g_n
\end{bmatrix}
$$

FIGURE 2.7 BUILT IN END SPLINE MATRICIES

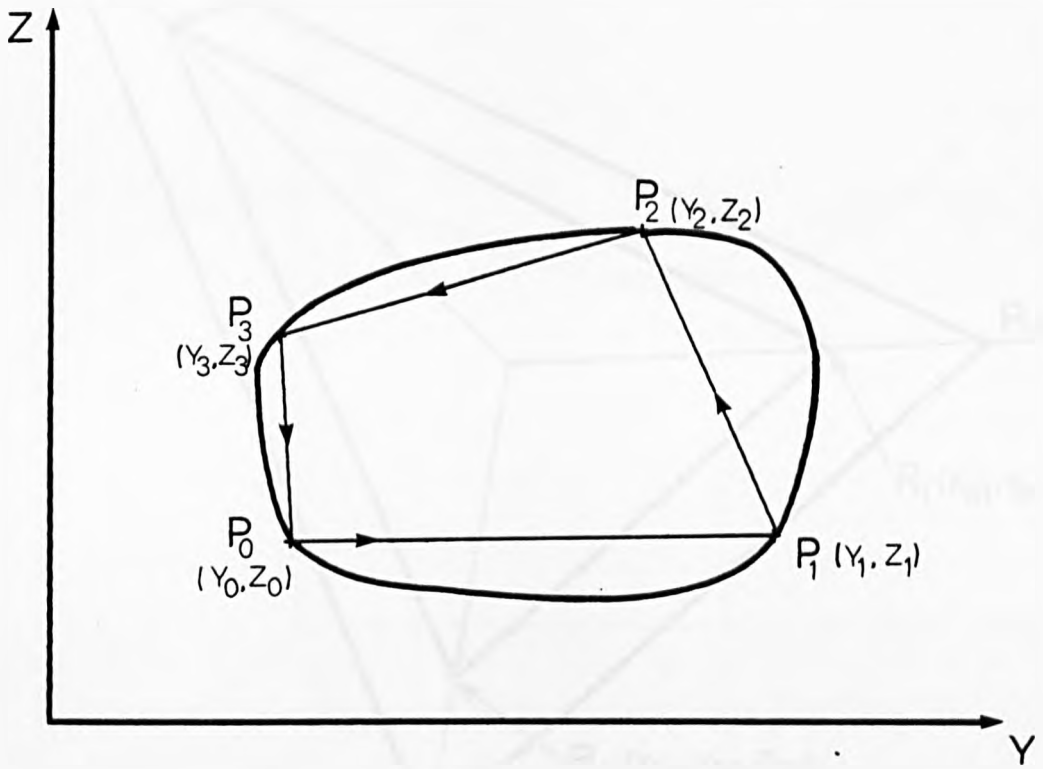


figure 2.8 CLOSED CURVE APPROXIMATION

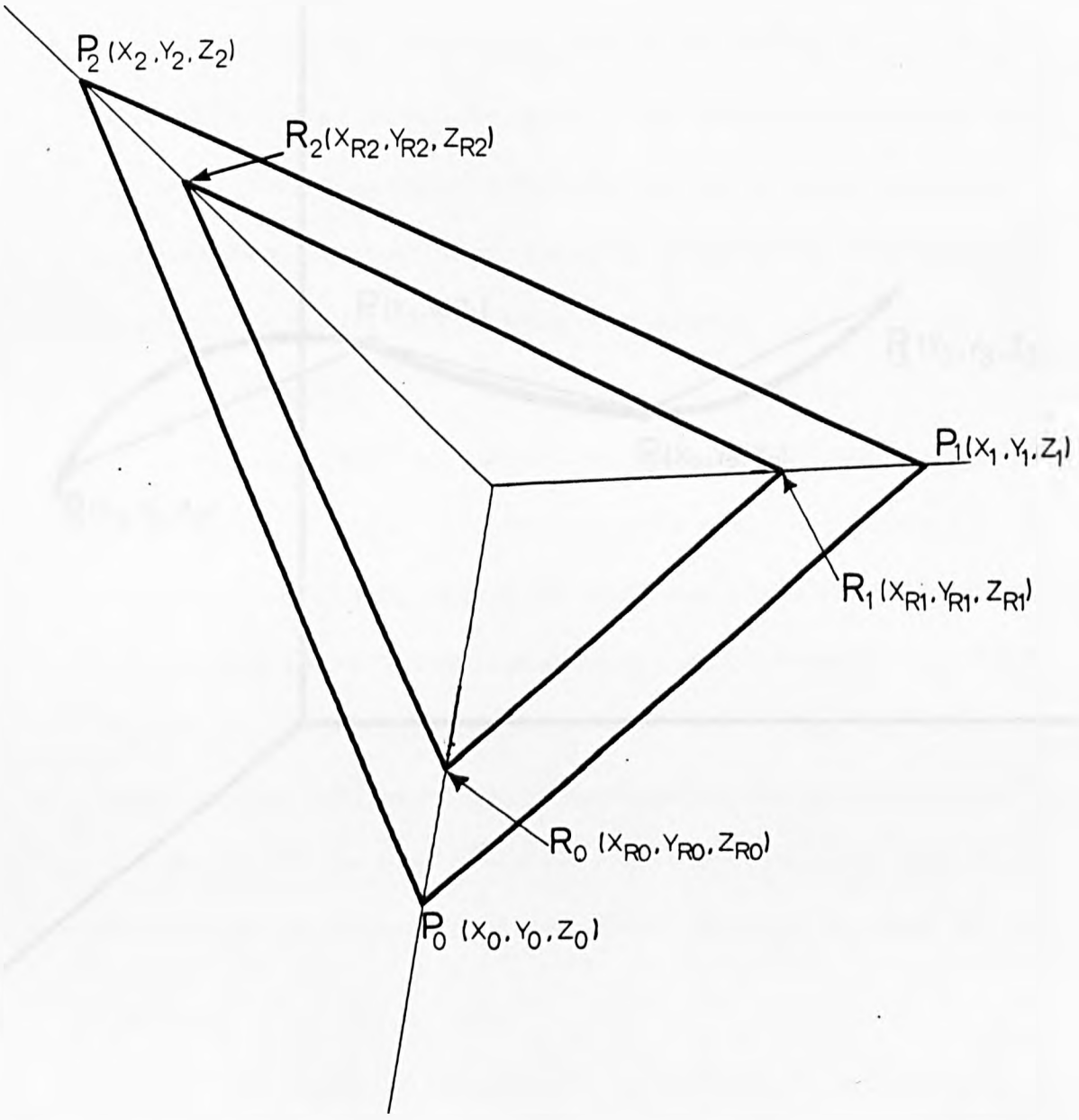


figure 2.9 OBJECT DIMESIONAL VARIATION

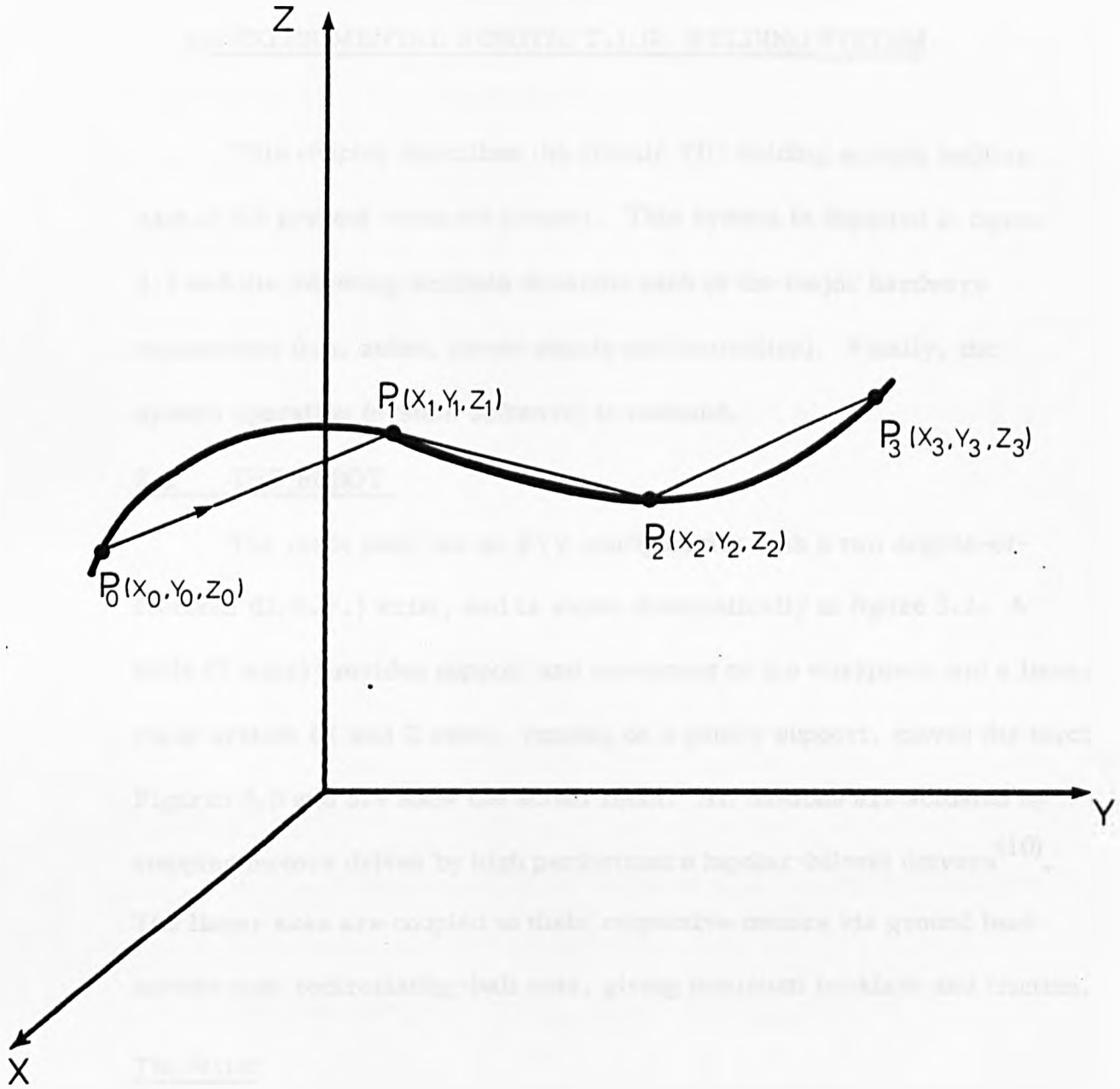


figure 2.10 SPLINE RE-PARAMETERISATION

CHAPTER 3

AN EXPERIMENTAL ROBOTIC T.I.G. WELDING SYSTEM

This chapter describes the robotic TIG welding system built as part of the present research project. This system is depicted in figure 3.1 and the following sections describe each of the major hardware components (i.e. robot, power supply and controller). Finally, the system operation (system software) is outlined.

3.1 THE ROBOT

The robot built has an XYZ configuration with a two degree-of-freedom (D.O.F.) wrist, and is shown schematically in figure 3.2. A table (Y-axis) provides support and movement of the workpiece and a linear track system (X and Z axes), running on a gantry support, moves the torch. Figures 3.3 and 3.4 show the actual robot. All motions are actuated by stepping motors driven by high performance bipolar-bilevel drivers[10]. The linear axes are coupled to their respective motors via ground lead-screws with recirculating-ball nuts, giving minimum backlash and friction.

The Wrist

From figure 3.5 the relationship between the position angles, A and B, of the wrist motors and the torch angles θ and ϕ (as used in the analysis in section 2.1) can be seen as :-

$$\phi = \frac{N_1}{N_2} \frac{(A - B)}{2}$$

$$\theta = \frac{N_1}{N_2} \frac{(A + B)}{2}$$

where :

A and B are the motor position angles

N_1 and N_2 are the number of gear teeth on the differential gears (as shown in figure 3.5).

For the wrist constructed $N_1 = 36$ and $N_2 = 50$.

Since the resolution of the stepping motors is 0.9^o, and a gearbox with a ratio N : 1 was used, then the resolution of the torch rotations θ and ϕ is given by :-

$$\text{Wrist resolution} = \frac{N_1}{N_2} \times 0.9^o \times \frac{1}{N}$$

$$= 0.65^o \times \frac{1}{N} \qquad (N = 1)$$

The Linear Axes

In considering the speed performance of the linear axes we need to know the loads presented to the motors.

The total inertial load (I_{total}) on a motor is given by

$$I_{total} = I_{motor} + I_{screw} + I_{load}$$

where :

I_{motor} is the moment of inertia of the motor's rotor.

I_{screw} is the moment of inertia of the leadscrew.

I_{load} is the effective moment of inertia of the load, referred to the motor shaft.

I_{motor} is given in the manufacturer's data for a particular motor.

I_{screw} can be calculated (approximating the screw by a cylinder)[11], i.e. :

$$I_{screw} = \frac{\pi p l r^4}{2} \quad (kgm^2)$$

where :

p = density of the material of the screw (kgm^{-3})

l = length of the screw (m)

r = radius of the screw (m).

I_{load} can be calculated from[12] :

$$I_{load} = M \left(\frac{L}{2\pi}\right)^2 \quad (kgm^2)$$

where :

M = mass of load (kg)

L = lead of the screw (m).

Table 3.1 summarises the above parameters for each of the linear axes of the robot.

When using stepping motors we need to ensure that the motor position angle remains in synchronism with the applied stepping pulses, in order that permanent positional errors do not occur. In particular there is a critical angle for the lag between the motor response and the applied pulses which if exceeded will cause a loss of synchronism[13]. For the motors in question this angle is 3.6°, i.e. corresponding to four stepping pulses.

We are particularly interested in knowing the maximum rate of pulses which may be applied to a motor at rest without losing synchronism. This pulse rate is known as the Start/Stop speed of the motor. Consider the case of accelerating the motor from rest to a speed of W steps per

second, the torque (T) required is given by :-

$$T = T_{static} + I_{total} \frac{W \pi \alpha}{t \ 180} \quad (Nm) \qquad (3.1)$$

where :-

α is the step angle of the motor (= 0.9^{o})

I_{total} is the total inertial load on the motor (Nm^2)

T_{static} is any static torque applied to the motor (Nm)

t is the time taken to reach the final speed (s).

We require a Start/Stop speed of 10^{-2} ms^{-1}, minimum, on each of the linear axes. On the X and Z axes this corresponds to a stepping pulse rate of 10^3 steps per second, and on the Y axis a rate of 0.8×10^3 steps per second. Using an acceleration time t = 4/W will ensure synchronism. Thus, (3.1) becomes :-

$$T = T_{static} + I_{total} \frac{W^2 \ \pi \ 0.9}{4 \quad 180} \quad (Nm)$$

$$= T_{static} + \frac{I_{total} \ W^2 \ \pi}{800} \quad (Nm) \qquad (3.2)$$

The required motor torques may be calculated by substituting into (3.2), using the above values and values from table 3.1. The results obtained are shown in table 3.2. These results may be compared with the actual motor torques obtained at the given stepping rate (as given in figure 3.6). These comparisons are given in table 3.2 and it can be seen that in all cases the actual motor torque is greater than that required, indicating that the Start/Stop speed of 10^{-2} ms^{-1} is obtainable.

Further, by repeatedly increasing the value of W substituted into (3.2) until the required torque becomes equal to the actual value, then

an indication of the maximum Start/Stop speed obtainable on each axis may be found. Table 3.2 also shows these values and it can be seen that they are at least 50% higher than the required value, thus ensuring adequate performance. Also it should be noted that if required the Start/Stop speeds could be increased by increasing the acceleration time (t) used.

Working Envelope

Table 3.3 shows the range of movement allowed on each axis of the robot and effectively defines it's working envelope. Also the resolution of each axis is stated.

3.2 TIG WELDING POWER SUPPLY

The power supply used is fundamentally a conventional series-pass regulator current source, as shown in figure 3.7. Here a smoothed, unregulated D.C. source (V_{dc}) is derived from the mains A.C. supply using a transformer (T1), a full wave bridge rectifier (D1) and a smoothing capacitor (C1). V_{dc} provides the collector voltage for the series-pass transistor (TR1) which controls the output current. The base of TR1 is driven by an error amplifier which compares a reference voltage (V_{ref}) and a feedback voltage (V_{fb}) (taken from across the current shunt resistor (R1)).

When V_{fb} is less than V_{ref} the drive to TR1 is increased. This causes TR1 to pass more current through the load, thus increasing V_{fb}. Under operating conditions V_{fb} settles to V_{ref}.

Now, $V_{fb} = I\ R1$ (V)

where :-

I = load current (A)

Thus, $I = \frac{V_{fb}}{R_1}$

$= \frac{V_{ref}}{R1}$

The output current (I) may therefore be controlled by varying the reference voltage (V_{ref}).

It should be noted that the open-circuit output voltage (i.e. for an infinite impedance load) is equal to V_{dc}.

The actual supply used was based on a Welding Institute Fifty Amp Polypack[14] and is outlined in figure 3.8. Here the series pass element consists of fifty bipolar transistors connected in parallel. The reference input to the error amplifier is derived from a digital-to-analogue converter (DAC), which is detailed in figure 3.9. The inputs to the DAC are active low, thus an input of 11111111 (binary) gives zero output current whilst an input of 00000000 gives fifty amps. 'Pull-up' resistors on the input lines ensure zero output current whenever no external input is connected, thus giving a 'fail-safe' condition.

Power supply sequencing is controlled by the /START (where the '/' signifies that the signal is active-low) input and is shown in figure 3.10. A low-going level on /START disables the main supply, enables the pilot supply and turns on the shielding gas. After a delay $T_{pre\text{-}purge}$ the high frequency (H.F.) arc starting unit is enabled. This establishes an arc, powered by the pilot supply. When arc current is sensed the H.F. unit and pilot supply are disabled and the main supply enabled. The procedure of starting on the pilot supply is necessary since the series regulator

cannot withstand the superimposed high frequency (about 1 MHz) oscillations of the arc starting unit.

When /START goes from low to high the main supply is disabled and after a delay $T_{post\text{-}purge}$ the gas is turned off. In this manner shielding gas is maintained until the weld cools sufficiently. Figure 3.11 shows the arc detection circuitry added to the power source. Here the arc voltage is used to control the 'ARCON' switch (SW1). The switch will be open for arc voltages greater than 21 V and closed for lower voltages. Thus the switch closes only when an arc is established.

3.3 SYSTEM CONTROLLER

The controller used is based on the Quarndon QMS[15] series of microcomputer boards. The computer and user interfaces are housed in a single 3U, nineteen inch rack mounting unit, and consists of the following items :-

(a) a QMS 8511-85B 8085 based microcomputer board.

(b) A QMS 00-1419-3 AMD9511[16] based arithmetic processor board.

(c) A QMS 00-1201-D 32K byte dynamic RAM memory board.

(d) A QMS 00-1400 Prototyping board (user interfaces).

Figure 3.12 shows the layout (and IC type number cross references) of the interface board, the circuit details of which are shown in figures 3.13 to 3.16.

The user interfaces are discussed individually in the following paragraphs.

Wrist Motor Interface

Stepping pulses for the wrist motors are generated by direct digital output (IC21, IC25 figure 3.15). The time delay between pulses is given by a counter/timer (counter 2 on IC1, figure 3.14), which produces an interrupt (/RST6.5) at the end of the delay.

Linear Axes Interface

Figure 3.20 shows, functionally, the linear stepping interface. Each motor (X, Y and Z) has associated with it a divider and a counter. The divider determines the frequency of the stepping pulses, and hence the motor speed, whilst the counter counts these pulses and thus measures distance moved. Also, each motor has an /ENABLE signal which enables motion.

When the controller requires a movement on a linear axis it firstly loads the corresponding divider with a speed value. Then it loads the counter with the distance of the required movement and sets the /ENABLE signal active. As the motor is stepped the value held in the counter is decremented. When zero is reached the counter output ('=0') disables further movement, by inhibiting the divider, and generates an interrupt (/RST6.5) to inform the computer.

Thus, in order to obtain a linear point-to-point motion the controller needs only to load the associated dividers and counters and set the /ENABLE signals. The controller is then free to carry out further processing and will be informed (/RST6.5) when the motion is complete.

The motor directions are set by direct digital output (IC4, figure 3.14).

Figures 3.14 and 3.15 show the stepping motor interfaces schematically. All signals are electrically isolated from the motor drivers by opto-isolators.

Power Source Interfaces

The signals for controlling the power source are derived from digital output ports as shown schematically in figure 3.16. /PSUDO to /PSUD7 (IC19, IC23) are the power source current level control lines, and /START (IC23) controls the power source sequencing. Again the computer is electrically isolated by opto-isolators (IC19, IC20 and IC23).

The arc detection signal (/ARCON) is sensed by both a digital input (IC22, figure 3.13) and an interrupt (/RST7.5, figure 3.13).

Pendant Keypad Interface

The pendant keypad is configured as a matrix of keys, shown schematically in figure 3.19, which are strobed and read by the I/O ports of the 8155[17] of the QMS 8511-85B microcomputer board. Also a set of light emitting diodes are provided to give visual status indications.

Figure 3.18 details a wiring modification implemented, on the QMS 8511-85B, in order to use port C of the 8155 as an output port.

Tape Interface

Audio cassette data storage is achieved using a Quarndon QMS-1000[15] cassette interface. Figure 3.17 shows how this unit is connected between the microcomputer and the VDU. Data sent to the VDU from the microcomputer may be stored onto tape, and data may be read from tape to the microcomputer. Normally the microcomputer communicates with the VDU at a rate of 1200 Baud. When it wishes to read or write to take it changes speed to 300 Baud.

3.4 ROBOT CALIBRATION

The main requirement of the robot was to move the welding torch with a constant speed over the workpiece to be welded in order to test out the controller. The robot was designed to give a positional accuracy of better than 0.1 mm at any position within its working envelope, and to have a speed accuracy of better than 1% in the range 10 (mm s^{-1}), to 0.5 (mm s^{-1}). The following paragraphs outline some of the tests carried out to ensure these values were achieved.

Axis Alignment

It was necessary to ensure that the robot X, Y, Z axes were all orthogonal and the worktable was parallel to the X and Y slides. The worktable was set horizontal using a spirit level and laser beam[22]. The Y and Z slides were also adjusted by using a spirit level, laser beam and plumb-line. The overall accuracy was examined and finely adjusted by attaching a dial gauge to the robot Z axis and then moving the gauge over the worktable. The dial gauge readings were used to confirm that correct alignment had been achieved.

Axis Calibration

A fixed number of pulses were applied to each axis motor and the distance traversed measured. It was confirmed that each step produced movements of 0.010 mm (X and Z axes) and 0.0125 mm (Y axis). Backlash in all cases accounted for less than 0.03 mm. By pulsing at a fixed rate speed measurements were carried out and better than 1% accuracy was confirmed.

In order to check that no pulses were 'lost' when accelerating from rest to maximum speed it was necessary to measure the distance moved for a small number (one hundred) pulses, using a dial gauge. Using this method a start/stop speed greater than 10 mm s^{-1} was confirmed.

3.5 SYSTEM SOFTWARE

This section describes the operating procedures for the robotic TIG welding system, and hence shows the facilities provided by the system software. Appendix A contains a complete set of source listings for the programs used. These are mainly written in PLM80[18], with some interrupt routines being in ASM80[19]. All program development was carried out on an Intellec[20] series II microcomputer development system, using the ISIS-II[21] operating system.

Operating Procedures

Basically the user is allowed to enter sets of welding parameters and to program a weld seam by moving the torch along the required trajectories. These welds may then be performed by the robot.

Welding Parameters

A set of welding parameters consists of the following items (as defined in figure 1.10) :-

(a)	Pulse current	0-50 (A)
(b)	Background current	0-50 (A)
(c)	Pulse time	0-655 (s)
(d)	Background time	0-655 (s)
(e)	Welding speed	0-10 (10^{-3}ms^{-1})
(f)	Fall time (slope out)	0-655 (s)
(g)	Pre-traverse time	0-655 (s)

Up to two hundred and fifty six sets of parameters may be defined, each set being identified by it's set number (0-255).

Weld Path

A single path may be held in the system memory. This path may consist of up to sixty separate weld seams. These seams will be executed in turn with the controller starting and stopping the welding power source at the beginning and end of each seam. The torch begins and ends the path at the same point. This point is called the 'rest-position' and is programmed along with the path.

Torch Positioning

The pendant keypad allows the user to move the welding torch. Two keys are provided for each of the robot's motors, giving forward and reverse action when pressed. Also the motor speeds may be increased or decreased by pressing the FAST or SLOW keys, respectively.

The ENTER key is used to record the present torch position. The START and STOP keys are used to inform the computer of the positions of the beginning and end of weld seams, allowing welding and non-welding movements to be distinguished.

Light emitting diode (LED) lamps are provided to indicate the keypad status. The ACTIVE LED is illuminated whenever the keypad is enabled by the controller. Pressing a key when this lamp is off will have no effect. The SEAM LED is illuminated when the present torch position corresponds to a welding position. Each of the wrist motors has an LED associated with it (R0 and R1). These are illuminated when the corresponding motor has been moved, and remain on until the motor position has been 'entered'. These are needed since the controller does not permit simultaneous wrist and linear axis motion, and serve to remind the user.

System Commands

Figure 3.21 is a simplified state diagram for the system software. At 'power-on' the RESET state is entered. The user is prompted to define the system origin and power source current range. Only when valid values for these have been entered will the MONITOR state be entered.

The origin is a cartesian reference point, within the robot's working envelope, which is to be taken as (0, 0, 0). Using the pendant keypad the user should position the torch-tip at the required origin and press the ENTER key.

The current range is the full range current value (in Amps) for the welding power source used. For the source described previously this is fifty.

Figure 3.22 shows the VDU screen, having entered the MONITOR state. The user may now enter any of the command states (as shown in figure 3.21) by typing a valid command name followed by a carriage-return (<CR>). These commands are summarised below.

Parameter Command

This command state is entered from the MONITOR level by typing PARAM <CR> .

The user is prompted for a parameter set number (0-255). The program will only accept a value if all preceding values have already been entered.

Having accepted a set number the program displays the present values, if any, of all the parameters in that set, and prompts for a new value for each in turn. The user should type each new value followed by <CR> . If a parameter was previously defined then typing <CR>

without a value will leave it unchanged. All values are checked on entry and only those found to be valid will be accepted.

When a complete set of parameters has been entered the program will prompt for another set number. The user may enter a set number and repeat the above process or return to the MONITOR level by typing ⟨CR⟩ without a value.

Sequence Command

This command state is entered from the MONITOR level by typing SEQU ⟨CR⟩ .

A weld path may contain up to sixty seams. Each of these seams must be assigned a set of weld parameters by assigning a set number to the seam.

The user is prompted for seam number and corresponding parameter set numbers. The user may return to the MONITOR level by entering ⟨CR⟩ without a value when prompted for a seam number.

Position Command

This command state is entered from the MONITOR level by typing POSIT ⟨CR⟩ .

Using the pendant keypad the user may move the welding torch to any required position. When finished the user may return to the MONITOR level by pressing ENTER on the keypad.

Limit Command

This command state is entered from the MONITOR level by typing LIMIT ⟨CR⟩ .

The user will be prompted to set motion limits on the linear

axes. This is done by moving the torch to the required limit position and pressing ENTER on the keypad.

Once the limits are set the controller will not allow further movement beyond those limits.

Path Command

This command state is entered from the MONITOR level by typing PATH ⟨CR⟩ .

The user is prompted to type a name (followed by ⟨CR⟩ which can latter be used as a reference to the path.

Having entered a name a weld path may be programmed by moving the torch (using the pendant keypad) from one point to the next along the path, and pressing ENTER at each point. Start points of weld seams should be recorded using the START key instead of ENTER, and end points recorded using the STOP key.

The final point entered for a path should not be on a weld seam and should be recorded using the STOP key. Under these conditions the controller will prompt the user to enter 'Y' at the console to confirm that the point is the last required. If the final point is within two centimetres of the rest-position then the controller will return the torch to there. Otherwise the user is prompted to return the torch to the rest-position (within two centimetres).

Weld Command

This command state is entered from the MONITOR level by typing WELD ⟨CR⟩ .

The WELD command will only execute if a path has previously

been entered and all the sets of parameters assigned to the weld seams have been defined. If these conditions are not fulfilled then an appropriate warning will be given and the MONITOR state return to

If the command executes then the user will be prompted for a choice of path tracing or welding. A reply of 'T ⟨CR⟩' will cause the previously entered path to be traced by the robot, without any welding taking place. A reply of 'W ⟨CR⟩ ' will cause the robot to carry out the welding sequence defined by the entered path.

Dump Command

This command state is entered from the MONITOR level by typing DUMP ⟨CR⟩ .

The user should then switch the cassette tape machine to 'record'. The path, weld parameters and motion limits (if any) will be dumped onto tape from memory.

Load Command

This command state is entered from the MONITOR level by typing LOAD ⟨CR⟩.

The user should then switch the cassette tape machine to 'replay'. Previously 'dumped' data will then be reloaded into memory. The user will be informed of any data errors which may occur during loading. If an error does occur then the data in memory is taken to be invalid.

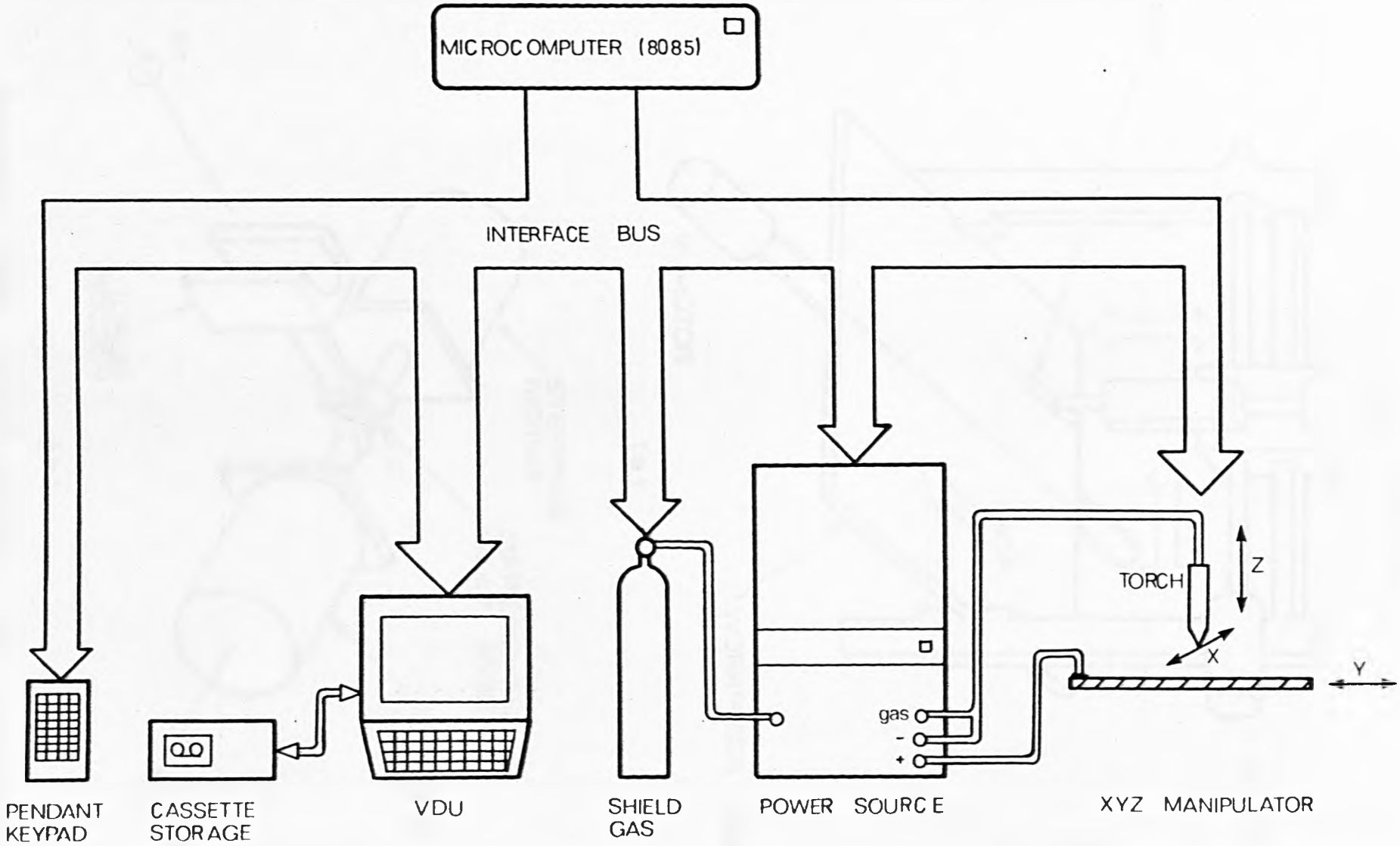
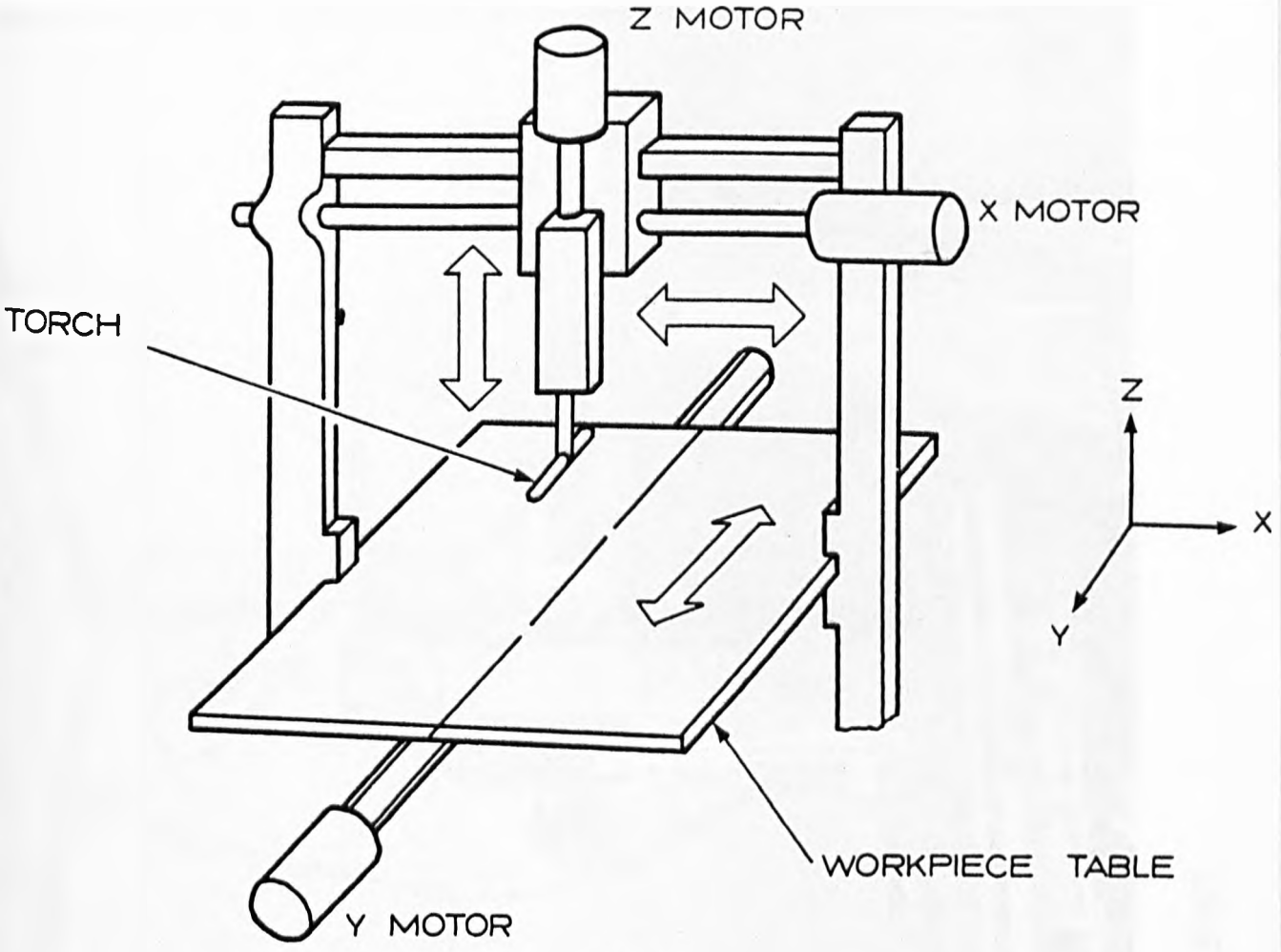


FIGURE 3.1 THE TIM3 TIG WELDING SYSTEM



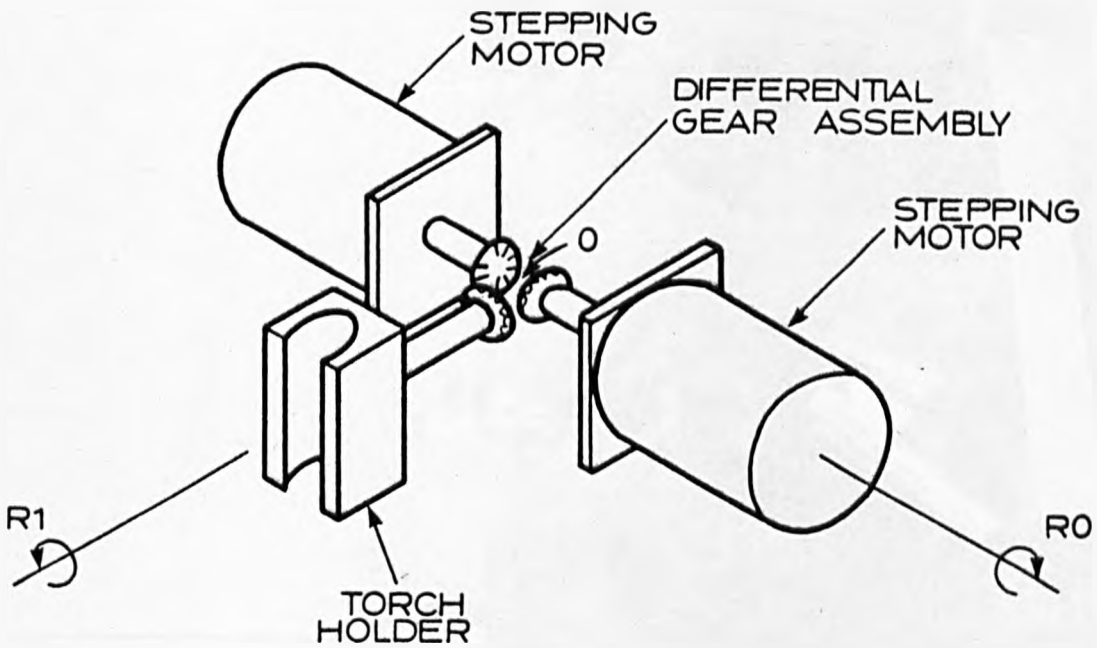


Figure. 3.2 THE TIM-3 ROBOT TIG MACHINE.
(a) TORCH AND WORKPIECE TABLE ARRANGEMENT.
(b) WRIST ACTION FOR PREPOSITIONING OF THE TORCH.

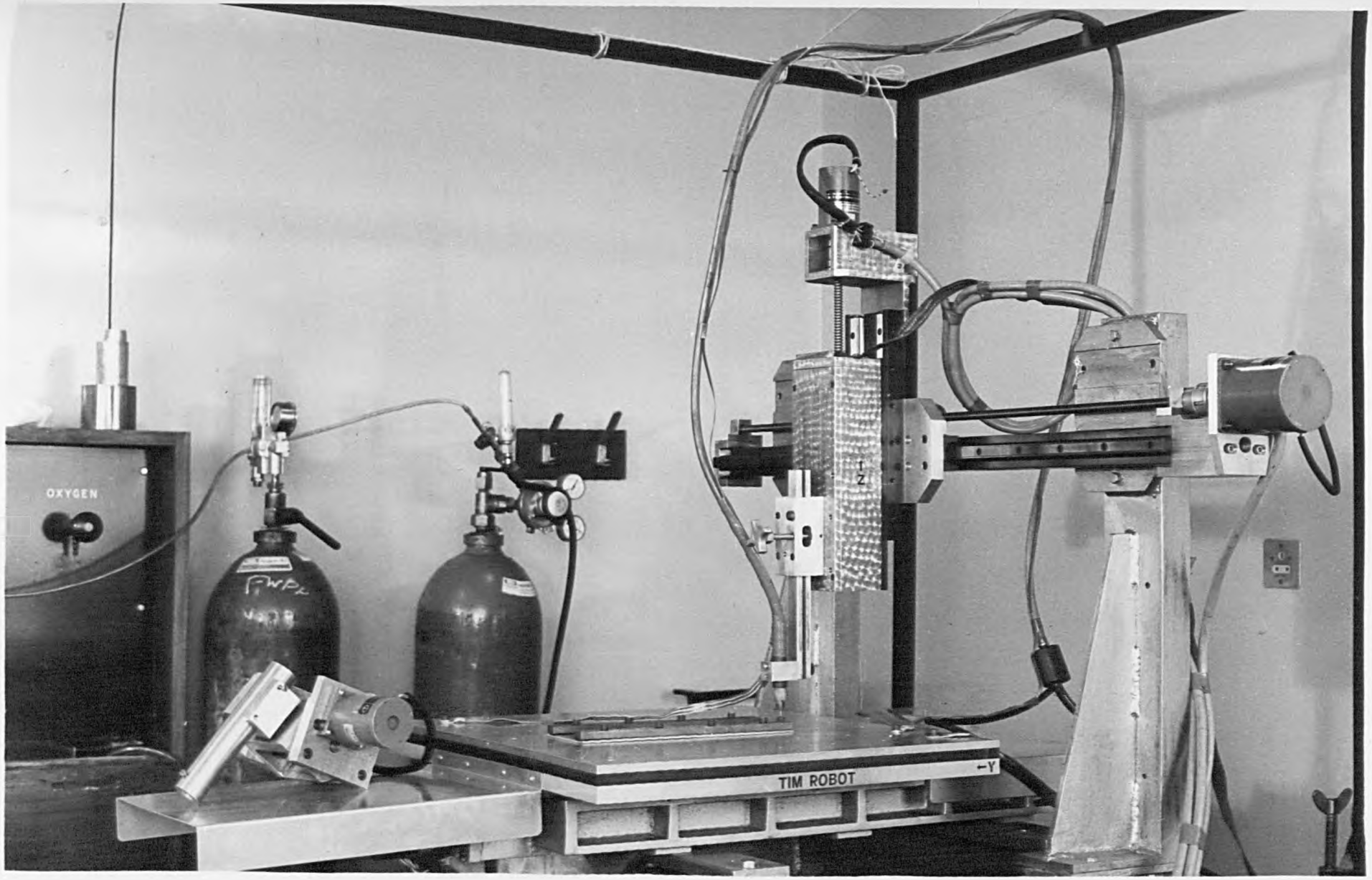


FIGURE 3.3 THE TIM 3 ROBOT

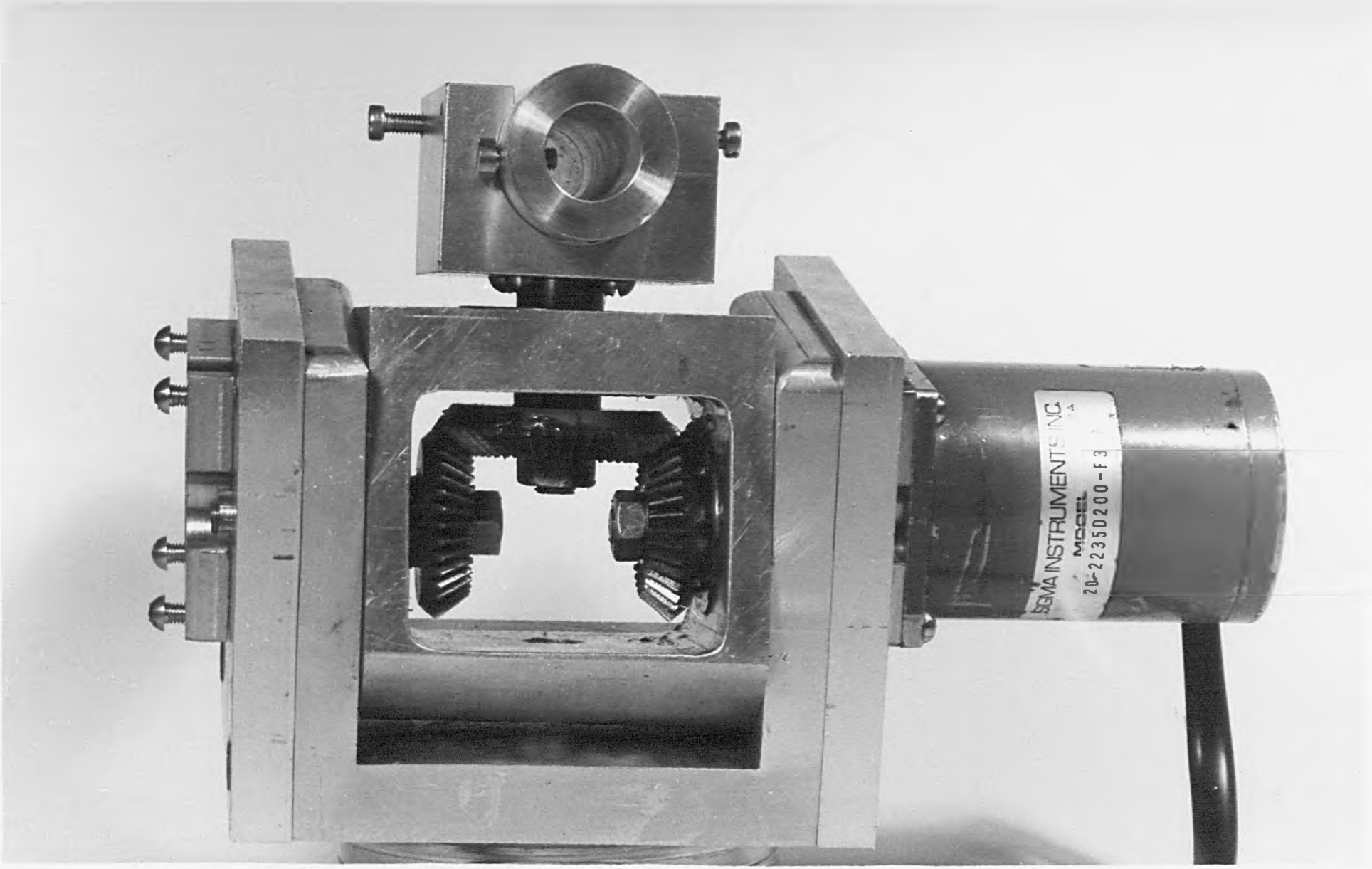


FIGURE 3.4 THE TWO DEGREES OF FREEDOM WRIST

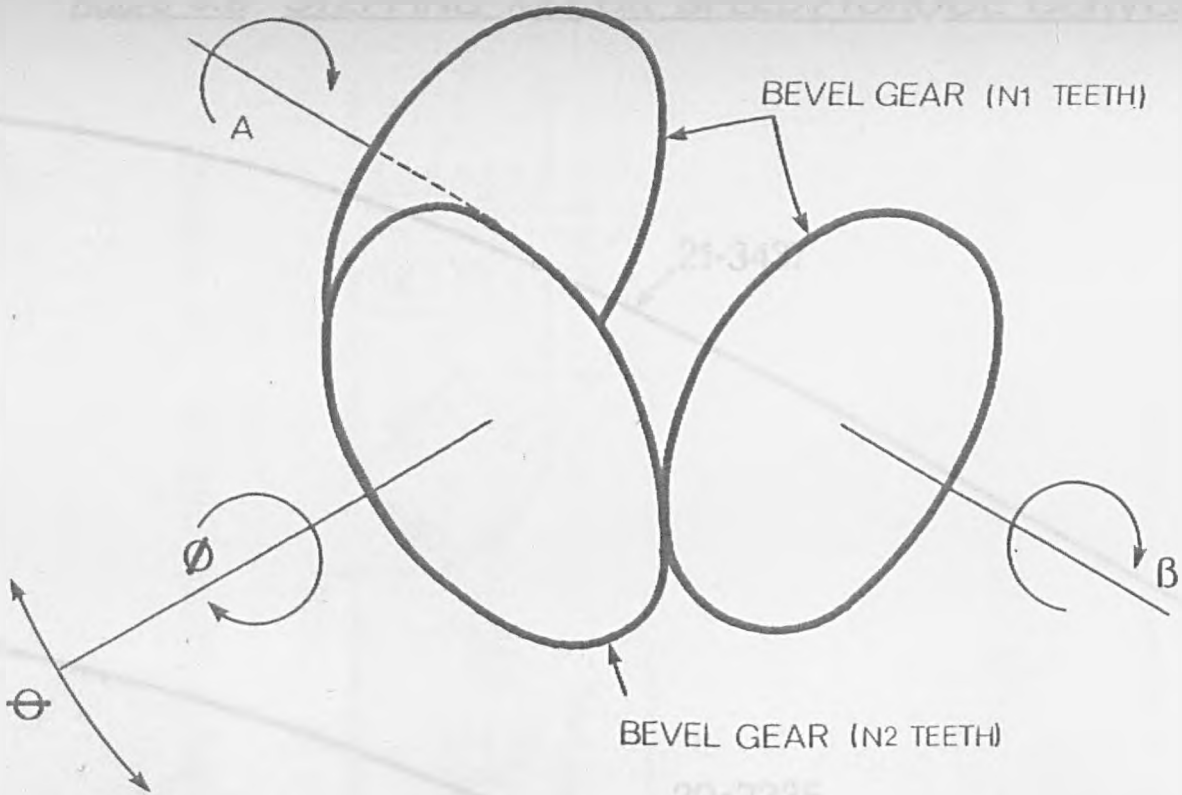
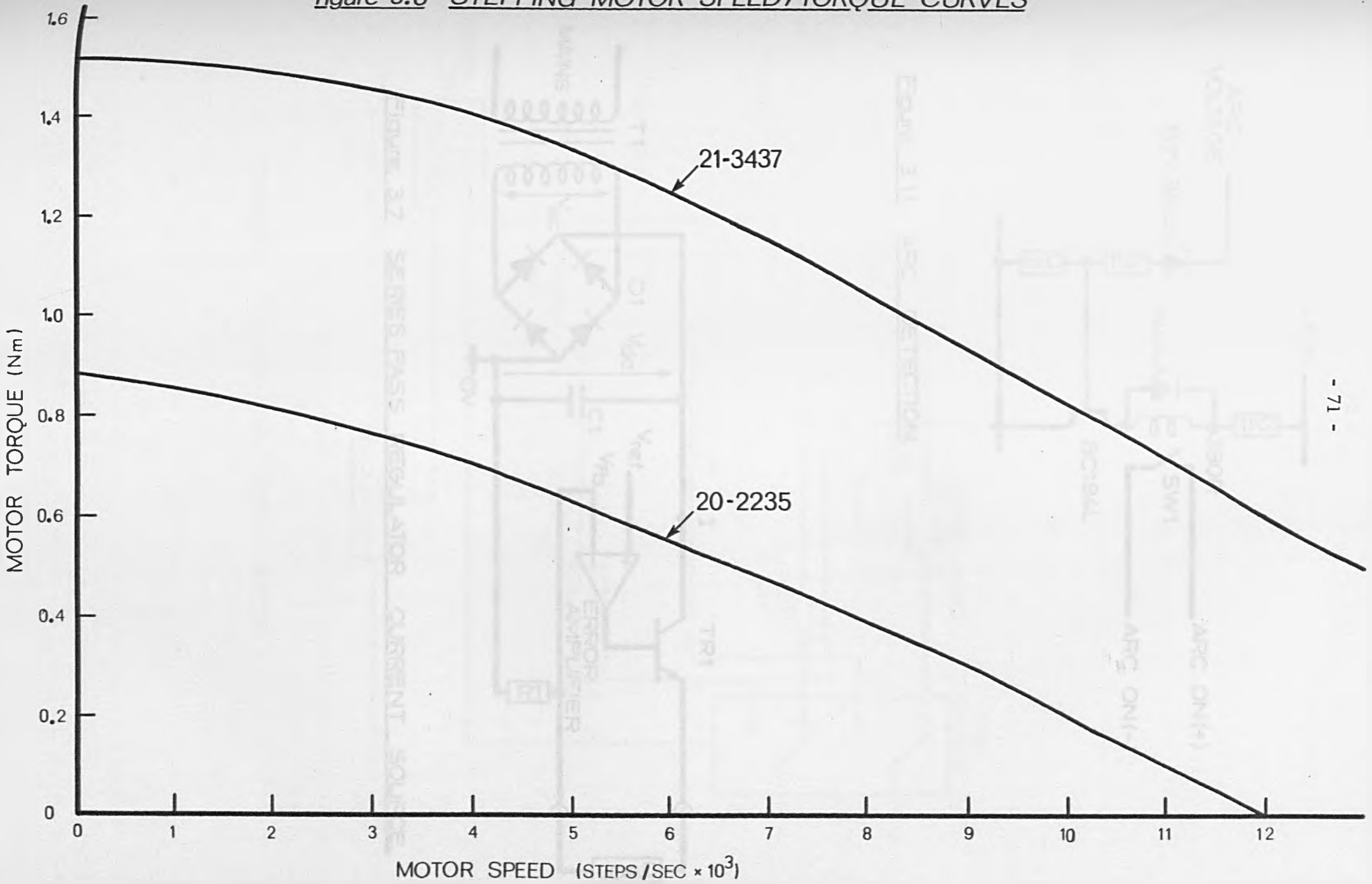


figure 3.5 WRIST MOTOR ANGLES

figure 3.6 *STEPPING MOTOR SPEED/TORQUE CURVES*



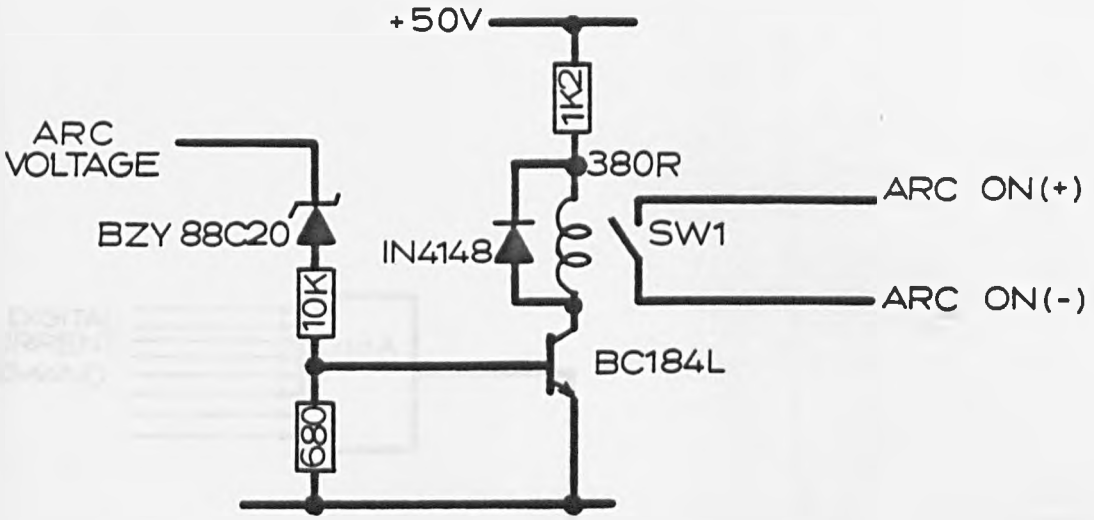


Figure. 3.11. ARC DETECTION.

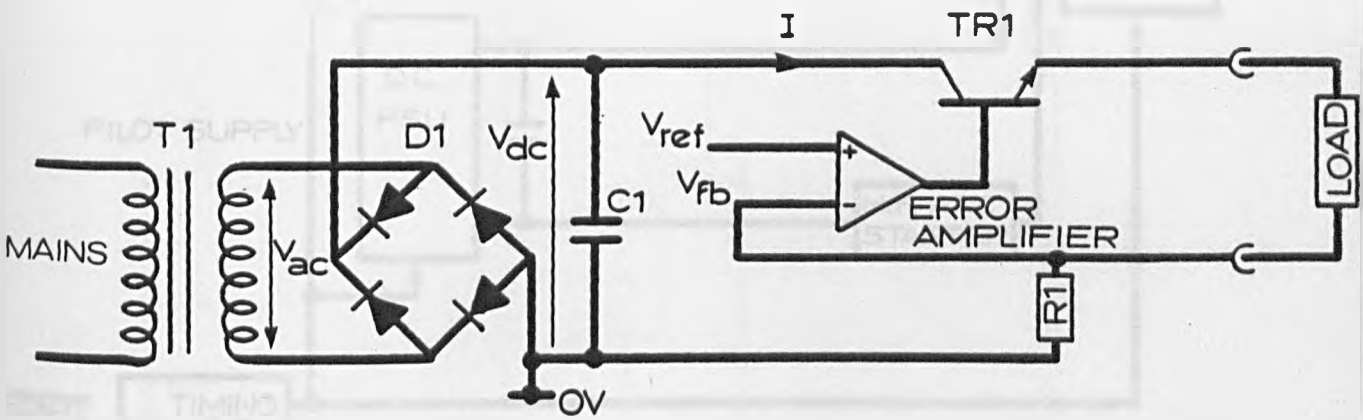


Figure. 3.7 SERIES PASS REGULATOR CURRENT SOURCE.

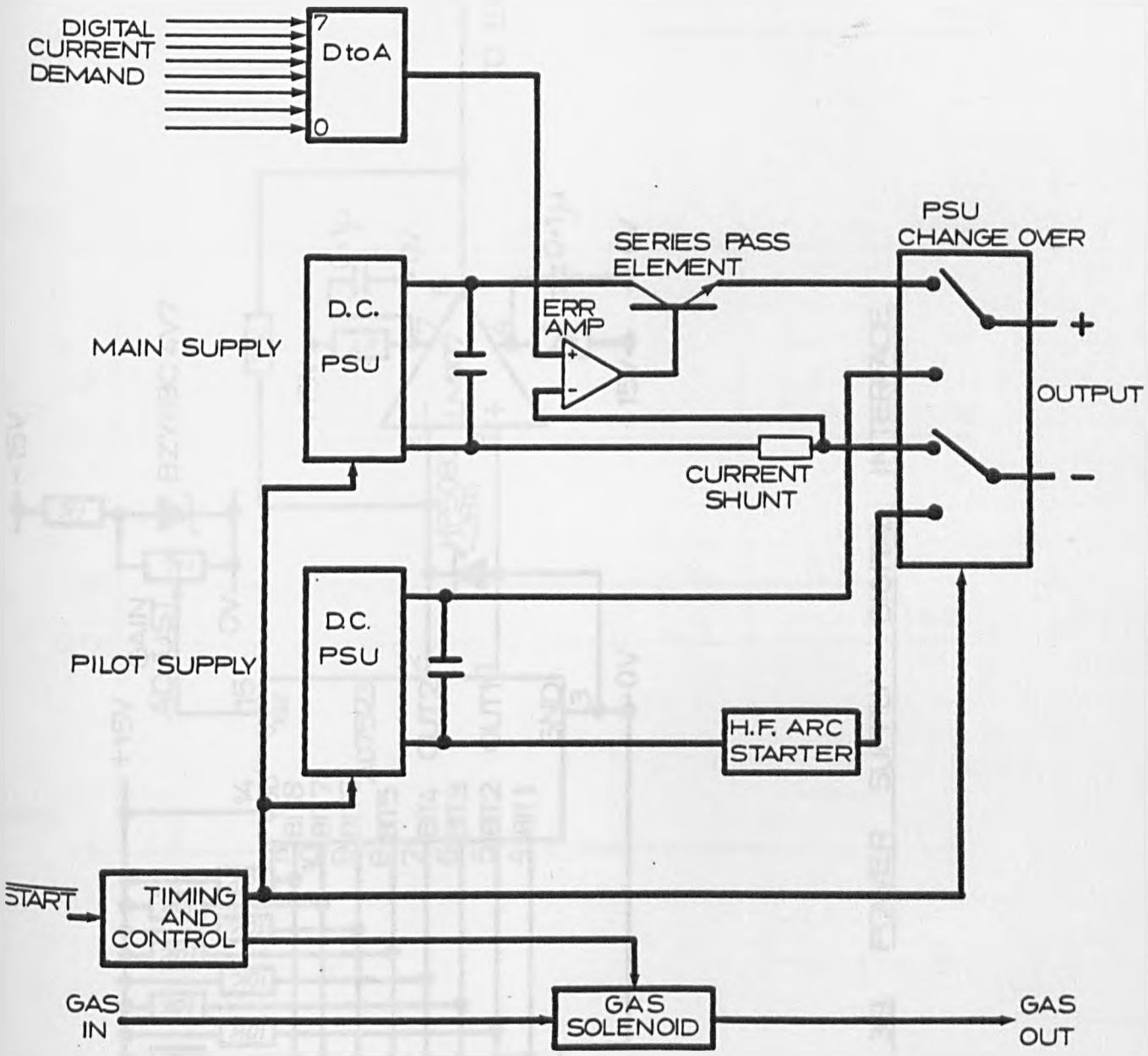


Figure. 3·8. TIG WELDING POWER SUPPLY.

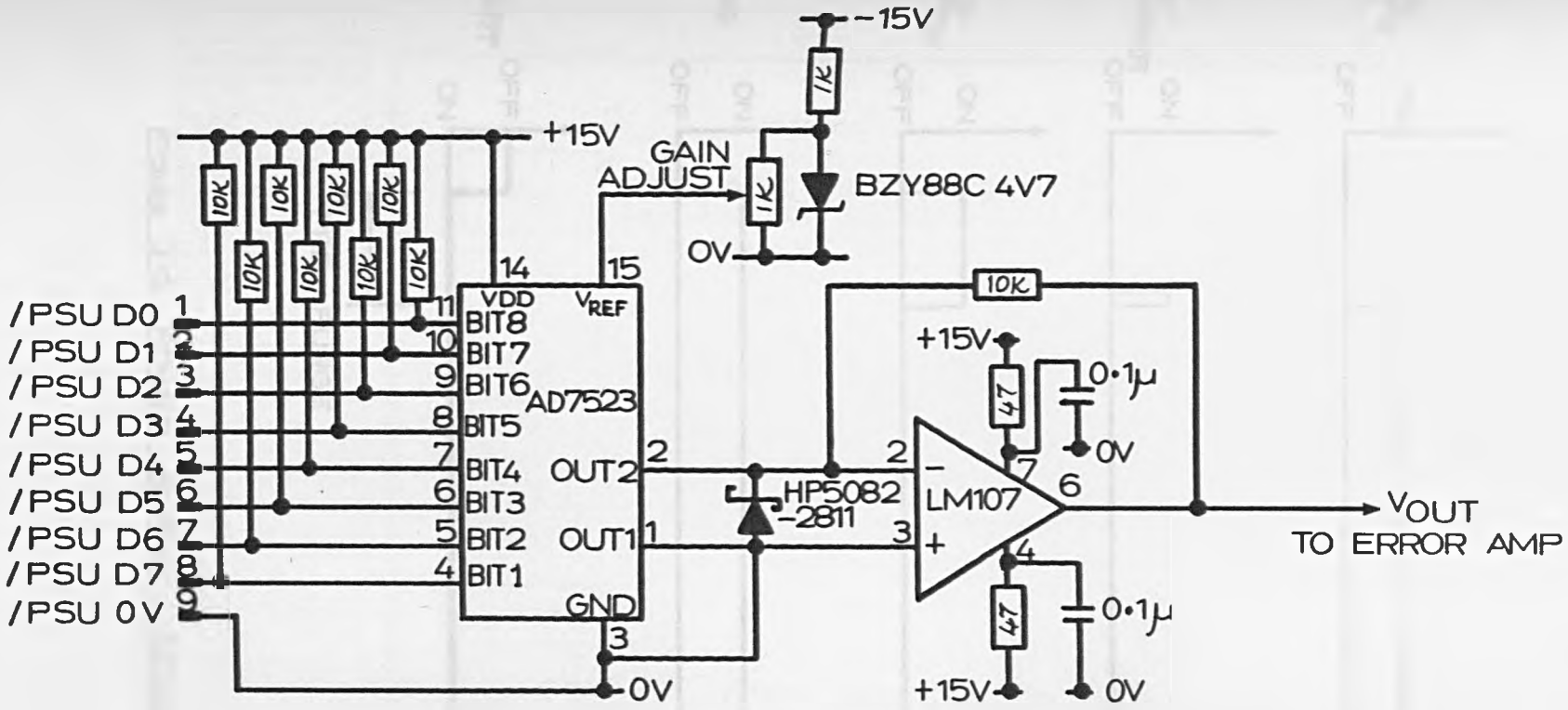


Figure. 3·9. POWER SUPPLY DIGITAL INTERFACE

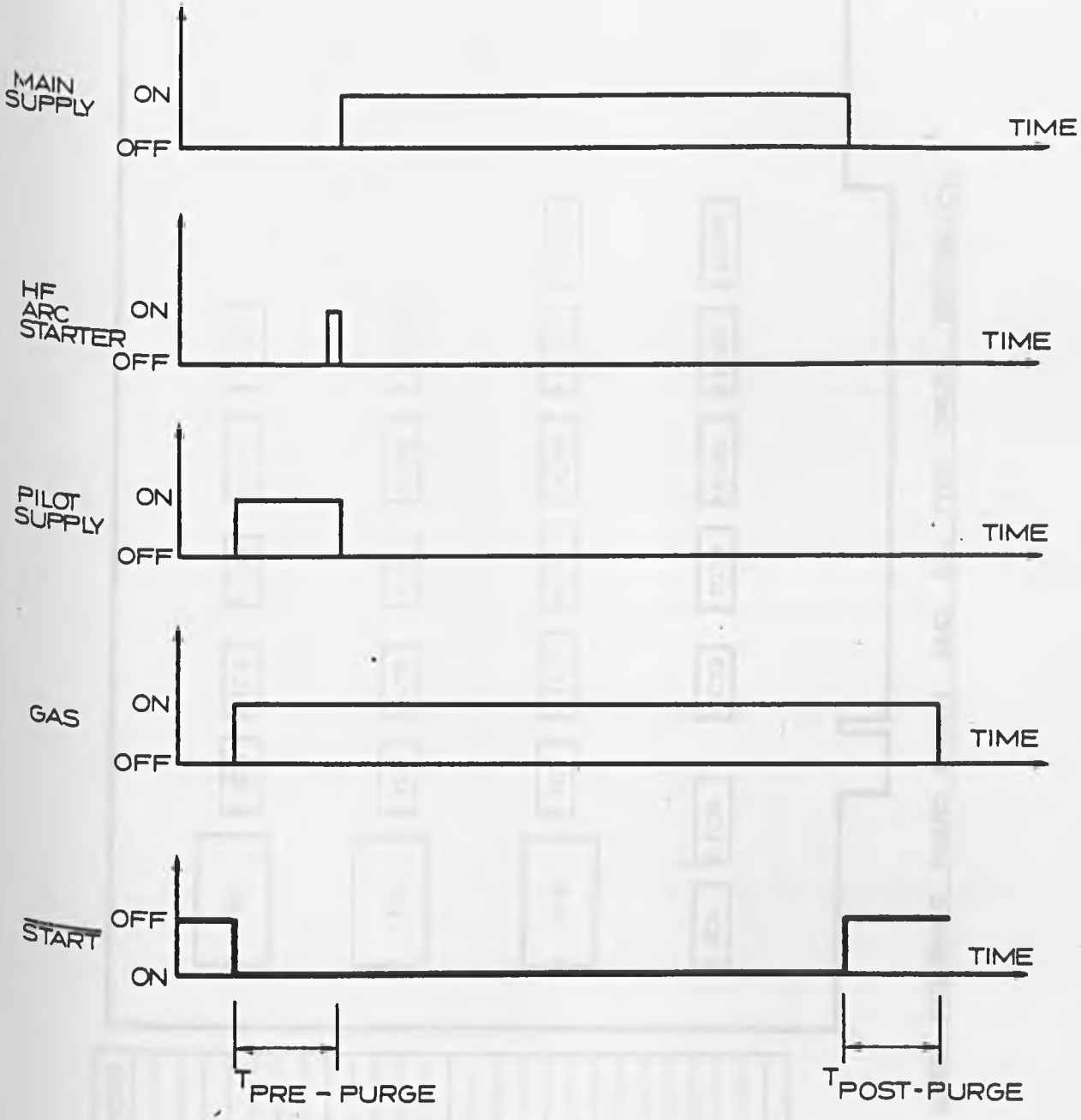


Figure 3·10. POWER SUPPLY SEQUENCING

INTERGRATED REF	CIRCUIT REFERENCES TYPE
IC1	8253
IC2	8253
IC3	8253
IC4	74LS75
IC5	7416
IC6	74LS02
IC7	74LS04
IC8	74LS75
IC9	74LS32
IC10	74LS02
IC11	74LS138
IC12	7416
IC13	7416
IC14	7416
IC15	ILQ74
IC16	ILQ74
IC17	74LS75
IC18	74LS75
IC19	ILQ74
IC20	ILQ74
IC21	74LS75
IC22	74LS365
IC23	ILQ74
IC24	74LS75
IC25	ILQ74
IC26	7416

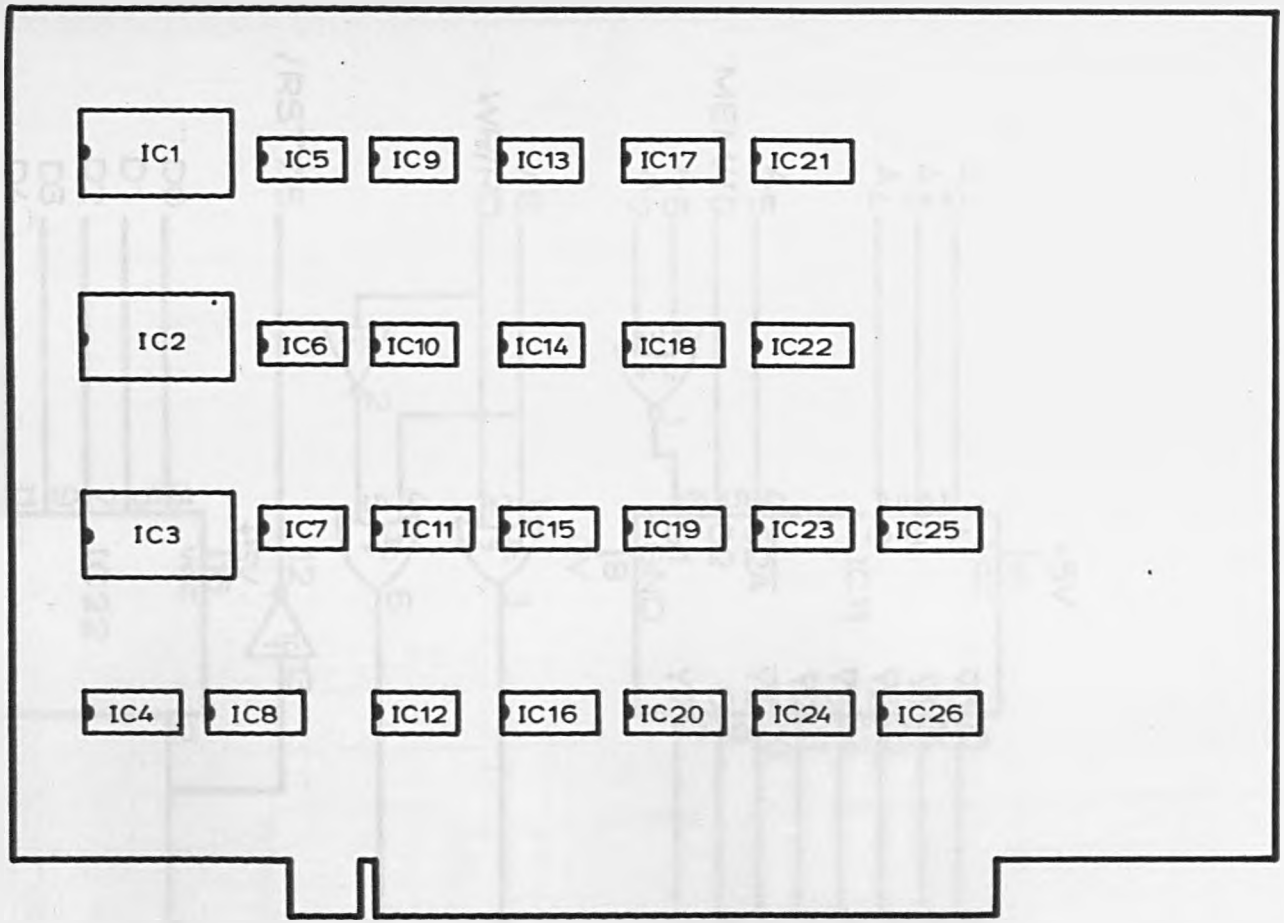


Figure. 3·12. USER INTERFACE BOARD LAYOUT AND I.C. TYPE CROSS REFERENCE.

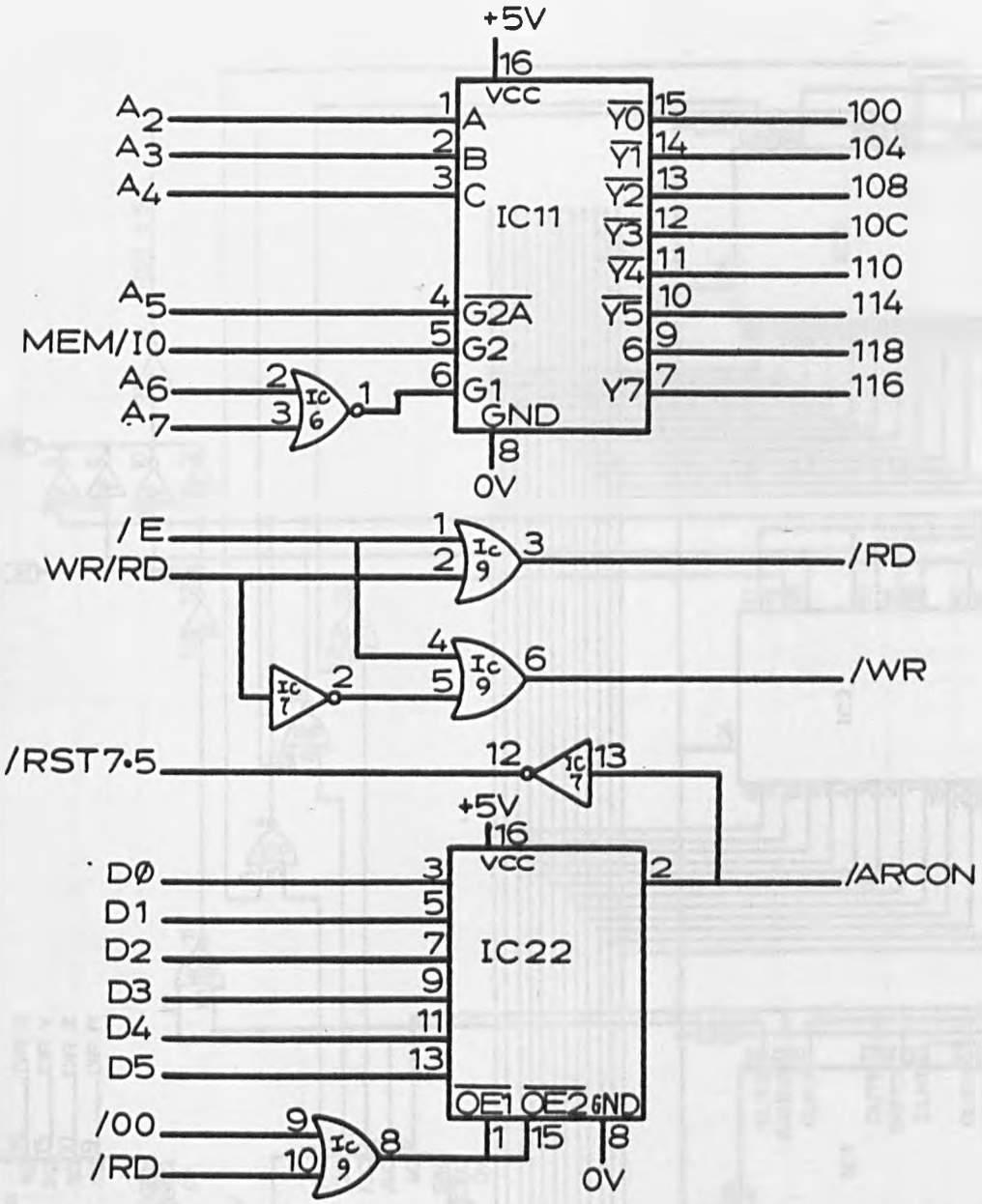


Figure. 3·13. ADDRESS DECODING AND DIGITAL INPUT.

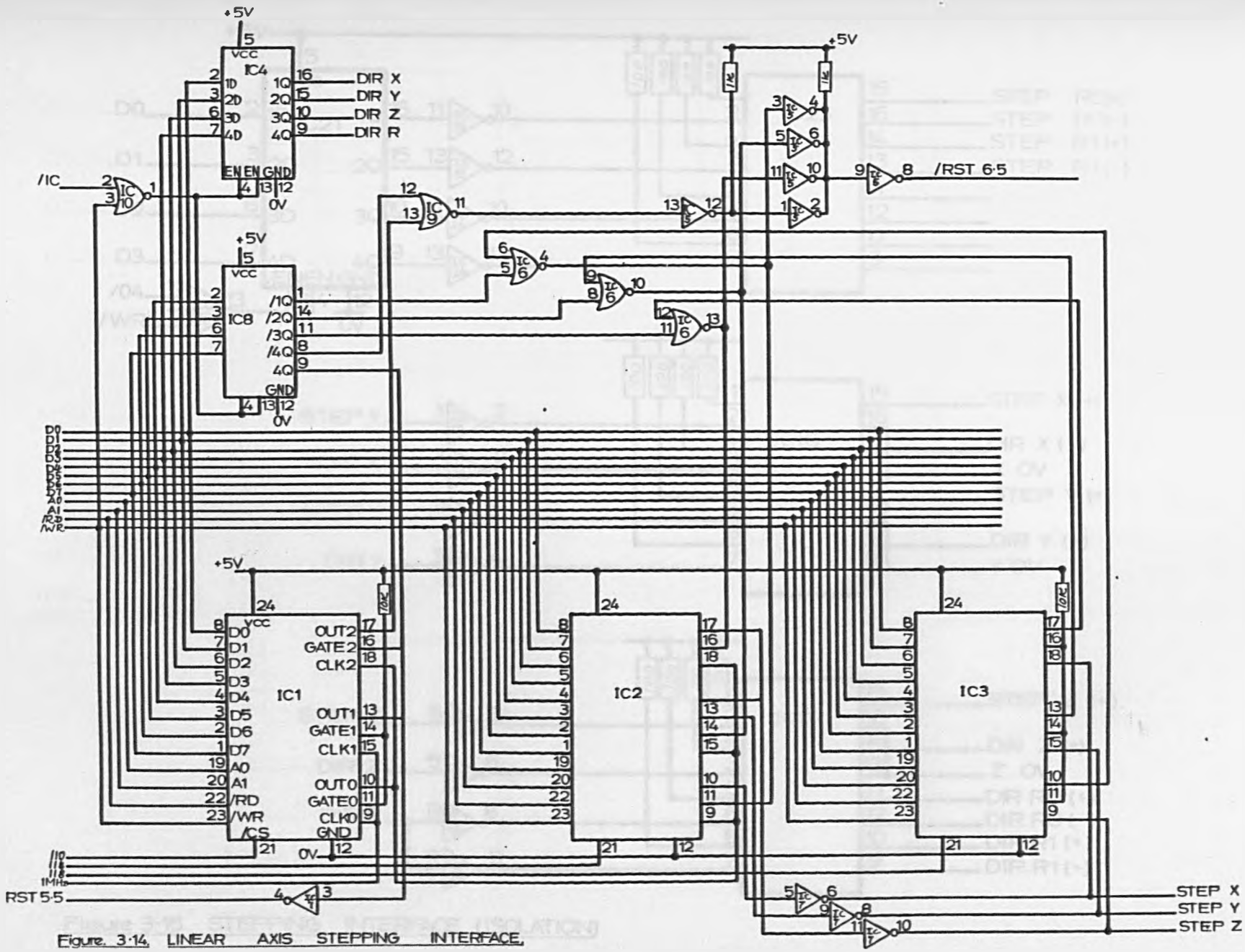


Figure. 3·14. LINEAR AXIS STEPPING INTERFACE.

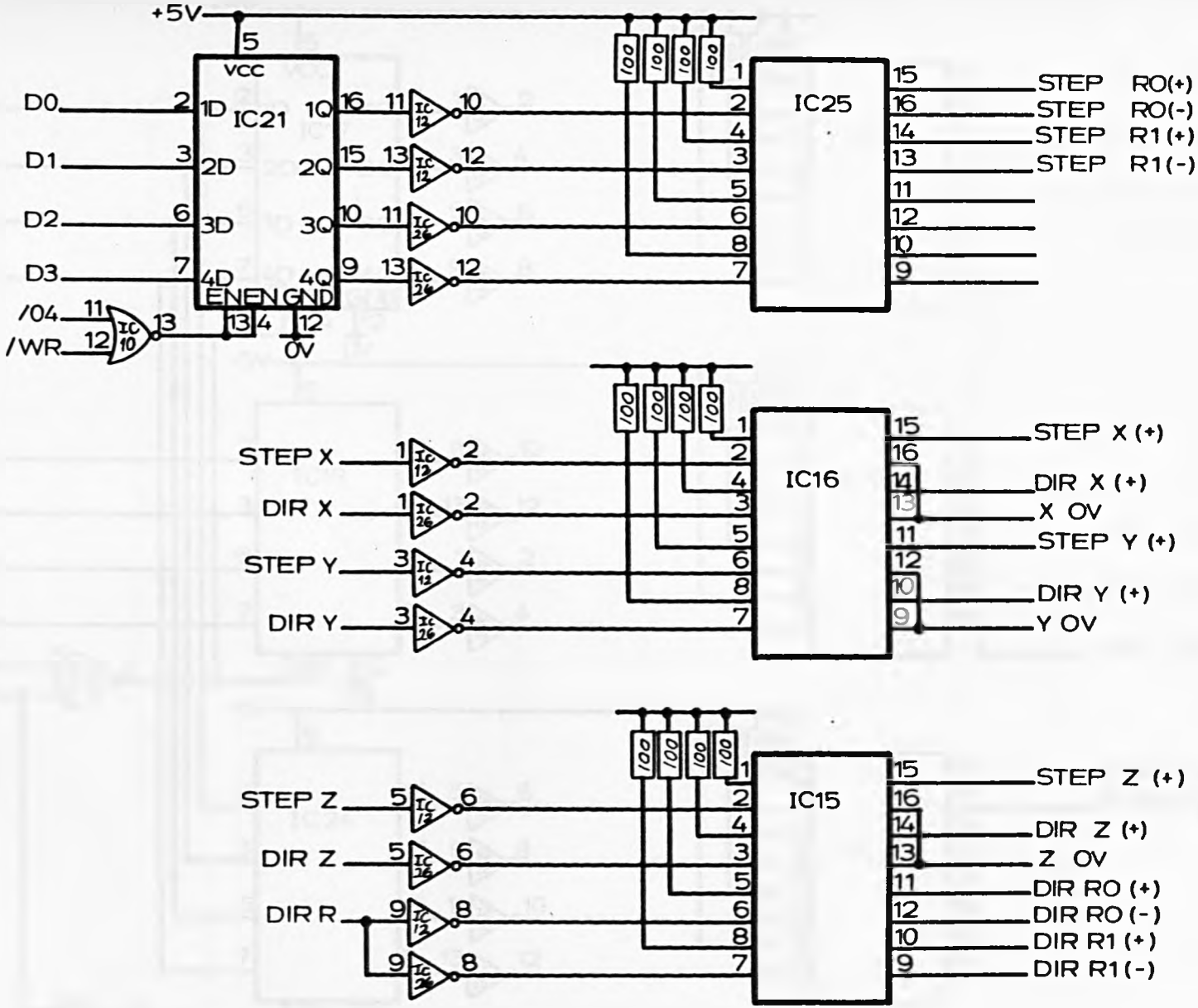


Figure 3·15 STEPPING INTERFACE (ISOLATION)

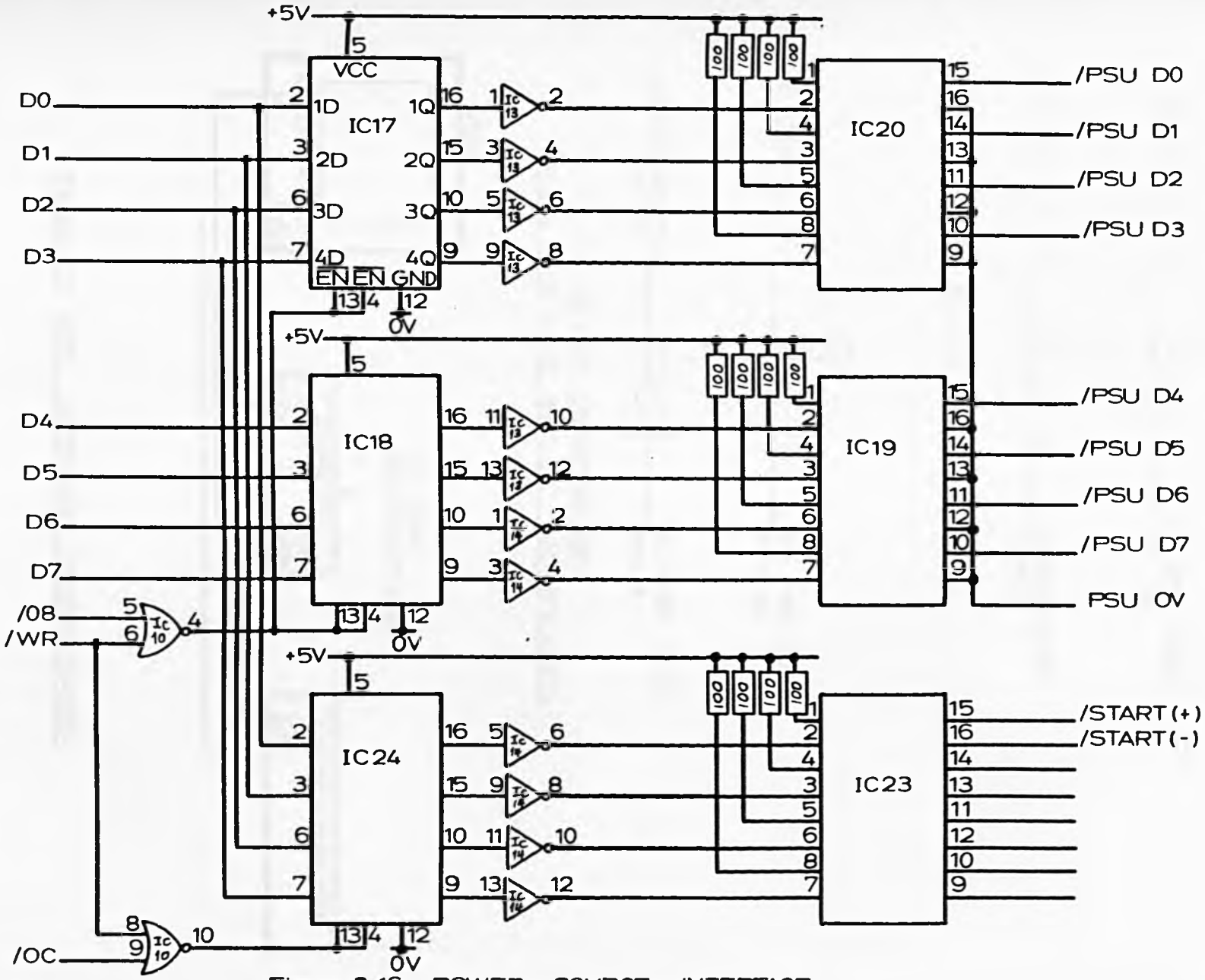
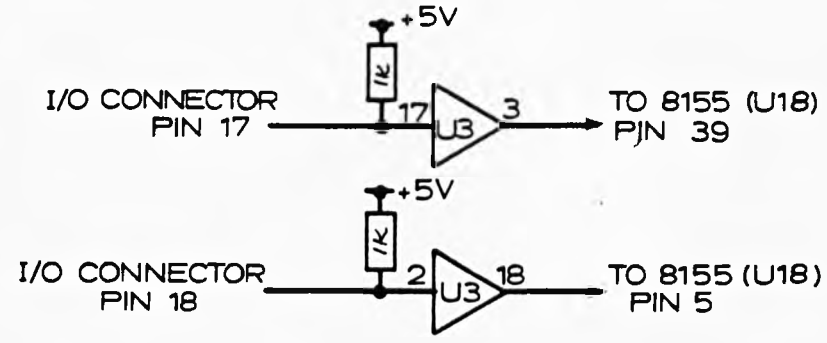
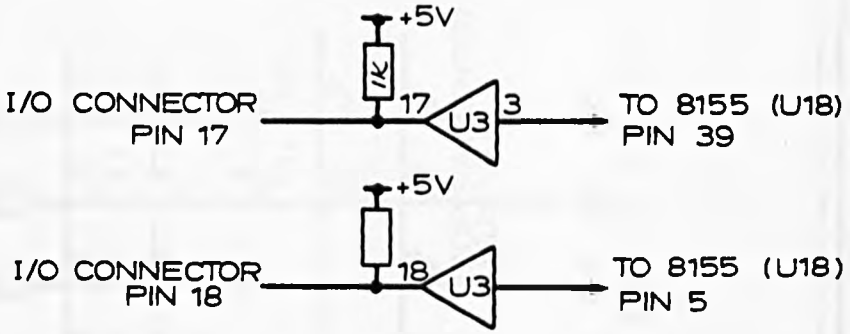


Figure 3·16 POWER SOURCE INTERFACE



(a) BEFORE MODIFICATION.



(b) AFTER MODIFICATION

Figure. 3·18. QMS 8511 I/O MODIFICATION

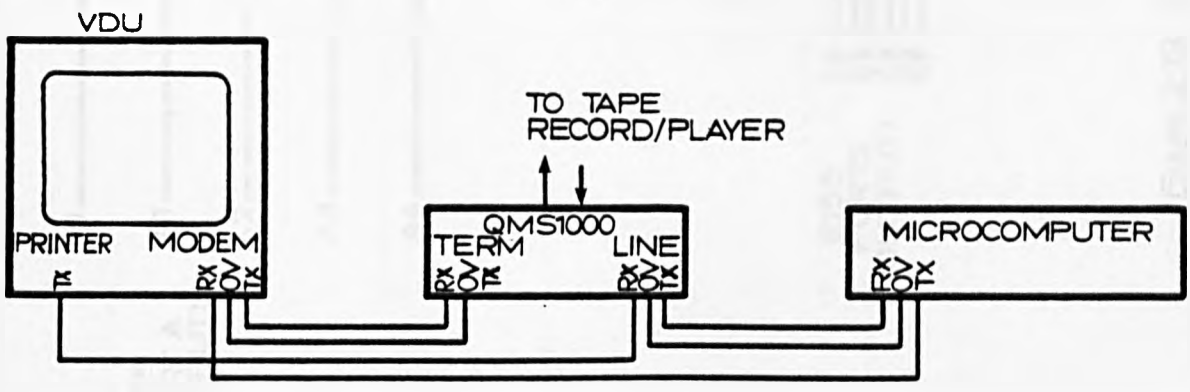


Figure. 3·17 TAPE INTERFACE CONFIGURATION.

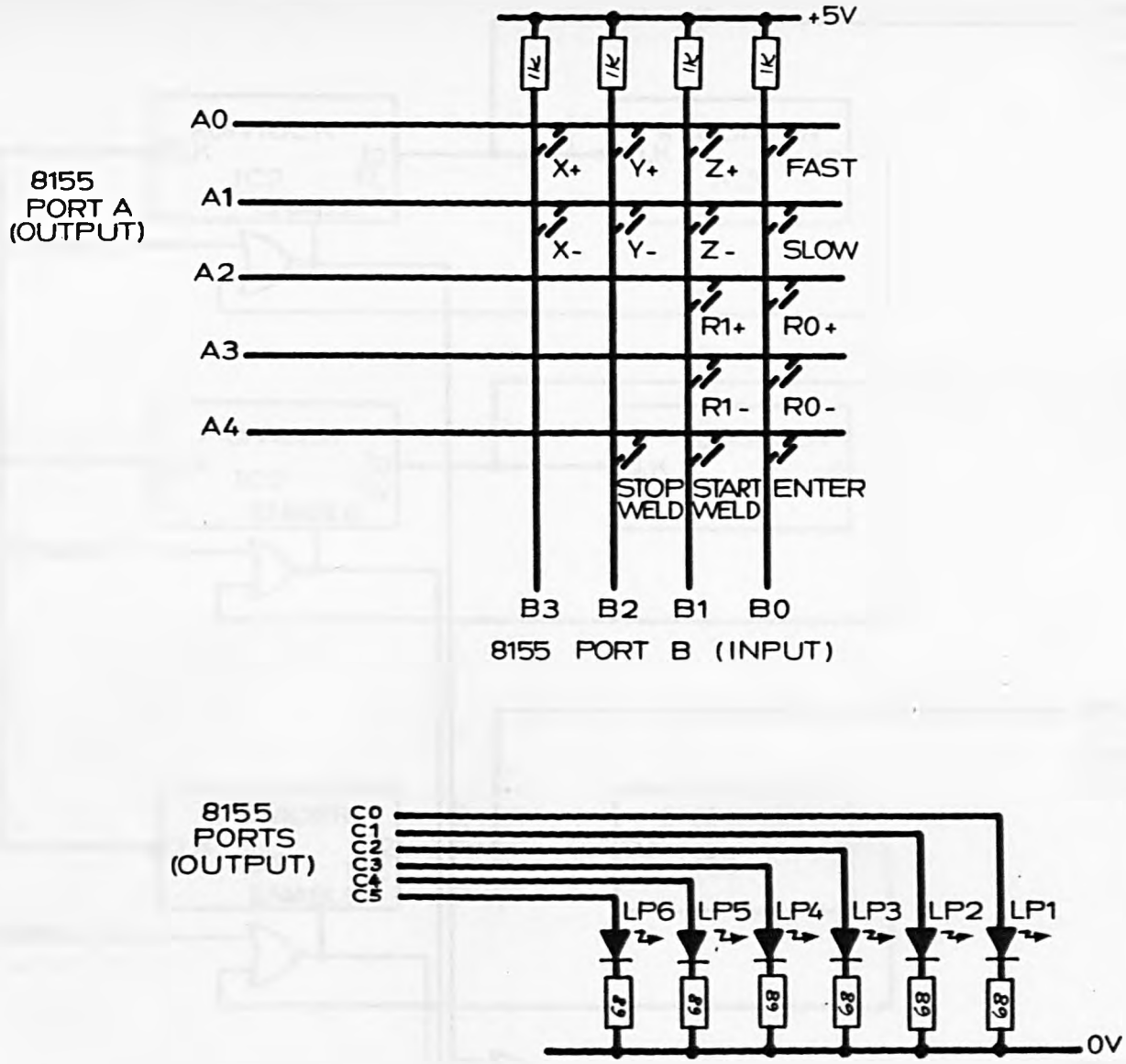


Figure. 3·19. PENDANT KEYPAD SCHEMATIC.

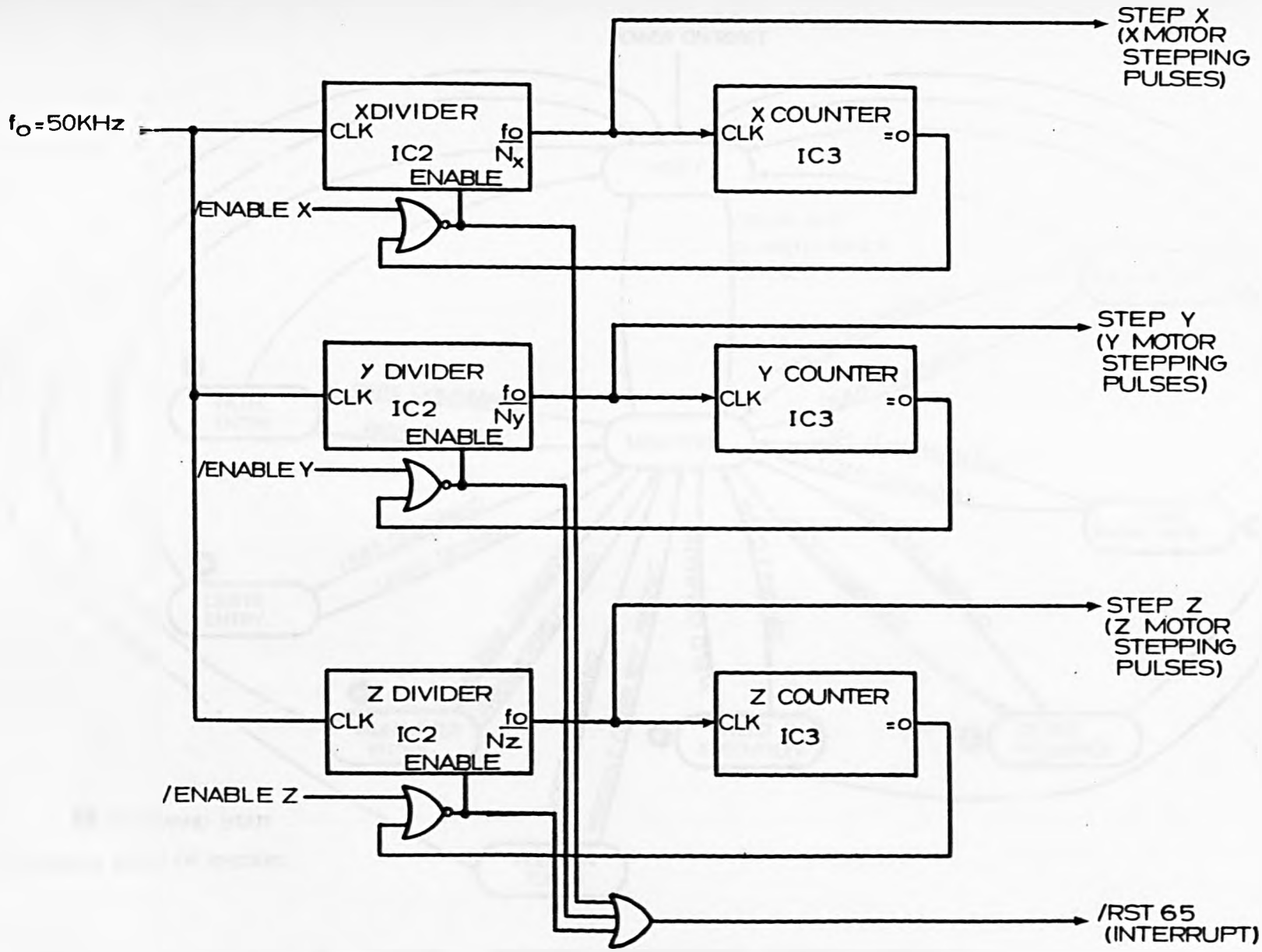


Figure. 3·20. LINEAR AXES STEPPING LOGIC.

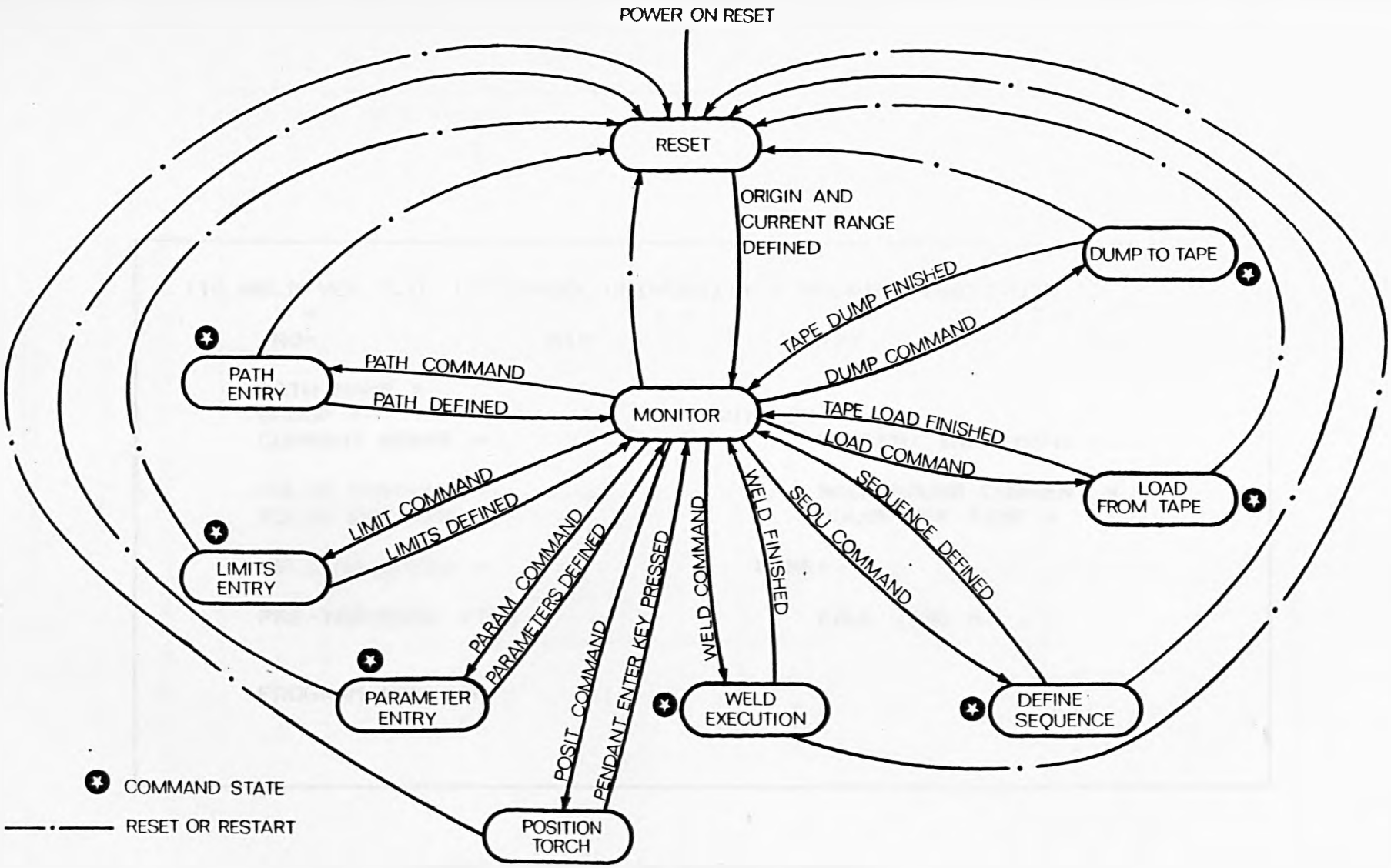


figure 3.21 SYSTEM SOFTWARE STATE DIAGRAM

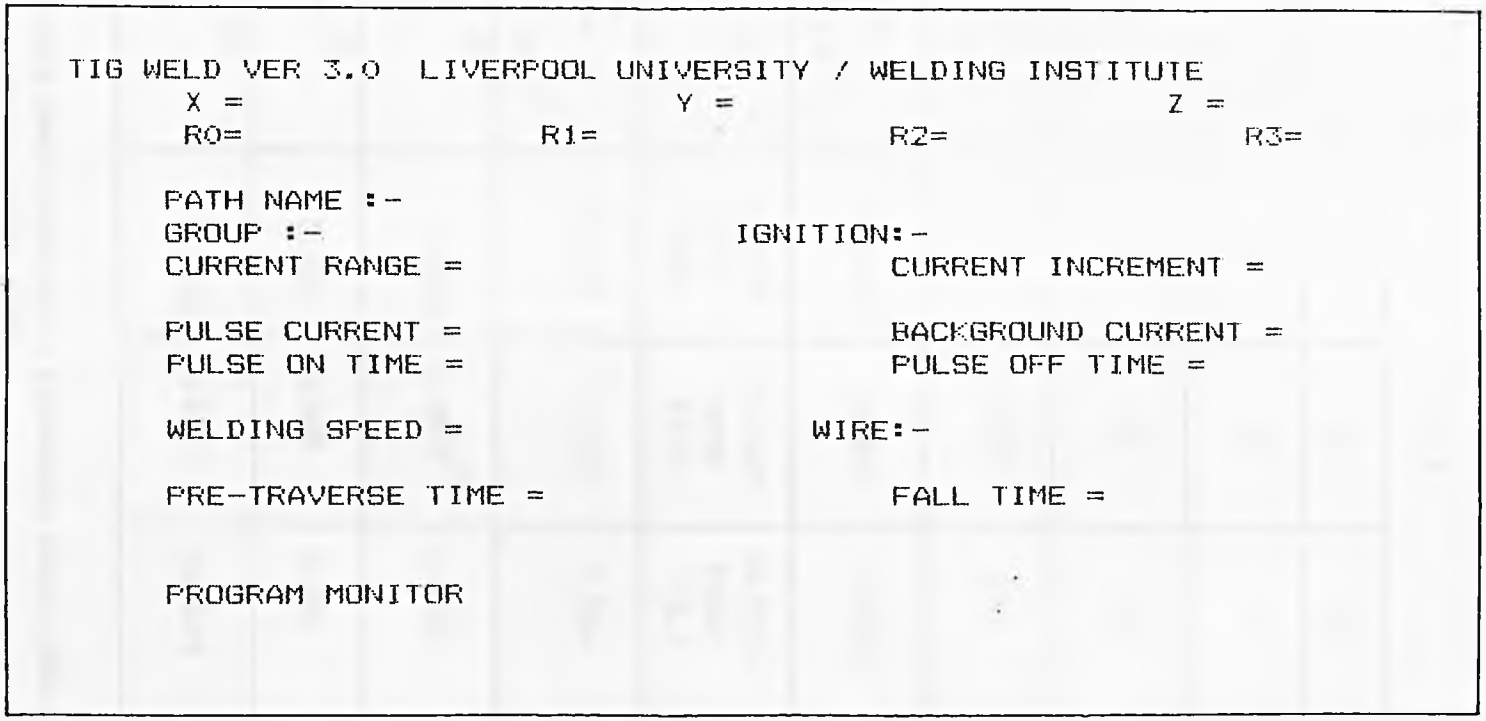


FIGURE 3.22 VDU SCREEN LAYOUT

		AXIS		
		X	Y	Z
LEAD SCREW	Lead 10^{-3}m	4	5	4
	Diameter 10^{-3}m	6	16	6
	Length 10^{-3}m	1000	1000	250
	Inertia 10^{-3}Kg m^2	0.0010	0.0510	0.0003
MOTOR	Type(1)	21.3437 D200 F075	21.3437 D200 F075	20.2235 D200 F3.7
	Rotor inertia 10^{-3} Kg m^2	.0.119	0.119	0.026
Load Inertia(2) 10^{-3} Kg m^2		0.0033	0.0675(3)	0.0035
Static Torque(2) Nm		0.000	0.000	0.054
Total Inertia(2) 10^{-3} Kg m^2		0.1233	0.2375	0.0298

1. Sigma Instruments Inc., Braintree, Mass., USA.
2. Referred to Motor Shaft
3. Workpiece mass = 75 kg

Table 3.1 : Linear Axes Parameters

	AXIS		
	X	Y	Z
REQUIRED TORQUE Nm	0.48	0.60	0.17
ACTUAL TORQUE Nm	1.50	1.50	0.85
MAXIMUM START/STOP SPEED $10^{-3}ms^{-1}$	18	15	27

Table 3.2 Linear Axes Performance

	AXIS				
	X	Y	Z	θ	ϕ
RANGE	0.5m	0.6m	0.2m	180^{o}	360^{o}
RESOLUTION	$1x10^{-5}m$	$1.25x10^{-5}m$	$1x10^{-5}m$		

Table 3.3 Robot Working Envelope

CHAPTER 4

TIM 3 WELDS

This chapter describes the welding tests carried out in order to assess the ability of the TIM3 welding robot.

The TIM 3 system is an open-loop welding system, i.e. the weld is carried out using pre-set values of speed and current. Here it is necessary to know the required parameter values when programming the weld. The main application of the TIM3 robot is envisaged as being the welding of stainless steel components. Hence a series of weld tests were carried out using the TIM 3 robot both at Liverpool University and The Welding Institute, in order to determine empirically the required parameters for various thicknesses of stainless steel material.

These tests consisted of executing a straight line 'melt-run' along a stainless steel plate, according to a set of weld parameters. The resulting 'seam' was then examined and the parameters judged as acceptable or not depending on the quality of the weld. Particular attention was given to the weld penetration. Lack of penetration being signified by the failure of the weld pool to extend through the plate to the reverse side throughout the seam length. Over-penetration showed up as bulging beads or even blow-through holes on the reverse side. Figure 4.1 shows a good weld both from the front and reverse side. In order to prevent the reverse side of the test plate from oxidising during welding a backing gas of argon was applied. This was achieved by bolting the test plate to the robot worktable using 10 mm x 10 mm steel strip as spacers between worktable and plate. Argon was passed through the resulting

space between plate and worktable.

Figures 4.2, 4.3 and 4.4 summarise the results using pulse currents of 50A, 100A and 150A respectively. The background current was in all cases 10A and the shield gas was 95% argon/ 5% hydrogen. The extended currents (100A and 150A) were obtained using an uprated Polypack welding supply at The Welding Institute.

The outlined areas in figures 4.2, 4.3 and 4.4 show the range of parameters which give acceptable welds for the corresponding material thickness. The best set of parameters may be taken as those given at the geometrical centre of the figure. For instance in figure 4.2 the optimal parameters for welding 1.0 mm material (I_P = 50A, I_B = 10A) are TP = 1.5S and V_T = 3.5 mm/S. From the results suitable parameters may be chosen for any material thickness from 0.8 mm to 3.5 mm.

Figure 4.5 shows three geometrical melt run patterns. This is included to indicate the degree of control that may be obtained using point by point path entry.

Figures 4.6 and 4.7 show tube to plate welds, both before and after welding, carried out on the TIM 3 robot. Here a tubular section is welded around its end to a plate through which it passes. For figure 4.6 the tube thickness is 1 mm and for 4.7 it is 1.5 mm. The weld parameters were chosen from figure 4.2.

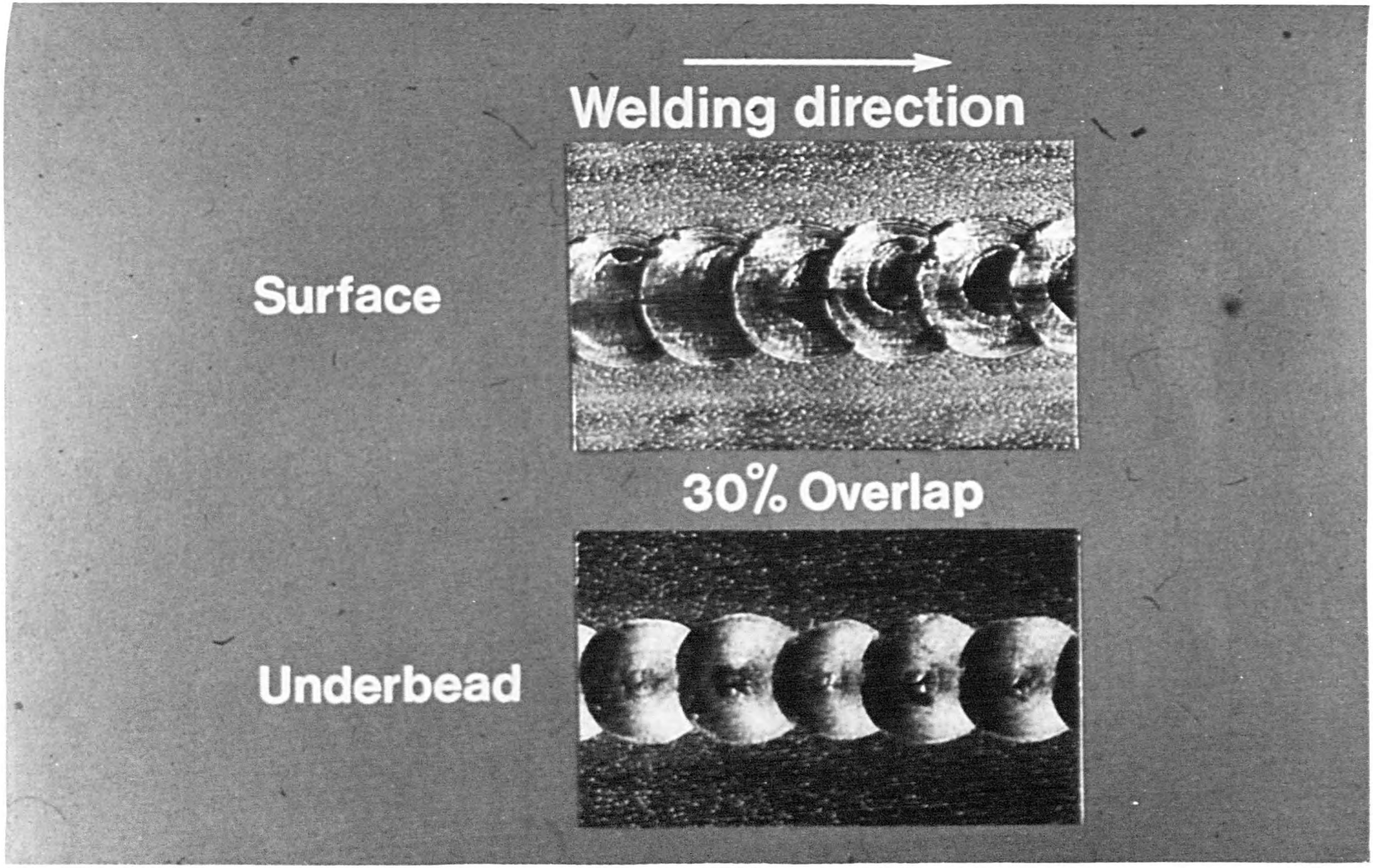


FIGURE 4.1 A PULSED TIG WELD

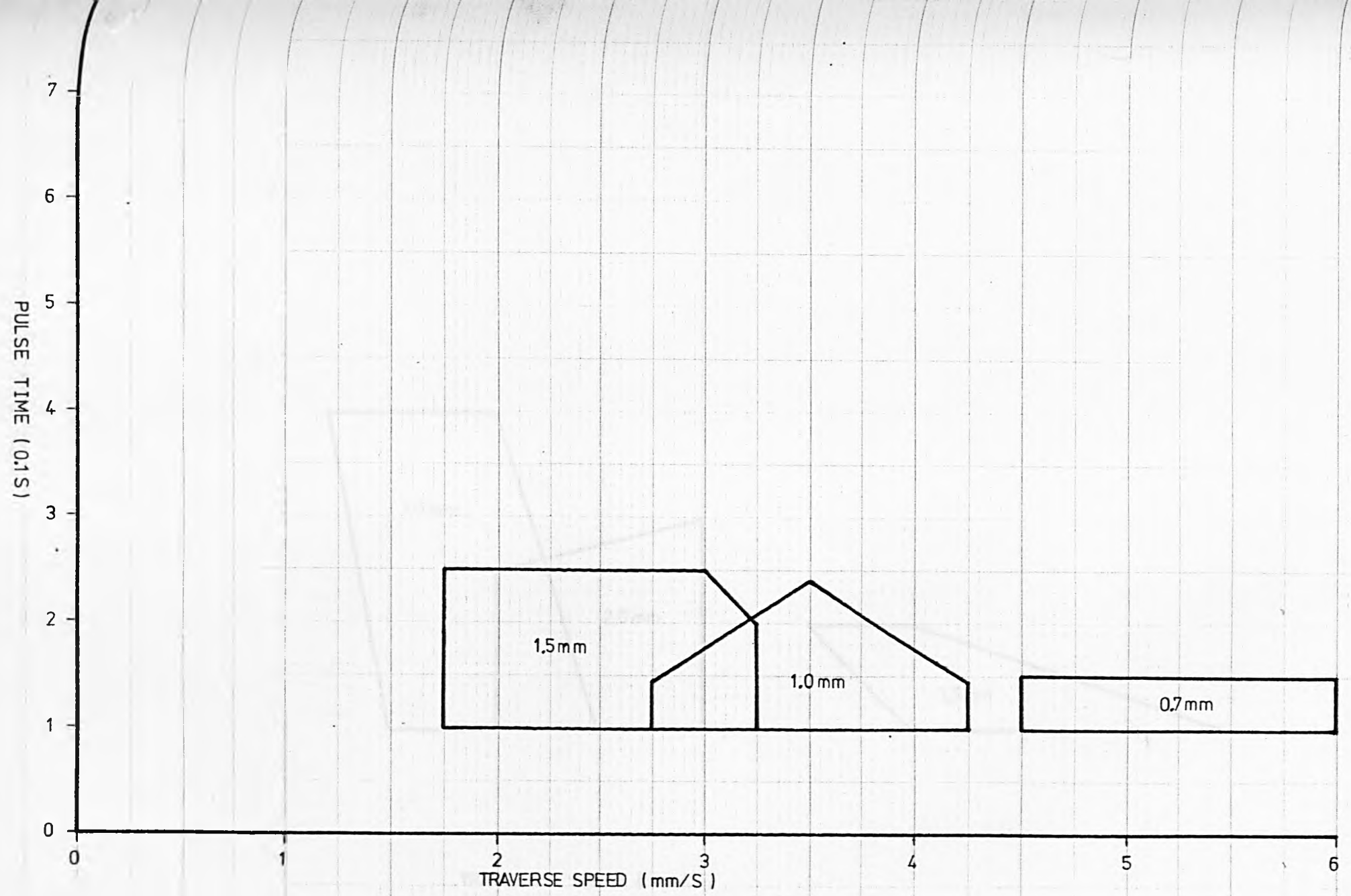


FIGURE 4.2 50 AMP WELD PARAMETERS

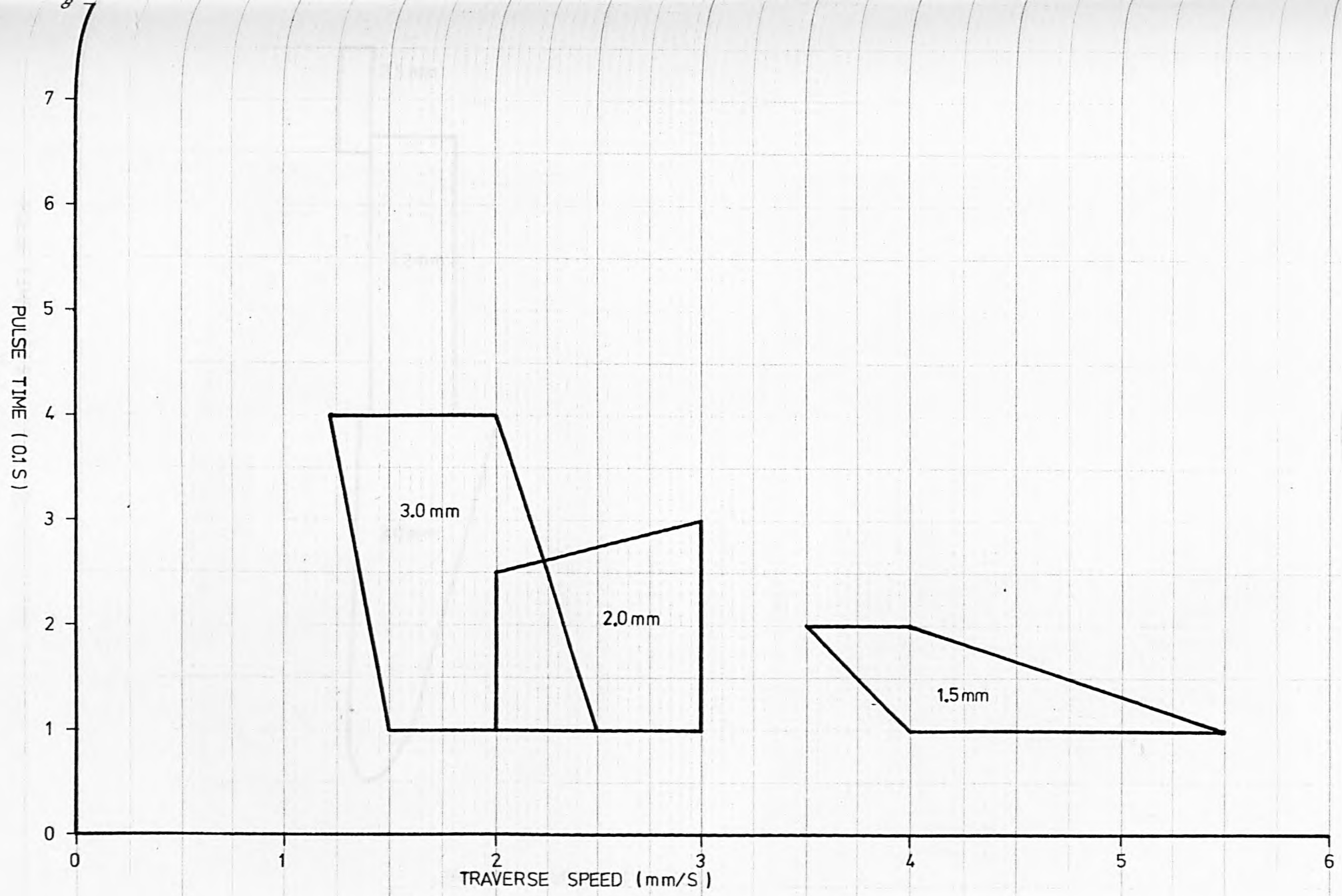


FIGURE 4.3 100 AMP WELD PARAMETERS

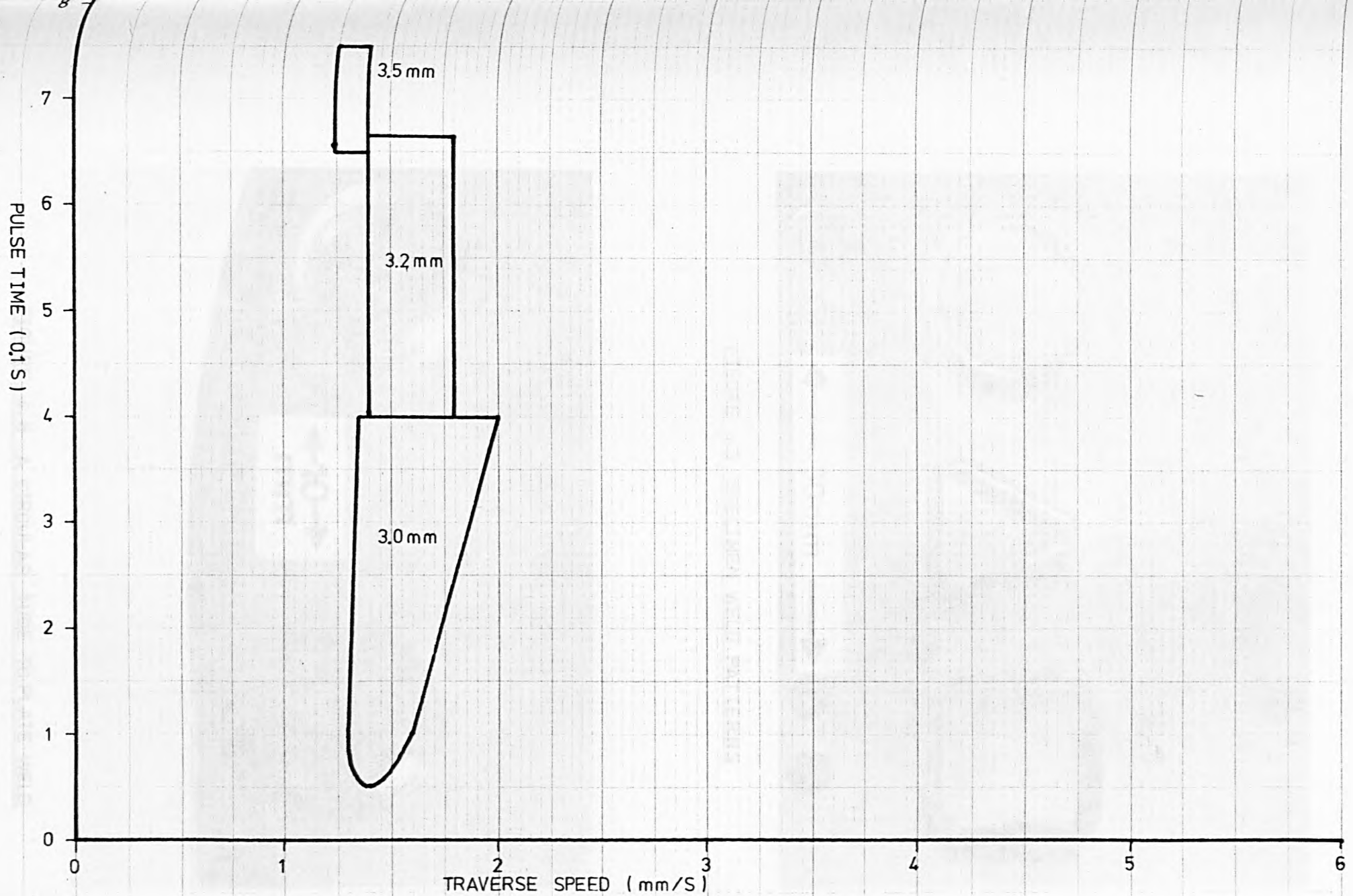


FIGURE 4.4 150 AMP WELD PARAMETERS

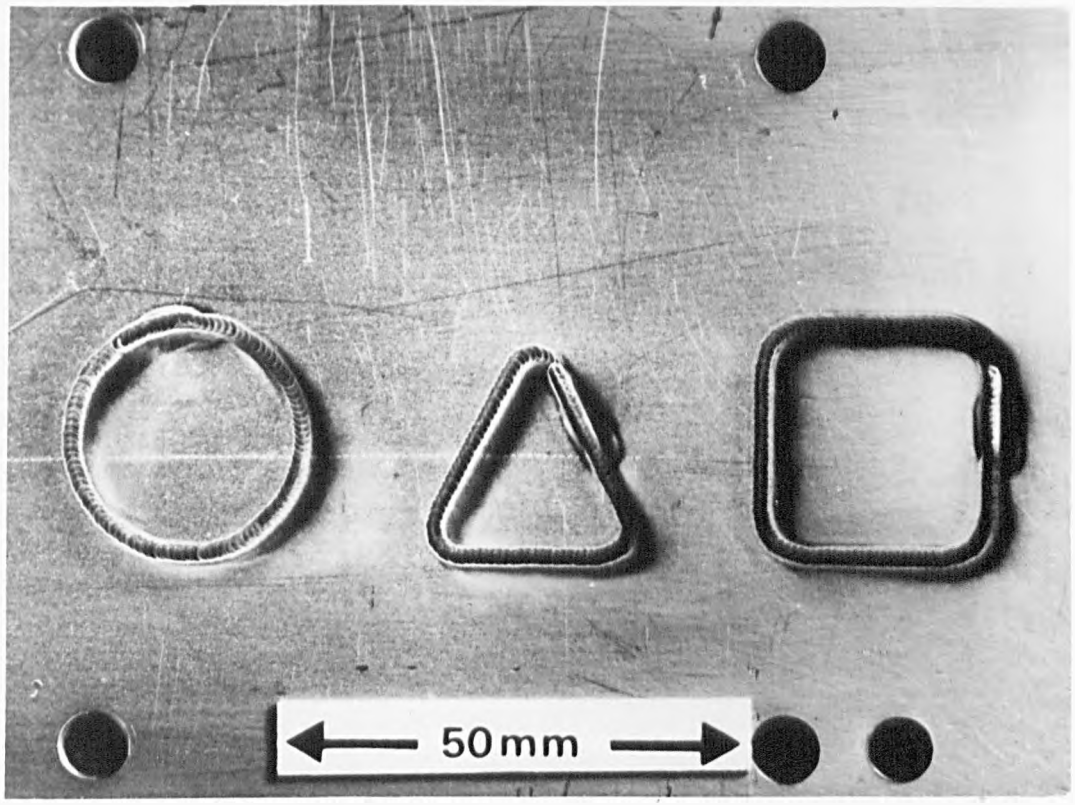


FIGURE 4.5 SPECIMEN WELD PATTERNS

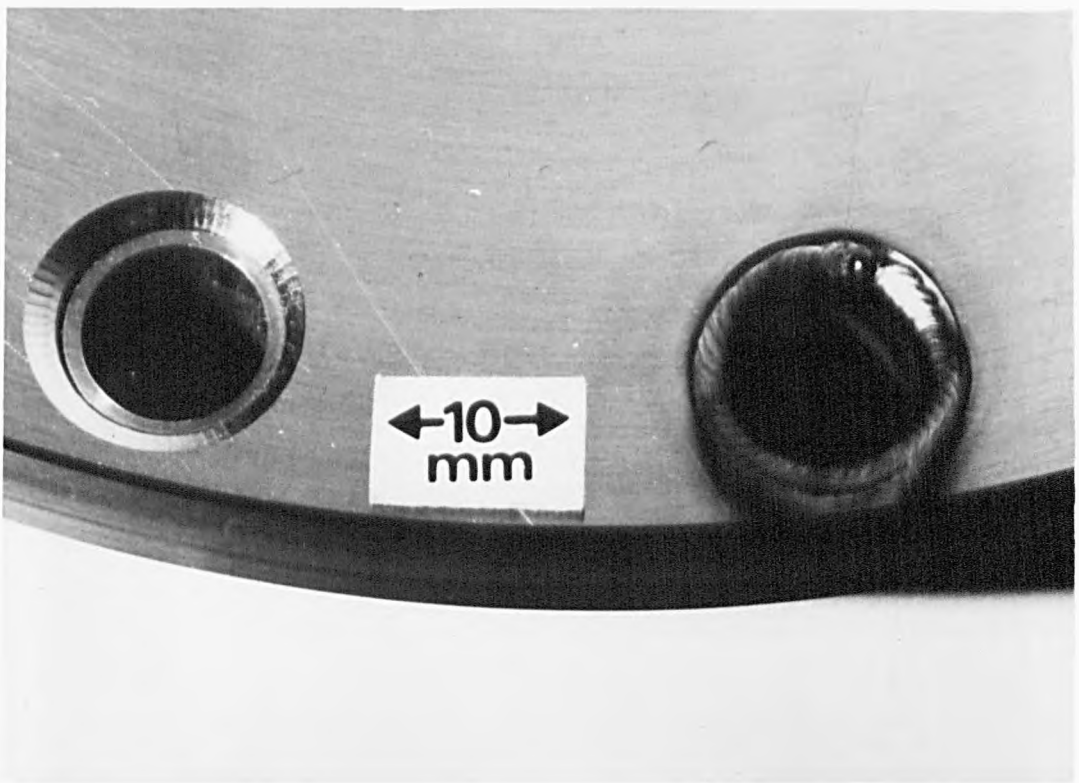
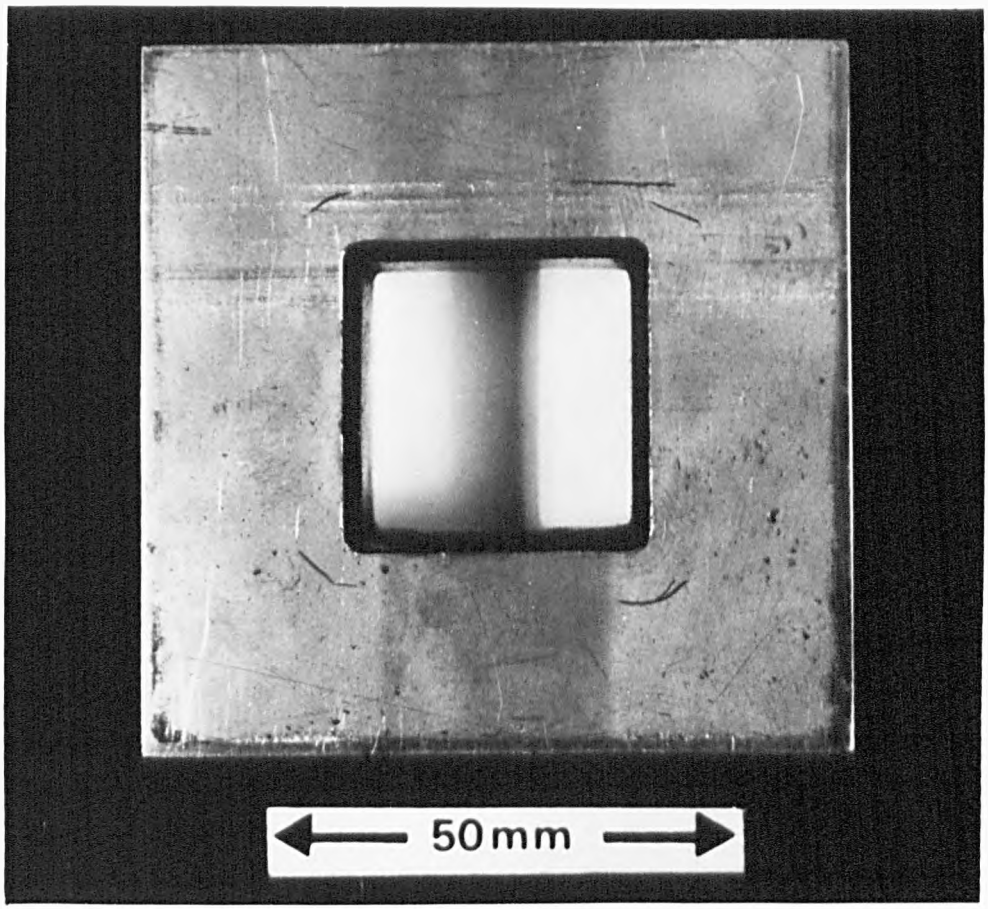
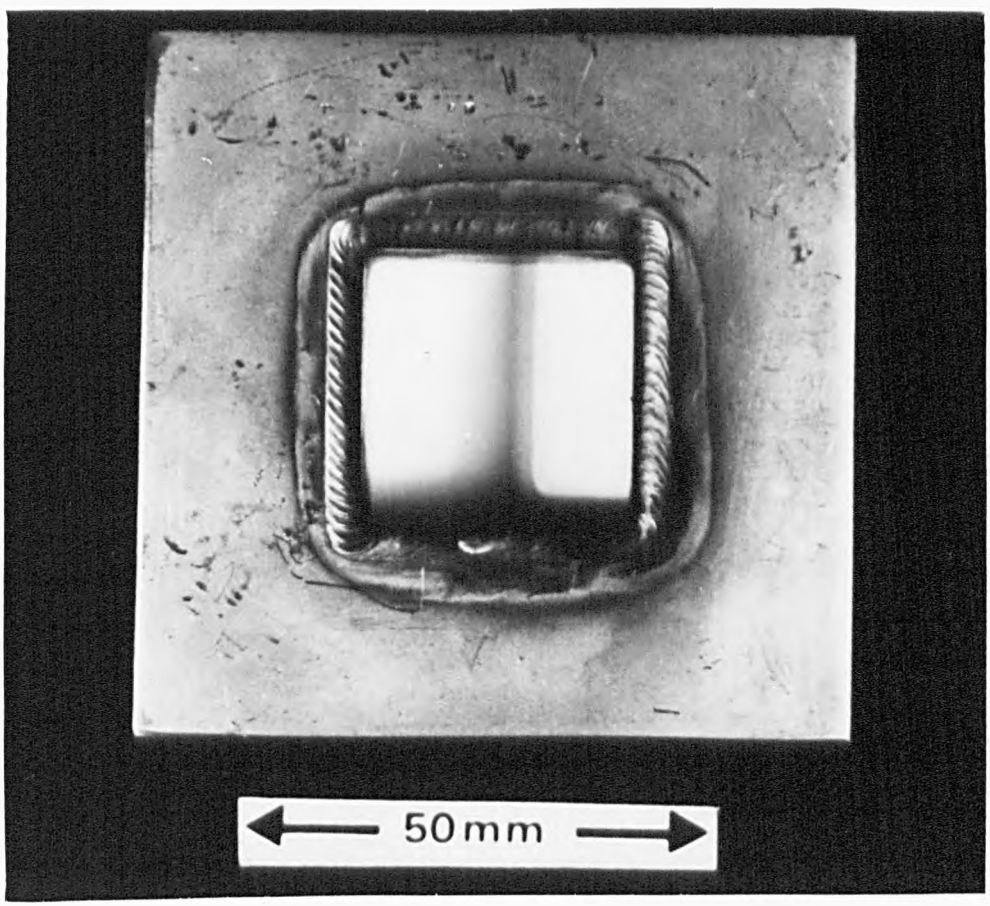


FIGURE 4.6 A CIRCULAR TUBE TO PLATE WELD



(a) before welding



(b) after welding

FIGURE 4.7 RECTANGULAR TUBE TO PLATE WELD

CHAPTER 5

CONCLUSIONS

This research project has demonstrated that a cartesian robot may achieve the required degree of accuracy for carrying out TIG welding. This has been proved both by 'calibration' measurements and by the execution of various welding tasks.

By performing various intricate welds the technique of linearly-interpolated three-dimensional space curves has been shown suitable for the representation of weld seams. A further method of representing space curves, i.e. the method of cubic splines, has been presented which has the prospect of reducing the number of points required to define a curve by a factor of at least an order of magnitude.

The project has proven that currently available micro-computers are suitable for the control of TIG welding. The computing power of the 8085 system used was sufficient to provide simultaneous control of arc current and torch position and speed. Also the controller has demonstrated its ability to operate reliably under the severe electrical disturbances associated with h.f. arc starting.

The robot developed is of the open loop type. Here welding is carried out according to a pre-defined path and with preset welding parameters. In order to reduce the failure rate due to component variations closed loop systems are beginning to be investigated.[23]

A closed loop system is one where the weld seam is monitored and the results used to control the welding process. Both the weld position and quality may be sensed. The great difficulty in producing a closed loop system

would seem to be in obtaining sensors that can produce useful information and cope with the wide variety of seam types and surface finishes.

At present it would seem to me that the ultimate welding system will be a hybrid open/closed loop system. Here the weld path and parameters would be pre-defined, but sensors would be used to compensate for variations. The system could switch between the closed and open loop modes during operation. In particular the system would revert to open loop mode whenever situations were encountered where the sensors might prove unreliable. This open loop mode could use the pre-stored data modified according to previous sensor readings. Also comparison between the pre-stored data and sensor readings would give an indication of acceptability and highlight possible malfunctions.

In order to produce a system such as that above a great deal of effort needs to be invested in software. The system can readily be broken down into distinct tasks such as motor control, welding power control, sensor monitoring, mathematical positional translations, etc. This suggests that a real-time-multi-tasking software operating system should be used.[24]. Such an operating system allows tasks to be programmed as separate entities, and for the tasks to be executed apparently simultaneously. Thus the system development may be undertaken task by task. Also several people can be assigned different tasks to develop knowing that at the end of the day these can be brought together, provided that the operating system rules have been adhered to.

REFERENCES

1. SLOAN, K., and LUCAS, J., "Microprocessor control of TIG welding system", Proc. I.E.E., 1981, 129, Pt.E., pp.1-8.

2. LANCASTER, J.F. Metallurgy of Welding, Allen and Unwin, 1980.

3. CARY, H.B., Modern Welding Technology, Prentice Hall, New Jersey, USA., 1979, pp.146.

4. AMIN, A., "Prediction of Square Wave Pulse Current Parameters for Control of Metal Transfer in MIG Welding", Welding Institute, Report PRAD 140/78.

5. Japanese Welding Industry Fact Book 1980/81.

6. CARY, H.B., Modern Welding Technology, Prentice Hall, New Jersey, USA., 1979, pp.95.

7. CARY, H.B. Modern Welding Technology, pp.107.

8. "Developments in Mechanised Automated and Robotic Welding", Int. Conf. of The Welding Institute, London, 1980.

9. PAUL, R.P., Robot Manipulators, MIT Press 1981, Cambridge, Mass.

10. Digiplan Drives, Type 1054. Unimatic Engineers Ltd., Cricklewood, London, NW2 2LN.

11. RYDER, G.H., BENNETT, M.D., Mechanics of Machines, Mamillan Press Limited, 1975.

12. ACARNLEY, P.P., Stepping Motors a Guide to Modern Theory and Practice, I.E.E. Control Engineering Series, 19, pp.40-41.

13. KUO, B.C., Theory and Applications of Step Motors, 10, Permanent Magnet Sept Motors, pp.206-251.

14. NEEDHAM, J.C., "Transistor Power Sources for High Performance Arc Welding", Weld. Inst. Res. Bull., 1977, 18, pp.63-67.

15. Quardon Qms 85-1185 Microcomputer Handbook, Quarndon Electronics, Ltd., Slack Lane, Derby, DE3 3ED.

16. AM9511 Arithmetic Processor Data Sheet, Advanced Micro Devices (U.K.) Ltd., A.M.D. House, Goldsworth Road, Woking, Surrey, GU21 1JT.

17. AM8155 Data Sheet, Advanced Micro Devices (U.K.) Ltd., A.M.D. House, Goldsworth Road, Woking, Surrey, GU21 1JT.

18. PL/M-80 Programming Manual, Intel Corporation (U.K.) Ltd., Broadfield House, 4 Between Towns Road, Cowely, Oxford, OX4 3NB.

19. 8080/8085 Assembly Language Programming Manual, Intel Corporation (U.K.) Ltd., Broadfield House, 4 Between Towns Road, Cowely, Oxford, OX4 3NB.

20. Guide to Intellec Series II, Intel Corporation (U.K.) Ltd., Broadfield House, 4 Between Towns Road, Cowely, Oxford, OX4 3NB.

21. ISIS II Users Guide, Intel Corporation (U.K.) Ltd., Broadfield House, 4 Between Towns Road, Cowely, Oxford, OX4 3NB.

22. ADAMS, L.F., Engineering Measurements and Instrumentation, English Universities Press Ltd., London, 1975.

23. DAVEY, P.G., SHULTZ, D.L., "Sensory Control of Fixed Arm Robots for Continuous Path Fusion Welding of Vehicle Bodies", Proc. Robotics Initiative, Royal Holloway College, 1983.

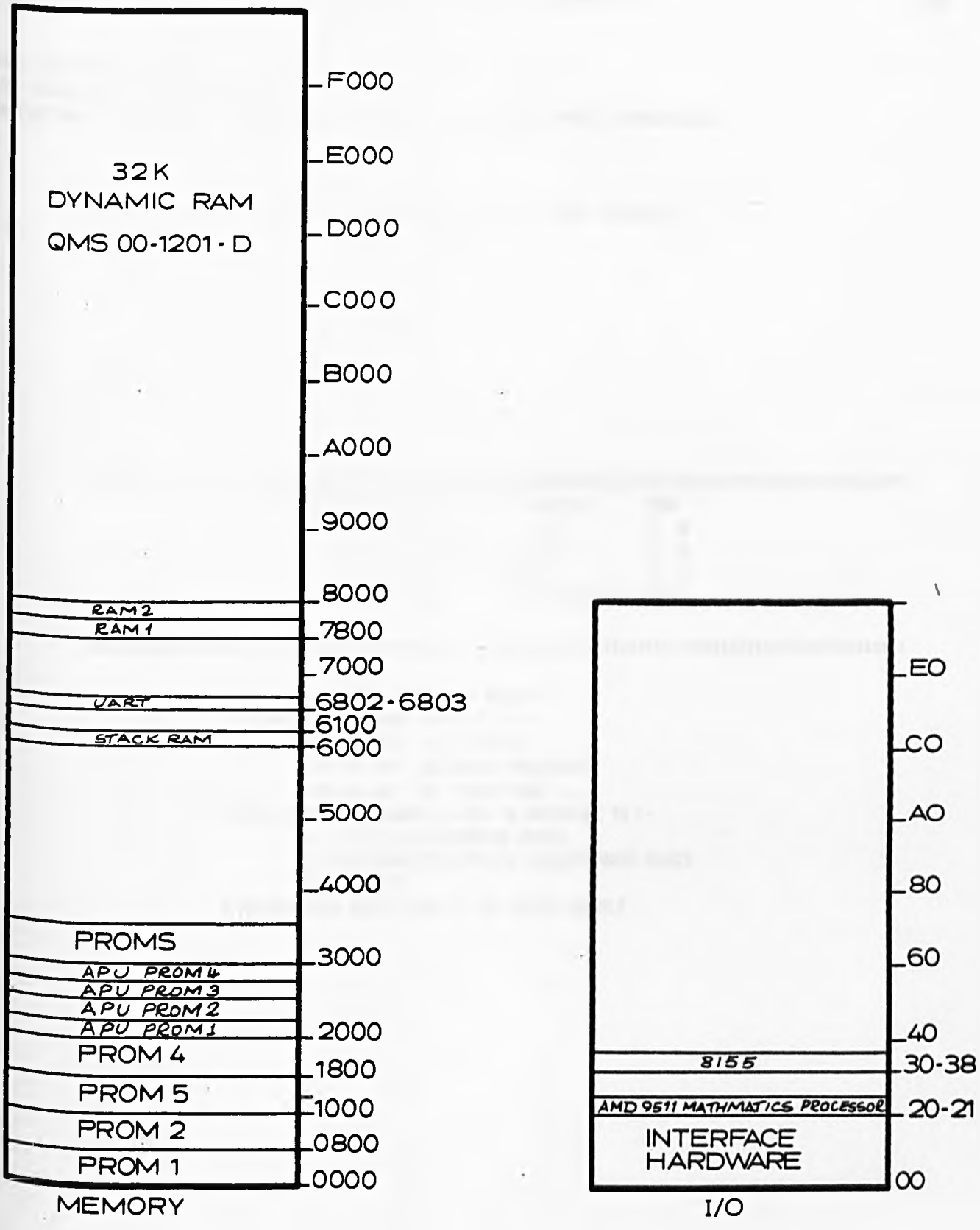
24. Intel 86/330 Microcomputer, Intel Corporation (U.K.) Ltd., Broadfield House, 4 Between Towns Road, Cowely, Oxford, OX4 3NB.

APPENDIX

MEMORY AND I/O MAPS

SOFTWARE LISTINGS

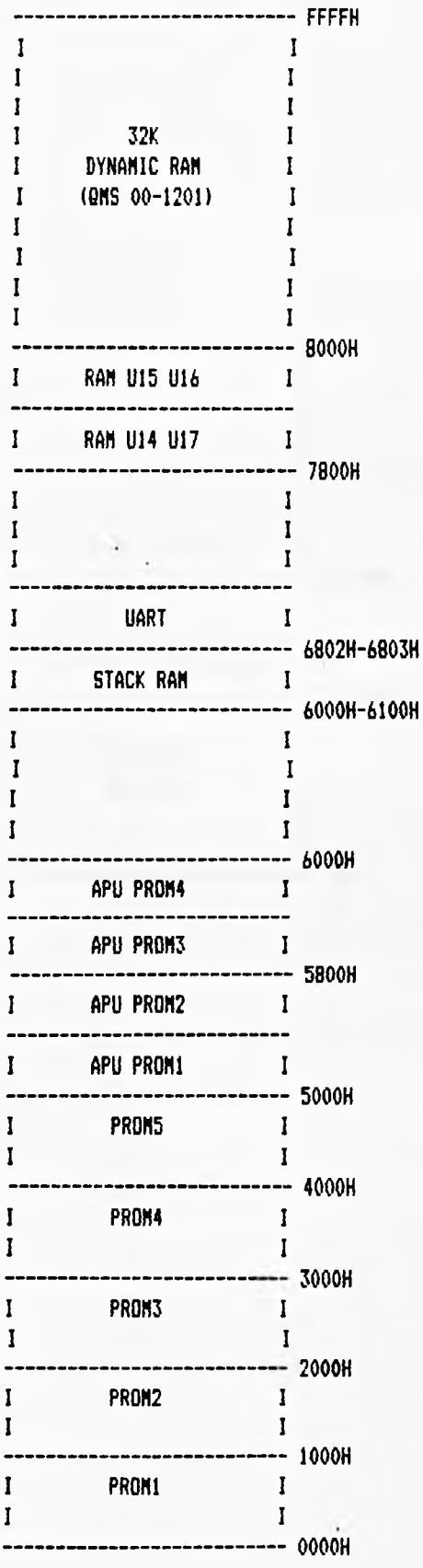
1. TIG Weld
2. Set Parameters Procedure
3. Position Procedure
4. Welds Procedure
5. Real Time Interrupt Procedure
6. Sequence Procedure
7. Path Procedure
8. Set Origin Procedure
9. Power Source Control Procedures
10. Set Limits Procedure
11. Tape I/O
12. Reset System Procedure
13. Motor Calculation Procedure
14. Arithmetic Procedures
15. Motor Control Routines
16. Interrupt Routines
17. Arc Interrupt Routine
18. Interrupt Mask Routine
19. Interrupt Vectors



CONTROLLER MEMORY / I/O MAPS

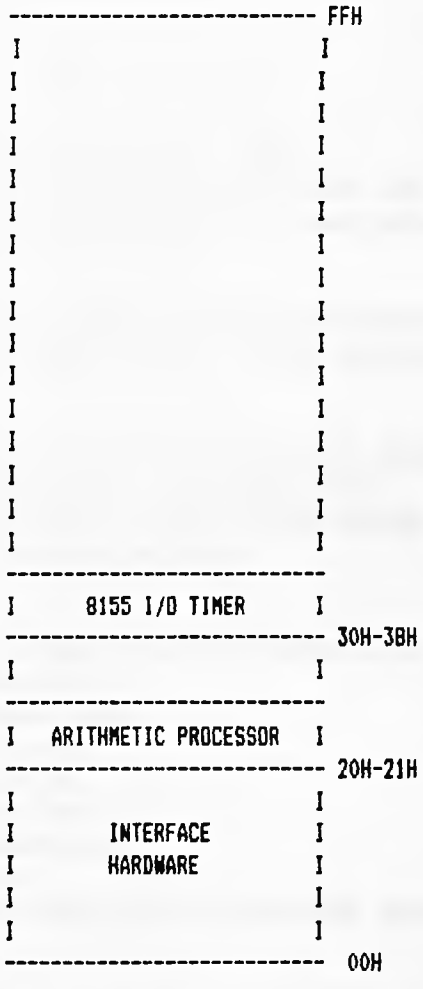
```
ISIS-II PL/M-80 V3.1 COMPILATION OF MODULE TIGWELD
OBJECT MODULE PLACED IN :F1:WELD1.OBJ
COMPILER INVOKED BY:  PLM80 :F1:WELD1.PLM DEBUG WORKFILES(:F1:,:F1:) PRINT(:F1:WELD1.LST)


          $TITLE ('TIG$WELD  WELDING CONTROL PROGRAM  VER 3.00  DATE 04/07/81')
  1       TIG$WELD: DO ;



/*****************************************************************************************************
                        TTTTT IIIII  GGGG     W   W EEEEE L     DDDD
                          T     I   G         W   W E     L     D   D
                          T     I   G  GG     W W W EEE   L     D   D
                          T     I   G   G     W W W E     L     D   D
                          T   IIIII  GGG       W W  EEEEE LLLLL DDDD

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

                A PLM80 PROGRAM TO CONTROL TIG WELDING.
                THE CONTROLER HARDWARE CONSISTS OF :-
                        QMS 85-1185  MICROCOMPUTER
                        QMS 00-1491  ARITHMETIC PROCESSOR
                        QMS 00-1201  32K DYNAMIC RAM
                SPECIAL PURPOSE HARDWARE IS USED TO INTERFACE TO :-
                        X, Y AND Z AXIS STEPPING MOTORS
                        50 AMP TRANSISTORISED TIG WELDING POWER SOURCE
                        GAS SOLENOID
                A NEWBURY 7008 VDU IS USED AS THE SYSTEM CONSOLE
*/
```

```
$EJECT
/*  SYSTEM MEMORY MAP :-
```




```
*/
```

```
$EJECT
/* IO MAP :-
```


FFH
8155 I/O TIMER
30H-38H
ARITHMETIC PROCESSOR
20H-21H
INTERFACE
HARDWARE
00H

```
*/
```

```
$EJECT
/* SYSTEM COMMANDS :-

    POSITION
          ALLOWS THE USER TO REPOSITION THE TORCH.
           THE MAIN USE OF THIS IS TO ENABLE THE TORCH TO BE RETRACTED TO A CONVIENIENT
           POSITION SO THAT THE WORKPIECE MAY BE LOADED AND UNLOADED.

    PATH
           ALLOWS THE USER TO ENTER A THREE DIMENSIONAL PATH INTO THE SYSTEM MEMORY.
           ALL PATHS MUST START AND END AT THE REST POSITION.

    LIMITS
           THIS ALLOWS THE USER TO SET UPPER AND LOWER LIMITS, IN EACH DIRECTION,
           BEYOND WHICH THE TORCH IS NOT ALLOWED TO MOVE.
           THESE LIMITS SHOULD BE SET SUCH THAT THE TRAVERSE DOES NOT MOVE BEYOND ITS
           MECHANICALY SAFE OPERATING REGION.

    PARAMETERS
           THIS ALLOWS THE USER TO MODIFY THE PRESENT SET OF WELDING PARAMETRS.
           i.e.  WELDING SPEED
                 BACKGROUND CURRENT
                 PULSE CURRENT
                 PULSE ON TIME
                 PULSE OFF TIME
                 PRE-TRAVERSE TIME
                 FALL TIME
           THE USER MAY ASSIGN VALUES TO EACH PARAMETER INDIVIDUALY.

    WELD
           THIS COMMAND INSTRUCTS THE CONTROLER TO CARRY OUT A WELDING SEQUENCE
           ACCORDING TO THE PRESENT SET OF PATH AND WELD PARAMETERS.

    DUMP
           THIS COMMAND ALLOWS THE USER TO SAVE THE PATH, PARAMETERS AND
           POSITIONING LIMITS FROM MEMORY TO MAGNETIC TAPE.

    LOAD
           THIS COMMAND ALLOWS THE USER TO LOAD A PATH, PARAMETERS AND POSITIONING LIMITS
           FROM MAGNETIC TAPE INTO SYSTEM MEMORY.
```

```
$EJECT
/*  ARITHMETIC CONVENTIONS
    THE SYSTEM SOFTWARE PROCESSES THREE TYPES OF ARITHMETIC DATA i.e.:-

        SINGLE PRECISION FIXED POINT
        THIS IS A 16 BIT REPRESENTATION OF INTEGERS IN THE RANGE  -32,768 TO +32,767.
        THEY ARE STORED IN MEMORY AS TWO CONSECUTIVE BYTES, THE MSB AT THE LOWER ADDRESS.
        PLM80 SYMBOLS USED TO REPRESENT THIS TYPE OF VARIABLE ARE GIVEN THE PSEUDO EXTENSION 'SINGLE'.
        E.G. :- FIVE$SINGLE

        DOUBLE PRECISION FIXED POINT
        THIS IA S 32 BIT REPRESENTATION OF INTEGERS IN THE RANGE -2,147,483,648 TO +2,147,483,647.
        THEY ARE STORED IN MEMORY AS FOUR CONSECUTIVE BYTES WITH THE MSB AT THE LOWER ADDRESS.
        PLM80 SYMBOLS USED TO REPRESENT THIS TYPE OF VARIABLE ARE NOT GIVEN A PSEUDO EXTENSION.
        E.G. :- FIVE

        FLOATING POINT
        THIS IS A 32 BIT REPRESENTATION OF REAL NUMBERS IN THE RANGE +-(2.7EE-20 TO 9.2EE+18) AND ZERO.
        THE MOST SIGNIFICANT BIT REPRESENTS THE SIGN OF THE MANTISSA.
        THE LEAST SIGNIFICANT 24 BITS REPRESENT THE MAGNITUDE OF THE MANTISSA.
        THE REMAINING SEVEN BITS (BITS 24 TO 30) REPRESENT THE EXPONENT AS A 2'S COMPLEMENT NUMBER.
        THE 32 BIT WORD IS STORED IN MEMORY AS FOUR CONSECUTIVE BYTES, THE MSB BEING AT THE LOWER ADDRESS.
        PLM80 SYMBOLS USED TO REPRESENT THIS TYPE OF VARIABLE ARE GIVEN THE PSEUDO EXTENSION 'FLOAT'.
        E.G. :- FIVE$FLOAT


    SYSTEM INTERRUPTS

        RESET
        CAUSES A TOTAL RE-INITIALISATION OF THE SYSTEM.
        ALL WELDING PARAMETERS ARE CLEARED.
        THE GAS SOLENOID IS SET TO OFF.
        THE TIG POWER SOURCE IS SET TO :-  START=INACTIVE   STOP=INACTIVE.
        THE POWER SOURCE CURRENT IS SET TO ZERO.
        THE VDU SCREEN IS INITIALISED.
        THE SYSTEM POSITIONING SPEED, ORIGIN AND
        POWER SOURCE CURRENT RANGE MUST BE SET UP BY THE USER
        BEFORE THE SOFTWARE ALLOWS ANY OTHER COMMANDS TO BE EXECUTED.

        RST 6.5
        THIS IS USED AS A TRAVERSE INTERRUPT.
        THE POSITION CONTROL INTERFACE ALLOWS THE SOFTWARE TO SET A TARGET POINT.
        THE HARDWARE WILL MOVE THE TORCH TO THAT POINT WITHOUT ANY FURTHER SOFTWARE INTERVENTION.
        RST 6.5 BECOMES ACTIVE WHEN THE TORCH REACHES ITS TARGET.

        RST 5.5
        THIS PROVIDES A REAL TIME INTERRUPT OF PERIOD 10mS.
        THIS IS USED FOR TIMING OF CURRENT PULSES DURING WELDING,
        AND OTHER REAL TIME EVENTS.


    INTERRUPT PRIORITY
    WHILST RST 6.5 IS BEING SERVICED RST 5.5 WILL BE ENABLED.
    WHILST RST 5.5 IS BEING SERVICED RST 6.5 WILL BE DISABLED.
    RESET AND TRAP ARE ALWAYS ENABLED.
*/
```

```
$EJECT

/* SYSTEM STATUS FLAGS
   FLAGS ARE USED TO COMMUNICATE THE STATUS OF THE SYSTEM BETWEEN ROUTINES.
   ANY ROUTINE WHICH CHANGES THE STATUS MUST ALSO UPDATE THE APPROPIATE FLAGS.

      CURRENT$INCREMENT$SET
         INDICATES THAT THE USER HAS SET A VALUE FOR THE CURRENT INCREMENT.

      PATH$NAME$SET
         INDICATES THAT THE USER HAS ENTERED A PATH NAME .

      REST$POSITION$SET
         INDICATES THAT THE USER HAS DEFINED THE RESTPOSITION.

      X$UPPER$LIMIT$SET          +
      X$LOWER$LIMIT$SET          +
      Y$UPPER$LIMIT$SET          + THESE FLAGS INDICATE THE STATUS OF THE POSITION LIMITS
      Y$LOWER$LIMIT$SET          +
      Z$UPPER$LIMIT$SET          +
      Z$LOWER$LIMIT$SET          +

*/
```

```
                $EJECT
                /* EXTERNAL PROCEDURES */
                $INCLUDE (:F1:WELD.EQU)
            =   $NOLIST
            =
                $INCLUDE (:F1:MATH.EQU)
            =   $NOLIST
                $INCLUDE (:F1:WELD.EDS)
            =   $NOLIST

 74   1         RESET$SYSTEM: PROCEDURE EXTERNAL ;
 75   2         END RESET$SYSTEM ;

 76   1         SEQUENCE: PROCEDURE EXTERNAL ;
 77   2         END SEQUENCE ;

 78   1         SET$INTERRUPT$MASK: PROCEDURE EXTERNAL ;
 79   2         END SET$INTERRUPT$MASK;

 80   1         SET$LIMITS: PROCEDURE EXTERNAL ;
 81   2         END SET$LIMITS ;

 82   1         SET$ORIGIN: PROCEDURE EXTERNAL ;
 83   2         END SET$ORIGIN ;

 84   1         POSITION: PROCEDURE EXTERNAL ;
 85   2         END POSITION ;

 86   1         PATH: PROCEDURE EXTERNAL ;
 87   2         END PATH ;

 88   1         SET$PARAMETERS: PROCEDURE EXTERNAL ;
 89   2         END SET$PARAMETERS ;

 90   1         WELD$SEQUENCE: PROCEDURE EXTERNAL ;
 91   2         END WELD$SEQUENCE ;

 92   1         SET$DIRECTION$CONTROL: PROCEDURE EXTERNAL ;
 93   2         END SET$DIRECTION$CONTROL ;

 94   1         STROBE$CONTROL$KEYS: PROCEDURE EXTERNAL ;
 95   2         END STROBE$CONTROL$KEYS ;

 96   1         UNDER$RANGE$ERROR: PROCEDURE EXTERNAL ;
 97   2         END UNDER$RANGE$ERROR ;

 98   1         OVER$RANGE$ERROR: PROCEDURE EXTERNAL ;
 99   2         END OVER$RANGE$ERROR ;

100   1         LOAD: PROCEDURE EXTERNAL ;
101   2         END LOAD ;

102   1         DUMP: PROCEDURE EXTERNAL ;
103   2         END DUMP ;
```

```
        $EJECT
        /* EXTERNAL DATA */
104  1      DECLARE PARAMETER$GROUP$TABLE (256) STRUCTURE (PULSE$CURRENT$FLOAT      (4) BYTE,
                                                          BACKGROUND$CURRENT$FLOAT (4) BYTE,
                                                          PULSE$ON$TIME$FLOAT      (4) BYTE,
                                                          PULSE$OFF$TIME$FLOAT     (4) BYTE,
                                                          WELDING$DIVISOR$FLOAT    (4) BYTE,
                                                          PRE$TRAVERSE$TIME$FLOAT  (4) BYTE,
                                                          FALL$TIME$FLOAT          (4) BYTE,
                                                          R$0$DIVISOR$SINGLE       (2) BYTE,
                                                          R$1$DIVISOR$SINGLE       (2) BYTE,
                                                          R$2$DIVISOR$SINGLE       (2) BYTE,
                                                          R$3$DIVISOR$SINGLE       (2) BYTE,
                                                          TOUCH$START                  BYTE) EXTERNAL ;
105  1      DECLARE GROUP$NUMBER BYTE EXTERNAL ;
106  1      DECLARE(ONE,
                    FOUR,
                    FIVE,
                    TEN   )                    (4) BYTE EXTERNAL ;
107  1      DECLARE TENSINGLE                  (2) BYTE EXTERNAL ;
108  1      DECLARE (ZERO$FLOAT,
                     ONE$FLOAT,
                     TEN$FLOAT,
                     TEN$THOUSAND$FLOAT,
                     TWO$FIVE$FIVE$FLOAT )      (4) BYTE EXTERNAL ;
109  1      DECLARE (CURRENT$INCREMENT$FLOAT,
                     CURRENT$RANGE$FLOAT     )  (4) BYTE EXTERNAL ;
110  1      DECLARE (X$ORDINATE,
                     Y$ORDINATE,
                     Z$ORDINATE  )              (4) BYTE EXTERNAL ;
111  1      DECLARE R$ORDINATES (4) STRUCTURE (ORDINATE (4) BYTE) EXTERNAL ;
112  1      DECLARE (PATH$NAME$SET,
                     CURRENT$INCREMENT$SET)           BYTE  EXTERNAL ;
113  1      DECLARE (MAIN$DIVISOR$FLOAT,
                     RTC$DIVISOR$FLOAT       )  (4) BYTE EXTERNAL ;
114  1      DECLARE  RTC$DIVISOR$SINGLE        (2) BYTE EXTERNAL ;
```

```
            $EJECT
            /* PUBLIC DATA */

            /* DISPLAY DATA */
115   1       DECLARE EDIT$BUFFER    (84) BYTE    PUBLIC ;
116   1       DECLARE ASCII          (11) BYTE    PUBLIC ;
117   1       DECLARE BUFFER$POINTER      ADDRESS PUBLIC ;
118   1       DECLARE DIGIT          (*)  BYTE    PUBLIC
                    DATA ('0123456789') ;
119   1       DECLARE NUMBER$FLOAT        (4) BYTE PUBLIC ;

                /* CONSOLE MESSAGES */
120   1       DECLARE UNITS$SEC        (*) BYTE PUBLIC
                    DATA (' Sec', EOT) ;
121   1       DECLARE UNITS$MM$PER$SEC (*) BYTE PUBLIC
                    DATA (' mm/S', EOT) ;
122   1       DECLARE UNITS$AMP        (*) BYTE PUBLIC
                    DATA (' Amp', EOT) ;
123   1       DECLARE OVERFLOW$MESSAGE (*) BYTE PUBLIC
                    DATA ('*****', BELL, EOT) ;
124   1       DECLARE OVER$RANGE$MESSAGE (*) BYTE PUBLIC
                    DATA (DELETE$LINE,
                          'VALUE TOO LARGE', EOT) ;
125   1       DECLARE UNDER$RANGE$MESSAGE (*) BYTE PUBLIC
                    DATA (DELETE$LINE,
                          'VALUE TOO SMALL', EOT) ;
126   1       DECLARE LIMITS$WARNING$MESSAGE (*) BYTE PUBLIC
                    DATA (DELETE$LINE, BACKGROUND$ON, BLINK$ON,
                          'LIMITS NOT SET',
                          BLINK$OFF, BACKGROUND$OFF, EOT          ) ;

                /* UART ADDRESSES */
127   1       DECLARE UART$STATUS                BYTE PUBLIC AT (06802H) ;
128   1       DECLARE UART$DATA                  BYTE PUBLIC AT (06803H) ;

              /* POWER SOURCE CURRENT RANGE LIMITS */
129   1        DECLARE MAX$CURRENT$RANGE$FLOAT (4) BYTE PUBLIC
                    DATA (ONE$THOUSAND$REAL) ;
130   1       DECLARE MIN$CURRENT$RANGE$FLOAT (4) BYTE PUBLIC
                    DATA (ZERO$REAL) ;
```

```
            $EJECT
            /* LOCAL DATA */
131   1         DECLARE I BYTE ;
132   1         DECLARE (TEMP$REG$1,
                       TEMP$REG$2 ) (11) BYTE ;
133   1         DECLARE BUFFER BASED BUFFER$POINTER BYTE ;
134   1         DECLARE (WORD$START,
                       WORD$END   ) ADDRESS ;
```

```
$EJECT

  /* DISPLAY ROUTINES */

 1  1    CONVERT$TO$ASCII: PROCEDURE (VALUE$ADDRESS) PUBLIC ;
 2  2        DECLARE VALUE$ADDRESS ADDRESS ;
 2  2        DECLARE VALUE BASED VALUE$ADDRESS (4) BYTE ;
 2  2        DECLARE (I, J) BYTE ;

 2  2       DO I = 0 TO 3 ;
 3  3           TEMP$REG$1(I) = VALUE(I) ;
 3  3       END ;
 2  2       IF TEMP$REG$1$NEGATIVE
            THEN
 2  2           DO ;
 3  3               CALL NEGATE (.TEMP$REG$1) ;
 3  3               CALL STORE$RESULT (.TEMP$REG$1) ;
 3  3               ASCII(0) = '-' ;
 3  3           END ;
            ELSE
 2  2          ASCII(0) = '+' ;
 2  2       DO I = 1 TO 10 ;
 3  3           CALL DIVIDE (.TEMP$REG$1, .TEN) ;
 3  3           CALL STORE$RESULT (.TEMP$REG$2) ;
 3  3           CALL MULTIPLY (.RESULT, .TEN) ;
 3  3           CALL SUBTRACT (.TEMP$REG$1, .RESULT) ;
 3  3           ASCII(11-I) = DIGIT(RESULT(3)) ;

 3  3           DO J = 0 TO 3 ;
 4  4               TEMP$REG$1(J) = TEMP$REG$2(J) ;
 4  4           END ;
 3  3       END ;
 2  2    END CONVERT$TO$ASCII ;

 1  1    CONSOLE$OUTPUT: PROCEDURE (CHARACTER) PUBLIC ;
 2  2        DECLARE CHARACTER BYTE ;
 2  2        DO WHILE UART$TRANSMIT$REGISTER$FULL ;
 3  3        END ;
 2  2        UART$DATA = CHARACTER ;
 2  2    END CONSOLE$OUTPUT ;

 1  1    CONSOLE$INPUT: PROCEDURE BYTE PUBLIC ;
 2  2        DECLARE READ$DATA BYTE ;
 2  2        DO FOREVER ;
 3  3            DO WHILE UART$RECEIVE$REGISTER$EMPTY ;
 4  4            END ;
 3  3           IF UART$RECEIVE$ERROR <> 0
                THEN
 3  3              READ$DATA = UART$DATA ;          /*DUMMY READ OF UART TO CLEAR ERRORS */

                ELSE
 3  3                RETURN UART$DATA MASKING$OFF BIT$7 ;
 3  3        END ;
 2  2    END CONSOLE$INPUT ;
```

```
        $EJECT
176  1    WRITE$MESSAGE: PROCEDURE (MESSAGE$ADDRESS) PUBLIC ;
177  2        DECLARE MESSAGE$ADDRESS ADDRESS ;
178  2        DECLARE CHARACTER BASED MESSAGE$ADDRESS BYTE ;
179  2        DO WHILE CHARACTER <> EOT ;
180  3            CALL CONSOLE$OUTPUT (CHARACTER) ;
181  3            MESSAGE$ADDRESS = MESSAGE$ADDRESS + 1 ;
182  3        END ;
183  2    END WRITE$MESSAGE ;

184  1     POSITION$CURSOR: PROCEDURE ( COLUMN$NUMBER, LINE$NUMBER) PUBLIC ;
185  2        DECLARE ( COLUMN$NUMBER,
                       LINE$NUMBER  ) BYTE ;
186  2        CALL CONSOLE$OUTPUT (MOVE$CURSOR) ;
187  2        CALL CONSOLE$OUTPUT (COLUMN$NUMBER + 20H) ;
188  2        CALL CONSOLE$OUTPUT (LINE$NUMBER +   20H) ;
189  2     END POSITION$CURSOR ;

190  1     DISPLAY$ASCII$BUFFER: PROCEDURE (FIRST, LAST, BLANKING$ENABLE) PUBLIC ;
191  2        DECLARE (FIRST, LAST, BLANKING$ENABLE, I) BYTE ;
192  2        DO I = FIRST TO (LAST-1) ;
193  3            IF BLANKING$ENABLE
                 THEN
194  3                DO ;
195  4                    IF ASCII(I) = '0'
                         THEN
196  4                        CALL CONSOLE$OUTPUT(SPACE) ;
                         ELSE
197  4                        DO ;
198  5                            CALL CONSOLE$OUTPUT (ASCII(I)) ;
199  5                            BLANKING$ENABLE = FALSE ;
200  5                        END ;
201  4                END ;
                     ELSE
202  3                    CALL CONSOLE$OUTPUT (ASCII(I)) ;
203  3            END ;
204  2            CALL CONSOLE$OUTPUT (ASCII(LAST)) ;
205  2    END DISPLAY$ASCII$BUFFER ;
```

```
        $EJECT
206  1    CONVERT$SINGLE$TO$ASCII: PROCEDURE (VALUE$ADDRESS) PUBLIC ;
207  2         DECLARE VALUE$ADDRESS ADDRESS ;
208  2         DECLARE VALUE BASED VALUE$ADDRESS (2) BYTE ;
209  2         DECLARE (I, J) BYTE ;

210  2        DO I = 0 TO 1 ;
211  3            TEMP$REG$1(I) = VALUE(I) ;
212  3         END ;

213  2         IF TEMP$REG$1$NEGATIVE
               THEN
214  2            DO ;
215  3                 CALL NEGATE$SINGLE (.TEMP$REG$1) ;
216  3                 CALL STORE$RESULT$SINGLE (.TEMP$REG$1) ;
217  3                 ASCII(0) = '-' ;
218  3            END ;
               ELSE
219  2            ASCII(0) = '+' ;
220  2         DO I = 1 TO 5 ;
221  3             CALL DIVIDE$SINGLE (.TEMP$REG$1, .TEN$SINGLE) ;
222  3             CALL STORE$RESULT$SINGLE (.TEMP$REG$2) ;
223  3             CALL MULTIPLY$SINGLE (.RESULT$SINGLE, .TEN$SINGLE) ;
224  3             CALL SUBTRACT$SINGLE (.TEMP$REG$1, .RESULT$SINGLE) ;
225  3             ASCII(6-I) = DIGIT (RESULT$SINGLE(1)) ;
226  3             DO J = 0 TO 1 ;
227  4                 TEMP$REG$1(J) = TEMP$REG$2(J) ;
228  4             END ;
229  3        END ;
230  2    END CONVERT$SINGLE$TO$ASCII ;
```

```
                $EJECT
231   1         READ$LINE: PROCEDURE PUBLIC ;
232   2             DECLARE ( CONSOLE$DATA,
                              I           ) BYTE ;
233   2             DO I = 0 TO 83 ;
234   3                EDIT$BUFFER(I) = SPACE ;
235   3             END ;
236   2             CONSOLE$DATA = SPACE ;
237   2             I = 0 ;
238   2             DO WHILE ((CONSOLE$DATA <> CARRIGE$RETURN) AND (I < 83) AND (CONSOLE$DATA <> ESC)) ;
239   3                 CONSOLE$DATA = CONSOLE$INPUT ;
240   3                 IF (    (CONSOLE$DATA > 1FH)
                          OR (CONSOLE$DATA = CARRIGE$RETURN)
                          OR (CONSOLE$DATA = ESC)
                          OR (CONSOLE$DATA = SPACE)
                          OR (CONSOLE$DATA = RUB$OUT)           )
                     THEN
241   3                    DO ;
242   4                       IF CONSOLE$DATA = RUBOUT
                              THEN
243   4                          DO ;
244   5                             IF I = 0
                                    THEN
245   5                                CALL CONSOLE$OUTPUT (BELL) ;
                                    ELSE
246   5                                DO ;
247   6                                   I = I - 1 ;
248   6                                   EDIT$BUFFER(I) = SPACE ;
249   6                                   CALL CONSOLE$OUTPUT (CURSOR$LEFT) ;
250   6                                   CALL CONSOLE$OUTPUT (DELETE$CHARACTER) ;
251   6                                END ;
252   5                          END ;
                             ELSE

253   4                          DO ;
254   5                             EDIT$BUFFER(I) = CONSOLE$DATA ;
255   5                              CALL CONSOLE$OUTPUT (CONSOLE$DATA) ;
256   5                             I = I + 1 ;
257   5                          END ;
258   4                    END ;
259   3             END ;
260   2             IF ((CONSOLE$DATA <> CARRIGE$RETURN) OR (CONSOLE$DATA <> ESC))
                    THEN
261   2                DO ;
262   3                   EDIT$BUFFER(83) = CARRIGE$RETURN ;
263   3                   CALL CONSOLE$OUTPUT (BELL) ;
264   3                   CALL CONSOLE$OUTPUT (CARRIGE$RETURN) ;
265   3                END ;
266   2         END READ$LINE ;
```

```
        $EJECT
267  1      DISPLAY$IN$NORMALISED$UNITS: PROCEDURE (VALUE$ADDRESS) PUBLIC ;
268  2          DECLARE VALUE$ADDRESS ADDRESS ;
269  2          DECLARE I BYTE ;
270  2          CALL CONVERT$TO$ASCII (VALUE$ADDRESS) ;
271  2          CALL CONSOLE$OUTPUT (ASCII(0)) ;
272  2          CALL DISPLAY$ASCII$BUFFER (1, 8, ENABLE$BLANKING) ;
273  2          CALL CONSOLE$OUTPUT ('.') ;
274  2          CALL DISPLAY$ASCII$BUFFER (9, 10, DISABLE$BLANKING) ;
275  2          CALL WRITE$MESSAGE (.('  ', EOT)) ;
276  2      END DISPLAY$IN$NORMALISED$UNITS ;

277  1      NORMALISE$Y$ORDINATE: PROCEDURE PUBLIC ;
278  2          CALL MULTIPLY (.Y$ORDINATE, .FIVE) ;
279  2          CALL DIVIDE   (.RESULT,         .FOUR) ;
280  2      END NORMALISE$Y$ORDINATE ;

281  1      DISPLAY$X: PROCEDURE PUBLIC ;
282  2          CALL POSITION$CURSOR (X$FIELD) ;
283  2          CALL DISPLAY$IN$NORMALISED$UNITS (.X$ORDINATE) ;
284  2      END DISPLAY$X ;

285  1      DISPLAY$Z: PROCEDURE PUBLIC ;
286  2          CALL POSITION$CURSOR (Z$FIELD) ;
287  2          CALL DISPLAY$IN$NORMALISED$UNITS (.Z$ORDINATE) ;
288  2      END DISPLAY$Z ;

289  1      DISPLAY$Y: PROCEDURE PUBLIC ;
290  2          CALL POSITION$CURSOR (Y$FIELD) ;
291  2          CALL NORMALISE$Y$ORDINATE ;
292  2          CALL DISPLAY$IN$NORMALISED$UNITS (.RESULT) ;
293  2      END DISPLAY$Y ;

294  1      DISPLAY$R: PROCEDURE (R$NUMBER) PUBLIC ;
295  2          DECLARE R$NUMBER BYTE ;
296  2          DECLARE ORDINATE$ADDRESS ADDRESS ;
297  2          DO CASE R$NUMBER ;
298  3              DO ;
299  4                  CALL POSITION$CURSOR (R$0$FIELD) ;
300  4                  ORDINATE$ADDRESS = .R$ORDINATES(0).ORDINATE ;
301  4              END ;
302  3              DO ;
303  4                  CALL POSITION$CURSOR (R$1$FIELD) ;
304  4                  ORDINATE$ADDRESS = .R$ORDINATES(1).ORDINATE ;
305  4              END ;
306  3              DO ;
307  4                  CALL POSITION$CURSOR (R$2$FIELD) ;
308  4                  ORDINATE$ADDRESS = .R$ORDINATES(2).ORDINATE ;
309  4              END ;
310  3              DO ;
311  4                  CALL POSITION$CURSOR (R$3$FIELD) ;
312  4                  ORDINATE$ADDRESS = .R$ORDINATES(3).ORDINATE ;
313  4              END ;
314  3          END ;
315  2          CALL CONVERT$TO$ASCII (ORDINATE$ADDRESS) ;
316  2          CALL CONSOLE$OUTPUT (ASCII(0)) ;
317  2          CALL DISPLAY$ASCII$BUFFER (1, 10, ENABLE$BLANKING) ;
```

```
318  2          END DISPLAY$R ;

319  1          NEXT$CHARACTER: PROCEDURE ADDRESS PUBLIC ;
320  2              DO WHILE BUFFER = SPACE ;
321  3              BUFFER$POINTER = BUFFER$POINTER + 1 ;
322  3              END ;
323  2              RETURN BUFFER$POINTER ;
324  2          END NEXT$CHARACTER ;

325  1          NEXT$SPACE: PROCEDURE ADDRESS PUBLIC;
326  2              DO WHILE (BUFFER <> SPACE) AND (BUFFER <> CARRIGE$RETURN) ;
327  3                      BUFFER$POINTER = BUFFER$POINTER + 1 ;
328  3              END ;
329  2              RETURN BUFFER$POINTER ;
330  2          END NEXT$SPACE ;

331  1          COMPARE$WORDS: PROCEDURE (POINTER$1, WORD$1$END, POINTER$2, WORD$2$LENGTH) BYTE PUBLIC ;
332  2              DECLARE ( POINTER$1,
                           POINTER$2,
                           WORD$1$END,
                           WORD$1$LENGTH,
                           WORD$2$LENGTH ) ADDRESS ;
333  2              DECLARE CHARACTER$1 BASED POINTER$1 BYTE ;
334  2              DECLARE CHARACTER$2 BASED POINTER$2 BYTE ;
335  2              DECLARE STATUS BYTE ;
336  2              WORD$1$LENGTH = WORD$1$END - POINTER$1 ;
337  2              IF (WORD$1$LENGTH > WORD$2$LENGTH) OR (WORD$1$LENGTH < 4)
                    THEN
338  2                  RETURN FALSE ;
339  2              STATUS = TRUE ;
340  2              DO WHILE POINTER$1 < WORD$1$END ;
341  3                  IF CHARACTER$1 <> CHARACTER$2
                        THEN
342  3                      STATUS = FALSE ;
343  3                  POINTER$1 = POINTER$1 + 1 ;
344  3                  POINTER$2 = POINTER$2 + 1 ;
345  3              END ;
346  2              RETURN STATUS ;
347  2          END COMPARE$WORDS ;

348  1          CLEAR$ERROR$FIELD: PROCEDURE PUBLIC ;
349  2              CALL POSITION$CURSOR (ERROR$FIELD) ;
350  2              CALL CONSOLE$OUTPUT (DELETE$LINE) ;
351  2          END CLEAR$ERROR$FIELD ;

352  1          CLEAR$PROMPT$FIELD: PROCEDURE PUBLIC ;
353  2              CALL POSITION$CURSOR (PROMPT$FIELD) ;
354  2              CALL CONSOLE$OUTPUT (DELETE$LINE) ;
355  2          END CLEAR$PROMPT$FIELD ;

356  1          CLEAR$WARNING$FIELD: PROCEDURE PUBLIC ;
357  2              CALL POSITION$CURSOR (WARNING$FIELD) ;
358  2              CALL CONSOLE$OUTPUT (DELETE$LINE) ;
359  2          END CLEAR$WARNING$FIELD ;
```

```
                $EJECT

                    /* CONSOLE NUMERIC I/O */

360   1             READ$REAL$NUMBER: PROCEDURE BYTE PUBLIC ;
361   2                 DECLARE (FORMAT$ERROR,
                               READ$STATUS,
                               NEGATIVE$SIGN,
                               CHARACTER       ) BYTE ;
362   2                 DECLARE (INTEGER$FLOAT,
                               FRACTION$FLOAT,
                               DENOMINATOR$FLOAT ) (4) BYTE ;

363   2                 FORMAT$ERROR = TRUE ;
364   2                 DO WHILE FORMAT$ERROR ;
365   3                     READ$STATUS,
                            NEGATIVE$SIGN = FALSE ;
366   3                     CALL STORE$NUMBER (.INTEGER$FLOAT, .ZERO$FLOAT) ;
367   3                     CALL STORE$NUMBER (.FRACTION$FLOAT, .ZERO$FLOAT) ;
368   3                     CALL POSITION$CURSOR (COMMAND$FIELD) ;
369   3                     CALL WRITE$MESSAGE (.(DELETE$LINE,
                                                '$', EOT)) ;
370   3                     CHARACTER = CONSOLE$INPUT ;
371   3                     IF CHARACTER = '+'
                            THEN
372   3                         DO ;
373   4                         CALL CONSOLE$OUTPUT ('+') ;
374   4                         CHARACTER = CONSOLE$INPUT ;
375   4                         END ;
                            ELSE
376   3                         DO ;
377   4                         IF CHARACTER = ('-')
                                THEN
378   4                             DO ;
379   5                             CALL CONSOLE$OUTPUT ('-') ;
380   5                             NEGATIVE$SIGN = TRUE ;
381   5                             CHARACTER = CONSOLE$INPUT ;
382   5                             END ;
383   4                         END ;
384   3                     DO WHILE ( (CHARACTER >= '0') AND (CHARACTER <= '9') ) ;
385   4                         CALL CONSOLE$OUTPUT (CHARACTER) ;
386   4                         READ$STATUS = TRUE ;
387   4                         CALL REAL$MULTIPLY (.INTEGER$FLOAT, .TEN$FLOAT) ;
388   4                         DO WHILE CHARACTER <> '0' ;
389   5                             CALL REAL$ADD (.RESULT, .ONE$FLOAT) ;
390   5                             CHARACTER = CHARACTER - 1 ;
391   5                         END ;
392   4                         CALL STORE$RESULT (.INTEGER$FLOAT) ;
393   4                         CHARACTER = CONSOLE$INPUT ;
394   4                     END ;
395   3                     IF CHARACTER = '.'
                            THEN
396   3                         DO ;
397   4                         CALL CONSOLE$OUTPUT ('.') ;
398   4                         CHARACTER = CONSOLE$INPUT ;
399   4                         CALL STORE$NUMBER (.DENOMINATOR$FLOAT, .TEN$FLOAT) ;
400   4                         DO WHILE ( (CHARACTER >= '0') AND (CHARACTER <= '9') ) ;
```

```
401  5                      CALL CONSOLE$OUTPUT (CHARACTER) ;
402  5                      READ$STATUS = TRUE ;
403  5                      CALL STORE$NUMBER (.RESULT, .ZERO$FLOAT) ;
404  5                      DO WHILE CHARACTER <> '0' ;
405  6                          CALL REAL$ADD (.RESULT, .ONE$FLOAT) ;
406  6                          CHARACTER = CHARACTER - 1 ;
407  6                      END ;
408  5                      CALL REAL$DIVIDE (.RESULT, .DENOMINATOR$FLOAT) ;
409  5                      CALL REAL$ADD    (.RESULT, .FRACTION$FLOAT) ;
410  5                      CALL STORE$RESULT (.FRACTION$FLOAT) ;
411  5                      CALL REAL$MULTIPLY (.DENOMINATOR$FLOAT, .TEN$FLOAT) ;
412  5                      CALL STORE$RESULT  (.DENOMINATOR$FLOAT) ;
413  5                      CHARACTER = CONSOLE$INPUT ;
414  5                   END ;
415  4                END ;
416  3             CALL POSITION$CURSOR (ERROR$FIELD) ;
417  3             CALL CONSOLE$OUTPUT  (DELETE$LINE) ;
418  3             IF CHARACTER = CARRIGE$RETURN
                   THEN
419  3                DO ;
420  4                   FORMAT$ERROR = FALSE ;
421  4                   CALL REAL$ADD (.INTEGER$FLOAT, .FRACTION$FLOAT) ;
422  4                   IF NEGATIVE$SIGN
                         THEN
423  4                      CALL REAL$NEGATE (.RESULT) ;
424  4                   CALL STORE$RESULT (.NUMBER$FLOAT) ;
425  4                END ;
                   ELSE
426  3                CALL WRITE$MESSAGE (.('ERROR IN NUMBER FORMAT', EOT)) ;
427  3          END ;
428  2          RETURN (READ$STATUS) ;
429  2       END READ$REAL$NUMBER ;
```

```
                $EJECT
430   1         DISPLAY$REAL$NUMBER: PROCEDURE (FLOAT$NUMBER$ADDRESS, DECIMAL$DIGITS, DECIMAL$PLACES) PUBLIC ;
431   2             DECLARE (DECIMAL$DIGITS, DECIMAL$PLACES, I) BYTE ;
432   2             DECLARE (FLOAT$NUMBER$ADDRESS) ADDRESS ;
433   2             DECLARE (NUMBER,
                               REMAINDER,
                               TEMP$FLOAT ) (4) BYTE ;
434   2             CALL STORE$NUMBER (.TEMP$FLOAT, FLOAT$NUMBER$ADDRESS) ;
435   2             CALL FIX (.TEMP$FLOAT) ;
436   2             IF MATH$OVERFLOW
                      THEN
437   2                 DO ;
438   3                     CALL WRITE$MESSAGE (.OVERFLOW$MESSAGE) ;
439   3                     RETURN ;
440   3                 END ;
441   2             CALL STORE$RESULT (.NUMBER) ;
442   2             CALL FLOAT (.NUMBER) ;
443   2             CALL REAL$SUBTRACT (.TEMP$FLOAT, .RESULT) ;
444   2             DO I = 0 TO (DECIMAL$PLACES-1) ;
445   3                 CALL REAL$MULTIPLY (.RESULT, .TEN$FLOAT) ;
446   3             END ;
447   2             CALL FIX$ROUND (.RESULT) ;
448   2             CALL STORE$RESULT (.REMAINDER) ;
449   2             DO I = 0 TO (DECIMAL$PLACES-1) ;
450   3                 CALL DIVIDE (.RESULT, .TEN) ;
451   3             END ;
452   2             IF NOT (MATH$ZERO)
                      THEN
453   2                 DO ;
454   3                     CALL ADD (.NUMBER, .ONE) ;
455   3                     CALL STORE$RESULT (.NUMBER) ;
456   3                 END ;
457   2             CALL CONVERT$TO$ASCII (.NUMBER) ;
458   2             CALL CONSOLE$OUTPUT (ASCII(0)) ;
459   2             DO I = 1 TO (10 - DECIMAL$DIGITS) ;
460   3                 IF ASCII(I) <> '0'
                          THEN
461   3                     DO ;
462   4                         CALL WRITE$MESSAGE (.OVERFLOW$MESSAGE) ;
463   4                         RETURN ;
464   4                     END ;
465   3             END ;
466   2             CALL DISPLAY$ASCII$BUFFER ((10 - DECIMAL$DIGITS), 10, ENABLE$BLANKING) ;
467   2             CALL CONSOLE$OUTPUT ('.') ;
468   2             CALL CONVERT$TO$ASCII (.REMAINDER) ;
469   2             CALL DISPLAY$ASCII$BUFFER ((11-DECIMAL$PLACES), 10, DISABLE$BLANKING) ;
470   2         END DISPLAY$REAL$NUMBER ;
```

```
                $EJECT
471   1         DISPLAY$CURRENT$INCREMENT: PROCEDURE PUBLIC ;
472   2             CALL POSITION$CURSOR (CURRENT$INCREMENT$FIELD) ;
473   2             CALL DISPLAY$REAL$NUMBER (.CURRENT$INCREMENT$FLOAT, 2, 4) ;
474   2             CALL WRITE$MESSAGE (.UNITS$AMP) ;
475   2         END DISPLAY$CURRENT$INCREMENT ;

476   1         DISPLAY$CURRENT$RANGE: PROCEDURE PUBLIC ;
477   2             CALL POSITION$CURSOR (CURRENT$RANGE$FIELD) ;
478   2             CALL DISPLAY$REAL$NUMBER (.CURRENT$RANGE$FLOAT, 4, 1 ) ;
479   2             CALL WRITE$MESSAGE (.UNITS$AMP) ;
480   2         END DISPLAY$CURRENT$RANGE ;

481   1         DISPLAY$PATH$NAME: PROCEDURE PUBLIC ;
482   2             DECLARE I BYTE ;
483   2             CALL POSITION$CURSOR (PATH$NAME$FIELD) ;
484   2             CALL CONSOLE$OUTPUT  (DELETE$LINE) ;
485   2             IF PATH$NAME$SET
                      THEN
486   2                 DO ;
487   3                     DO I = 0 TO LAST(PATH$NAME) ;
488   4                         CALL CONSOLE$OUTPUT (PATH$NAME(I)) ;
489   4                     END ;
490   3                 END ;
491   2         END DISPLAY$PATH$NAME ;
```

```
$EJECT

    /* SYSTEM CURRENT RANGE DEFINITION */

492  1  SET$CURRENT$INCREMENT: PROCEDURE ;
493  2      DECLARE (OVER$RANGE,
                    UNDER$RANGE ) BYTE ;

494  2      CALL POSITION$CURSOR (PROMPT$FIELD) ;
495  2      CALL WRITE$MESSAGE (.(DELETE$LINE,
                              'ENTER POWER SOURCE FULL RANGE CURRENT IN Amps', EOT)) ;
496  2      OVER$RANGE,
           UNDER$RANGE = TRUE ;
497  2      DO WHILE (OVER$RANGE OR UNDER$RANGE) ;
498  3          IF READ$REAL$NUMBER
               THEN
499  3              DO ;
500  4                  CALL REAL$SUBTRACT (.MAX$CURRENT$RANGE$FLOAT, .NUMBER$FLOAT) ;
501  4                  IF RESULT$NEGATIVE
                       THEN
502  4                      DO ;
503  5                          OVER$RANGE = TRUE ;
504  5                          CALL OVER$RANGE$ERROR ;
505  5                      END ;
                       ELSE
506  4                      DO ;
507  5                          OVER$RANGE = FALSE ;
508  5                          CALL REAL$SUBTRACT (.NUMBER$FLOAT, .MIN$CURRENT$RANGE$FLOAT) ;
509  5                          IF RESULT$NEGATIVE
                               THEN
510  5                              DO ;
511  6                                  UNDER$RANGE = TRUE ;
512  6                                  CALL UNDER$RANGE$ERROR ;
513  6                              END ;
                               ELSE
514  5                              UNDER$RANGE = FALSE ;
515  5                      END ;
516  4              END ;
517  3      END ;
518  2      CALL STORE$NUMBER (.CURRENT$RANGE$FLOAT, .NUMBER$FLOAT) ;
519  2      CALL REAL$DIVIDE (.NUMBER$FLOAT, .TWO$FIVE$FIVE$FLOAT) ;
520  2      CALL STORE$RESULT (.CURRENT$INCREMENT$FLOAT) ;
521  2      CURRENT$INCREMENT$SET = TRUE ;
522  2      CALL CLEAR$ERROR$FIELD ;
523  2      CALL CLEAR$PROMPT$FIELD ;
524  2      CALL DISPLAY$CURRENT$RANGE ;
525  2      CALL DISPLAY$CURRENT$INCREMENT ;
526  2  END SET$CURRENT$INCREMENT ;
```

```
            $EJECT
                /* INITIALISE SYSTEM */

527   1         CALL RESET$SYSTEM ;
```

```
$EJECT

    /* USER MUST ENTER ORIGIN AND CURRENT RANGE */

528  1      CALL POSITION$CURSOR (LIMITS$WARNING$FIELD) ;
529  1      CALL WRITE$MESSAGE   (.LIMITS$WARNING$MESSAGE) ;
530  1      CALL POSITION$CURSOR (PROG$FIELD) ;
531  1      CALL WRITE$MESSAGE (.('RESET', EOT)) ;
532  1      CALL SET$ORIGIN ;
533  1      DO WHILE NOT (CURRENT$INCREMENT$SET) ;
534  2          CALL SET$CURRENT$INCREMENT ;
535  2      END ;
```

```
          $EJECT

              /* COMMAND INTERPRETER */
536   1     A1: CALL POSITION$CURSOR (PROG$FIELD) ;
537   1         CALL WRITE$MESSAGE (.(DELETE$LINE,
                                    'MONITOR', EOT)) ;
538   1         CALL POSITION$CURSOR (COMMAND$FIELD) ;
539   1         CALL CONSOLE$OUTPUT (DELETE$LINE) ;
540   1         CALL CONSOLE$OUTPUT ('*') ;
541   1         CALL READ$LINE ;
542   1         CALL POSITION$CURSOR (ERROR$FIELD) ;
543   1         CALL CONSOLE$OUTPUT (DELETE$LINE) ;
544   1         BUFFER$POINTER = .EDIT$BUFFER ;
545   1         WORD$START = NEXT$CHARACTER ;
546   1         IF BUFFER = CARRIGE$RETURN
                THEN
547   1             GOTO A1 ;
548   1         WORD$END = NEXT$SPACE ;
549   1         IF COMPARE$WORDS (WORD$START, WORD$END, .('POSITION'), 8)
                THEN
550   1             DO ;
551   2                 CALL POSITION$CURSOR (PROG$FIELD) ;
552   2                 CALL WRITE$MESSAGE (.(DELETE$LINE,
                                            'POSITION', EOT )) ;
553   2                 CALL POSITION ;
554   2             END ;
555   1         IF COMPARE$WORDS (WORD$START, WORD$END, .('PATH'), 4)
                THEN
556   1             CALL PATH ;
557   1         IF COMPARE$WORDS (WORD$START, WORD$END, .('LIMITS'), 6)
                THEN
558   1             CALL SET$LIMITS ;
559   1         IF COMPARE$WORDS (WORD$START, WORD$END, .('PARAMETERS'), 10)
                THEN
560   1             CALL SET$PARAMETERS ;
561   1         IF COMPARE$WORDS (WORD$START, WORD$END, .('WELD'), 4)
                THEN
562   1             CALL WELD$SEQUENCE ;
563   1         IF COMPARE$WORDS (WORD$START, WORD$END, .('SEQUENCE'), 8)
                THEN
564   1             CALL SEQUENCE ;
565   1         IF COMPARE$WORDS (WORD$START, WORD$END, .('DUMP'), 4)
                THEN
566   1             CALL DUMP ;
567   1         IF COMPARE$WORDS (WORD$START, WORD$END, .('LOAD'), 4)
                THEN
568   1             CALL LOAD ;
569   1         GOTO A1 ;
570   1     END TIG$WELD ;
```

MODULE INFORMATION:

```
    CODE AREA SIZE     = 0B20H   2848D
    VARIABLE AREA SIZE = 00C9H    201D
    MAXIMUM STACK SIZE = 000AH     10D
```

```
    1275 LINES READ
    0 PROGRAM ERROR(S)

END OF PL/M-80 COMPILATION
```

```
ISIS-II PL/M-80 V3.1 COMPILATION OF MODULE SETPARAMETERSPROCEDURE
OBJECT MODULE PLACED IN :F1:WELD4.OBJ
COMPILER INVOKED BY:  PLM80 :F1:WELD4.PLM DEBUG WORKFILES(:F1:,:F1:) PRINT(:F1:WELD4.LST)


           $TITLE('SET$PARAMETERS$PROCEDURE  VER 3.00  DATE 01/08/81')
  1        SET$PARAMETERS$PROCEDURE: DO ;

           /* A PLM80 PROCEDURE TO ALLOW THE USER TO ENTER GROUPS OF WELDING PARAMETERS.
              EACH GROUP CONTAINS VALUES FOR :-
                                              PULSE CURRENT
                                              BACKGROUND CURRENT
                                              PULSE ON TIME
                                              PULSE OFF TIME
                                              WELDING SPEED
                                              PRE-TRAVERSE TIME
                                              FALL TIME
              UP TO 256 GROUPS OF PARAMETERS MAY BE ENTERED.
              EACH GROUP IS IDENTIFIED BY ITS GROUP$NUMBER.
              A GROUP MAY ONLY BE ENTERED IF ALL LOWER NUMBER GROPUPS HAVE BEEN ENTERED.
           */
```

```
                $EJECT
                /* EXTERNALS */
                $INCLUDE (:F1:WELD.EQU)
            =   $NOLIST
            =
                $INCLUDE (:F1:WELD1.EDS)
            =   $NOLIST
 79   1         REAL$SUBTRACT: PROCEDURE (B$ADDRESS, A$ADDRESS) EXTERNAL ;
 80   2             DECLARE (A$ADDRESS, B$ADDRESS) ADDRESS ;
 81   2         END REAL$SUBTRACT ;

 82   1         REAL$DIVIDE: PROCEDURE (B$ADDRESS, A$ADDRESS) EXTERNAL ;
 83   2             DECLARE (A$ADDRESS, B$ADDRESS) ADDRESS ;
 84   2         END REAL$DIVIDE ;

 85   1         REAL$MULTIPLY: PROCEDURE (B$ADDRESS, A$ADDRESS) EXTERNAL ;
 86   2             DECLARE (A$ADDRESS, B$ADDRESS) ADDRESS ;
 87   2         END REAL$MULTIPLY ;

 88   1         FIX$16$BIT: PROCEDURE (A$ADDRESS) EXTERNAL ;
 89   2             DECLARE A$ADDRESS ADDRESS ;
 90   2         END FIX$16$BIT ;

 91   1         STORE$RESULT$SINGLE: PROCEDURE (STORE$ADDRESS) EXTERNAL ;
 92   2             DECLARE STORE$ADDRESS ADDRESS ;
 93   2         END STORE$RESULT$SINGLE ;

 94   1         FLOAT$SINGLE: PROCEDURE (A$ADDRESS) EXTERNAL ;
 95   2             DECLARE A$ADDRESS ADDRESS ;
 96   2         END FLOAT$SINGLE ;

 97   1         FIX$8$BIT: PROCEDURE (A$ADDRESS) EXTERNAL ;
 98   2             DECLARE A$ADDRESS ADDRESS ;
 99   2         END FIX$8$BIT ;

100   1         FIX$SINGLE$ROUND: PROCEDURE (A$ADDRESS) EXTERNAL ;
101   2             DECLARE A$ADDRESS ADDRESS ;
102   2         END FIX$SINGLE$ROUND ;

103   1         STORE$NUMBER: PROCEDURE (DESTINATION$ADDRESS, SOURCE$ADDRESS) EXTERNAL ;
104   2             DECLARE ( DESTINATION$ADDRESS, SOURCE$ADDRESS) ADDRESS ;
105   2         END STORE$NUMBER ;

106   1         STORE$RESULT: PROCEDURE (STORE$ADDRESS) EXTERNAL ;
107   2             DECLARE STORE$ADDRESS ADDRESS ;
108   2         END STORE$RESULT ;

109   1         STORE$RESULT$SINGLE$ADDRESS: PROCEDURE (STORE$ADDRESS) EXTERNAL ;
110   2             DECLARE STORE$ADDRESS ADDRESS ;
111   2         END STORE$RESULT$SINGLE$ADDRESS ;

112   1         DECLARE RESULT                (4) BYTE EXTERNAL ;
113   1         DECLARE RESULT$SINGLE         (2) BYTE EXTERNAL ;
114   1         DECLARE (TEN$THOUSAND$FLOAT,
                       TWO$FIVE$FIVE$FLOAT,
                       ONE$MILLION$FLOAT,
                       ZERO$FLOAT          ) (4) BYTE EXTERNAL ;
```

```
                /* PUBLIC DATA */

115  1          DECLARE PARAMETER$GROUP$TABLE (256) STRUCTURE (PULSE$CURRENT$FLOAT      (4) BYTE,
                                                   BACKGROUND$CURRENT$FLOAT (4) BYTE,
                                                   PULSE$ON$TIME$FLOAT      (4) BYTE,
                                                   PULSE$OFF$TIME$FLOAT     (4) BYTE,
                                                   WELDING$DIVISOR$FLOAT    (4) BYTE,
                                                   PRE$TRAVERSE$TIME$FLOAT  (4) BYTE,
                                                   FALL$TIME$FLOAT          (4) BYTE,
                                                   R$0$DIVISOR$SINGLE       (2) BYTE,
                                                   R$1$DIVISOR$SINGLE       (2) BYTE,
                                                   R$2$DIVISOR$SINGLE       (2) BYTE,
                                                   R$3$DIVISOR$SINGLE       (2) BYTE,
                                                   TOUCH$START                 BYTE,
                                                   WIRE$FEED                   BYTE )PUBLIC ;
116  1          DECLARE GROUP$NUMBER BYTE PUBLIC ;
117  1          DECLARE PARAMETERS$SET$FLAG (256) BYTE PUBLIC ;
118  1          DECLARE BYTE$VALUE BYTE PUBLIC ;
119  1          DECLARE (PULSE$ON$TIME$FLOAT,
                        PULSE$OFF$TIME$FLOAT,
                        FALL$TIME$FLOAT,
                        PRE$TRAVERSE$TIME$FLOAT,
                        WELDING$SPEED$FLOAT,
                        PULSE$CURRENT$FLOAT,
                        BACKGROUND$CURRENT$FLOAT,
                        CURRENT$INCREMENT$FLOAT,
                        CURRENT$RANGE$FLOAT        ) (4) BYTE PUBLIC ;
120  1          DECLARE  MAX$WELDING$SPEED$FLOAT      (4) BYTE PUBLIC
                        DATA (TEN$REAL) ;
121  1          DECLARE  MIN$WELDING$SPEED$FLOAT      (4) BYTE PUBLIC
                        DATA (POINT$ZERO$ZERO$SEVEN$EIGHT$REAL) ;
122  1          DECLARE MAX$R$SPEED$FLOAT             (4) BYTE PUBLIC
                       DATA (ONE$HUNDRED$REAL) ;
123  1          DECLARE MIN$R$SPEED$FLOAT             (4) BYTE PUBLIC
                       DATA (POINT$ZERO$ZERO$SEVEN$EIGHT$REAL) ;
124  1          DECLARE  MAX$PULSE$ON$TIME$FLOAT      (4) BYTE PUBLIC
                        DATA (TEN$REAL) ;
125  1          DECLARE  MIN$PULSE$ON$TIME$FLOAT      (4) BYTE PUBLIC
                        DATA (ZERO$REAL) ;
126  1          DECLARE  MAX$PULSE$OFF$TIME$FLOAT     (4) BYTE PUBLIC
                        DATA (TEN$REAL) ;
127  1          DECLARE  MIN$PULSE$OFF$TIME$FLOAT     (4) BYTE PUBLIC
                        DATA (ZERO$REAL) ;
128  1          DECLARE  MAX$FALL$TIME$FLOAT         (4) BYTE PUBLIC
                        DATA (ONE$HUNDRED$REAL) ;
129  1          DECLARE  MIN$FALL$TIME$FLOAT         (4) BYTE PUBLIC
                        DATA (ZERO$REAL) ;
130  1          DECLARE  MAX$PRE$TRAVERSE$TIME$FLOAT  (4) BYTE PUBLIC
                        DATA (ONE$HUNDRED$REAL) ;
131  1          DECLARE  MIN$PRE$TRAVERSE$TIME$FLOAT  (4) BYTE PUBLIC
                        DATA (ZERO$REAL) ;
132  1          DECLARE  MAX$PULSE$CURRENT$FLOAT      (4) BYTE PUBLIC
                        DATA(ONE$THOUSAND$REAL) ;
133  1          DECLARE  MIN$PULSE$CURRENT$FLOAT      (4) BYTE PUBLIC
                        DATA (ZERO$REAL) ;
134  1          DECLARE  MAX$BACKGROUND$CURRENT$FLOAT (4) BYTE PUBLIC
                        DATA (ONE$THOUSAND$REAL) ;
```

```
135  1          DECLARE  MIN$BACKGROUND$CURRENT$FLOAT (4) BYTE PUBLIC
                        DATA (ZERO$REAL) ;

                /* LOCAL DATA */
136  1          DECLARE (OVER$RANGE,
                        UNDER$RANGE,
                        FLAG,
                        I     ) BYTE ;
137  1          DECLARE BLANK$DATA (*) BYTE
                       DATA ('            ',
                             CURSOR$LEFT,
                             CURSOR$LEFT,
                             CURSOR$LEFT,
                             CURSOR$LEFT,
                             CURSOR$LEFT,
                             CURSOR$LEFT,
                             CURSOR$LEFT,
                             CURSOR$LEFT,
                             CURSOR$LEFT,
                             CURSOR$LEFT,
                             CURSOR$LEFT,
                             CURSOR$LEFT,
                             CURSOR$LEFT,
                             CURSOR$LEFT,
                             CURSOR$LEFT,
                             EOT           ) ;
```

```
                $EJECT
138   1           READ$BYTE$VALUE: PROCEDURE BYTE PUBLIC ;
139   2               DECLARE (ERROR$STATUS,
                               ENTRY$STATUS,
                               CHARACTER    ) BYTE ;
140   2               DO FOREVER ;
141   3                   ERROR$STATUS,
                          ENTRY$STATUS  = FALSE ;
142   3                   BYTE$VALUE = 0 ;
143   3                   CALL POSITION$CURSOR (COMMAND$FIELD) ;
144   3                   CALL CONSOLE$OUTPUT  (DELETE$LINE) ;
145   3                   CALL CONSOLE$OUTPUT ('#') ;
146   3                   DO WHILE NOT (ERROR$STATUS) ;
147   4                       CHARACTER = CONSOLE$INPUT ;
148   4                       IF CHARACTER = CARRIGE$RETURN
                              THEN
149   4                           DO ;
150   5                               CALL CLEAR$ERROR$FIELD ;
151   5                               RETURN ENTRY$STATUS ;
152   5                           END;
                              ELSE
153   4                            CALL CONSOLE$OUTPUT (CHARACTER) ;
154   4                       IF ((CHARACTER > '9') OR (CHARACTER < '0'))
                              THEN
155   4                           ERROR$STATUS = TRUE ;
156   4                       IF BYTE$VALUE > 25
                              THEN
157   4                       ERROR$STATUS = TRUE ;
158   4                       IF ((BYTE$VALUE = 25) AND (CHARACTER > '5'))
                              THEN
159   4                           ERROR$STATUS = TRUE ;
160   4                       BYTE$VALUE = (BYTE$VALUE * 10) + (CHARACTER - '0') ;
161   4                       ENTRY$STATUS = TRUE ;
162   4                   END ;
163   3                   CALL POSITION$CURSOR (ERROR$FIELD) ;
164   3                   CALL WRITE$MESSAGE (.(DELETE$LINE,
                                               'BYTE VALUE ERROR', EOT)) ;
165   3               END ;
166   2           END READ$BYTE$VALUE ;

167   1           DISPLAY$BYTE: PROCEDURE (BYTE$VALUE) PUBLIC ;
168   2               DECLARE BYTE$VALUE BYTE ;
169   2               DECLARE TEMP$REG (3) BYTE ;
170   2               DECLARE I BYTE ;
171   2               DO I = 0 TO 2 ;
172   3                   TEMP$REG (2-I) = BYTE$VALUE MOD 10 ;
173   3                   BYTE$VALUE = BYTE$VALUE / 10 ;
174   3               END ;
175   2               DO I = 0 TO 2 ;
176   3                   CALL CONSOLE$OUTPUT (TEMP$REG(I) + '0') ;
177   3               END ;
178   2           END DISPLAY$BYTE ;
```

```
                $EJECT
179   1         OVER$RANGE$ERROR: PROCEDURE PUBLIC ;
180   2             CALL POSITION$CURSOR (ERROR$FIELD) ;
181   2             CALL WRITE$MESSAGE (.OVER$RANGE$MESSAGE) ;
182   2             OVER$RANGE = TRUE ;
183   2         END OVER$RANGE$ERROR ;

184   1         UNDER$RANGE$ERROR: PROCEDURE PUBLIC ;
185   2             CALL POSITION$CURSOR (ERROR$FIELD) ;
186   2             CALL WRITE$MESSAGE (.UNDER$RANGE$MESSAGE) ;
187   2             UNDER$RANGE = TRUE ;
188   2         END UNDER$RANGE$ERROR ;
```

```
                $EJECT
189   1         DISPLAY$R$SPEED: PROCEDURE (R$ID) ;
190   2             DECLARE (R$ID) BYTE ;
191   2             DO CASE R$ID ;
192   3                 DO ;
193   4                     CALL FLOAT$SINGLE (.PARAMETER$GROUP$TABLE(GROUP$NUMBER).R$0$DIVISOR$SINGLE) ;
194   4                     CALL POSITION$CURSOR (R$0$SPEED$FIELD) ;
195   4                 END ;
196   3                 DO ;
197   4                     CALL FLOAT$SINGLE (.PARAMETER$GROUP$TABLE(GROUP$NUMBER).R$1$DIVISOR$SINGLE) ;
198   4                     CALL POSITION$CURSOR (R$1$SPEED$FIELD) ;
199   4                 END ;
200   3                 DO ;
201   4                     CALL FLOAT$SINGLE (.PARAMETER$GROUP$TABLE(GROUP$NUMBER).R$2$DIVISOR$SINGLE) ;
202   4                     CALL POSITION$CURSOR (R$2$SPEED$FIELD) ;
203   4                 END ;
204   3                 DO ;
205   4                     CALL FLOAT$SINGLE (.PARAMETER$GROUP$TABLE(GROUP$NUMBER).R$3$DIVISOR$SINGLE) ;
206   4                     CALL POSITION$CURSOR (R$3$SPEED$FIELD) ;
207   4                 END ;
208   3             END ;
209   2             CALL REAL$DIVIDE (.ONE$MILLION$FLOAT, .RESULT) ;
210   2             CALL REAL$DIVIDE (.RESULT, .MAIN$DIVISOR$FLOAT) ;
211   2             CALL DISPLAY$REAL$NUMBER (.RESULT, 4, 1) ;
212   2         END DISPLAY$R$SPEED ;
```

```
                $EJECT
213   1         DISPLAY$WIRE$FEED: PROCEDURE ;

214   2             CALL POSITION$CURSOR (WIRE$FIELD) ;
215   2             IF PARAMETER$GROUP$TABLE(GROUP$NUMBER).WIRE$FEED
                    THEN
216   2                 CALL WRITE$MESSAGE (.('ON ', EOT)) ;
                    ELSE
217   2                 CALL WRITE$MESSAGE (.('OFF', EOT)) ;
218   2         END DISPLAY$WIRE$FEED ;

219   1         CLEAR$WIRE$FIELD: PROCEDURE PUBLIC ;
220   2             CALL POSITION$CURSOR (WIRE$FIELD) ;
221   2             CALL WRITE$MESSAGE (.BLANK$DATA) ;
222   2         END CLEAR$WIRE$FIELD ;

223   1         DISPLAY$GROUP$NUMBER: PROCEDURE ;

224   2             CALL POSITION$CURSOR (GROUP$NUMBER$FIELD) ;
225   2             CALL DISPLAY$BYTE (GROUP$NUMBER) ;
226   2         END DISPLAY$GROUP$NUMBER ;

227   1         CLEAR$R$FIELD: PROCEDURE (R$ID) PUBLIC ;
228   2             DECLARE R$ID BYTE ;
229   2             DO CASE R$ID ;
230   3                 CALL POSITION$CURSOR (R$0$SPEED$FIELD) ;
231   3                 CALL POSITION$CURSOR (R$1$SPEED$FIELD) ;
232   3                 CALL POSITION$CURSOR (R$2$SPEED$FIELD) ;
233   3                 CALL POSITION$CURSOR (R$3$SPEED$FIELD) ;
234   3             END ;
235   2             CALL WRITE$MESSAGE (.BLANK$DATA) ;
236   2         END CLEAR$R$FIELD ;

237   1         CLEAR$WELDING$SPEED$FIELD: PROCEDURE PUBLIC ;
238   2             CALL POSITION$CURSOR (WELDING$SPEED$FIELD) ;
239   2             CALL WRITE$MESSAGE (.BLANK$DATA) ;
240   2         END CLEAR$WELDING$SPEED$FIELD ;

241   1         CLEAR$BACKGROUND$CURRENT$FIELD: PROCEDURE PUBLIC ;
242   2             CALL POSITION$CURSOR (BACKGROUND$CURRENT$FIELD) ;
243   2             CALL WRITE$MESSAGE (.BLANK$DATA) ;
244   2         END CLEAR$BACKGROUND$CURRENT$FIELD ;

245   1         CLEAR$PULSE$CURRENT$FIELD: PROCEDURE PUBLIC ;
246   2             CALL POSITION$CURSOR (PULSE$CURRENT$FIELD) ;
247   2             CALL WRITE$MESSAGE (.BLANK$DATA) ;
248   2         END CLEAR$PULSE$CURRENT$FIELD ;

249   1         CLEAR$PULSE$ON$TIME$FIELD: PROCEDURE PUBLIC ;
250   2             CALL POSITION$CURSOR (PULSE$ON$TIME$FIELD) ;
251   2             CALL WRITE$MESSAGE (.BLANK$DATA) ;
252   2         END CLEAR$PULSE$ON$TIME$FIELD ;

253   1         CLEAR$PULSE$OFF$TIME$FIELD: PROCEDURE PUBLIC ;
254   2             CALL POSITION$CURSOR (PULSE$OFF$TIME$FIELD) ;
255   2             CALL WRITE$MESSAGE (.BLANK$DATA) ;
256   2         END CLEAR$PULSE$OFF$TIME$FIELD ;
```

```
257  1          CLEAR$PRE$TRAVERSE$TIME$FIELD: PROCEDURE PUBLIC ;
258  2              CALL POSITION$CURSOR (PRE$TRAVERSE$TIME$FIELD) ;
259  2              CALL WRITE$MESSAGE (.BLANK$DATA) ;
260  2          END CLEAR$PRE$TRAVERSE$TIME$FIELD ;

261  1          CLEAR$FALL$TIME$FIELD: PROCEDURE PUBLIC ;
262  2              CALL POSITION$CURSOR (FALL$TIME$FIELD) ;
263  2              CALL WRITE$MESSAGE (.BLANK$DATA) ;
264  2          END CLEAR$FALL$TIME$FIELD ;

265  1          CLEAR$IGNITION$FIELD: PROCEDURE PUBLIC ;
266  2              CALL POSITION$CURSOR (IGNITION$FIELD) ;
267  2              CALL WRITE$MESSAGE (.BLANK$DATA) ;
268  2          END CLEAR$IGNITION$FIELD ;
```

```
                $EJECT
269   1         DISPLAY$IGNITION: PROCEDURE PUBLIC ;

270   2             CALL POSITION$CURSOR (IGNITION$FIELD) ;
271   2             IF PARAMETER$GROUP$TABLE(GROUP$NUMBER).TOUCH$START
                    THEN
272   2                 CALL WRITE$MESSAGE (.('TOUCH', EOT)) ;
                    ELSE
273   2                 CALL WRITE$MESSAGE (.('H$F  ', EOT)) ;
274   2         END DISPLAY$IGNITION ;

275   1         DISPLAY$WELDING$SPEED: PROCEDURE  PUBLIC ;

276   2             CALL POSITION$CURSOR (WELDING$SPEED$FIELD) ;
277   2             CALL REAL$DIVIDE (.TEN$THOUSAND$FLOAT,
                                .PARAMETER$GROUP$TABLE(GROUP$NUMBER).WELDING$DIVISOR$FLOAT) ;
278   2             CALL REAL$DIVIDE (.RESULT, .MAIN$DIVISOR$FLOAT) ;
279   2             CALL DISPLAY$REAL$NUMBER (.RESULT, 2, 4) ;
280   2             CALL WRITE$MESSAGE (.UNITS$MM$PER$SEC) ;
281   2         END DISPLAY$WELDING$SPEED ;

282   1         DISPLAY$PULSE$CURRENT: PROCEDURE  PUBLIC ;
283   2             CALL POSITION$CURSOR (PULSE$CURRENT$FIELD) ;
284   2             CALL DISPLAY$REAL$NUMBER (.PARAMETER$GROUP$TABLE(GROUP$NUMBER).PULSE$CURRENT$FLOAT, 4, 1) ;
285   2             CALL WRITE$MESSAGE (.UNITS$AMP) ;
286   2         END DISPLAY$PULSE$CURRENT ;

287   1         DISPLAY$BACKGROUND$CURRENT: PROCEDURE  PUBLIC ;

288   2             CALL POSITION$CURSOR (BACKGROUND$CURRENT$FIELD) ;
289   2             CALL DISPLAY$REAL$NUMBER (.PARAMETER$GROUP$TABLE(GROUP$NUMBER).BACKGROUND$CURRENT$FLOAT, 4, 1) ;
290   2             CALL WRITE$MESSAGE (.UNITS$AMP) ;
291   2         END DISPLAY$BACKGROUND$CURRENT ;

292   1         DISPLAY$PULSE$ON$TIME: PROCEDURE PUBLIC ;

293   2             CALL POSITION$CURSOR (PULSE$ON$TIME$FIELD) ;
294   2             CALL DISPLAY$REAL$NUMBER (.PARAMETER$GROUP$TABLE(GROUP$NUMBER).PULSE$ON$TIME$FLOAT, 2, 3) ;
295   2             CALL WRITE$MESSAGE (.UNITS$SEC) ;
296   2         END DISPLAY$PULSE$ON$TIME ;

297   1         DISPLAY$PULSE$OFF$TIME: PROCEDURE PUBLIC ;

298   2             CALL POSITION$CURSOR (PULSE$OFF$TIME$FIELD) ;
299   2             CALL DISPLAY$REAL$NUMBER (.PARAMETER$GROUP$TABLE(GROUP$NUMBER).PULSE$OFF$TIME$FLOAT, 2, 3) ;
300   2             CALL WRITE$MESSAGE (.UNITS$SEC) ;
301   2         END DISPLAY$PULSE$OFF$TIME ;

302   1         DISPLAY$FALL$TIME: PROCEDURE PUBLIC ;

303   2             CALL POSITION$CURSOR (FALL$TIME$FIELD) ;
304   2             CALL DISPLAY$REAL$NUMBER (.PARAMETER$GROUP$TABLE(GROUP$NUMBER).FALL$TIME$FLOAT, 3, 3) ;
305   2             CALL WRITE$MESSAGE (.UNITS$SEC) ;
306   2         END DISPLAY$FALL$TIME ;

307   1         DISPLAY$PRE$TRAVERSE$TIME: PROCEDURE PUBLIC ;
```

```
308  2            CALL POSITION$CURSOR (PRE$TRAVERSE$TIME$FIELD) ;
309  2            CALL DISPLAY$REAL$NUMBER (.PARAMETER$GROUP$TABLE(GROUP$NUMBER).PRE$TRAVERSE$TIME$FLOAT, 3, 3) ;
310  2            CALL WRITE$MESSAGE (.UNITS$SEC) ;
311  2         END DISPLAY$PRE$TRAVERSE$TIME ;
```

```
                $EJECT
312   1         DISPLAY$PARAMETER$GROUP: PROCEDURE PUBLIC ;

313   2             CALL DISPLAY$GROUP$NUMBER ;
314   2             CALL CLEAR$PULSE$CURRENT$FIELD ;
315   2             CALL CLEAR$BACKGROUND$CURRENT$FIELD ;
316   2             CALL CLEAR$PULSE$ON$TIME$FIELD ;
317   2             CALL CLEAR$PULSE$OFF$TIME$FIELD ;
318   2             CALL CLEAR$WELDING$SPEED$FIELD ;
319   2             CALL CLEAR$PRE$TRAVERSE$TIME$FIELD ;
320   2             CALL CLEAR$FALL$TIME$FIELD ;
321   2             CALL CLEAR$IGNITION$FIELD ;
322   2             CALL CLEAR$WIRE$FIELD ;
323   2             CALL CLEAR$R$FIELD (0) ;
324   2             CALL CLEAR$R$FIELD (1) ;
325   2             CALL CLEAR$R$FIELD (2) ;
326   2             CALL CLEAR$R$FIELD (3) ;
327   2             IF PARAMETERS$SET$FLAG(GROUP$NUMBER)
                    THEN
328   2                 DO ;
329   3                     CALL DISPLAY$PULSE$CURRENT ;
330   3                     CALL DISPLAY$BACKGROUND$CURRENT ;
331   3                     CALL DISPLAY$PULSE$ON$TIME ;
332   3                     CALL DISPLAY$PULSE$OFF$TIME ;
333   3                     CALL DISPLAY$WELDING$SPEED ;
334   3                     CALL DISPLAY$PRE$TRAVERSE$TIME ;
335   3                     CALL DISPLAY$FALL$TIME ;
336   3                     CALL DISPLAY$IGNITION ;
337   3                     CALL DISPLAY$WIRE$FEED ;
338   3                     CALL DISPLAY$R$SPEED (0) ;
339   3                     CALL DISPLAY$R$SPEED (1) ;
340   3                     CALL DISPLAY$R$SPEED (2) ;
341   3                     CALL DISPLAY$R$SPEED (3) ;
342   3                 END ;
343   2         END DISPLAY$PARAMETER$GROUP ;
```

```
                $EJECT
344   1           SET$IGNITION: PROCEDURE BYTE PUBLIC ;
345   2               DECLARE CHARACTER    BYTE ;

346   2               CALL POSITION$CURSOR (PROMPT$FIELD) ;
347   2               CALL WRITE$MESSAGE (.(DELETE$LINE,
                                        'TOUCH START ?', EOT)) ;
348   2               CHARACTER = CONSOLE$INPUT ;
349   2               DO WHILE ((CHARACTER <> 'Y') AND (CHARACTER <> 'N') AND (CHARACTER <> CARRIGE$RETURN)) ;
350   3                   CHARACTER = CONSOLE$INPUT ;
351   3               END ;
352   2               IF CHARACTER = CARRIGE$RETURN
                      THEN
353   2                   RETURN TRUE ;
354   2               IF CHARACTER = 'Y'
                      THEN
355   2                   PARAMETER$GROUP$TABLE(GROUP$NUMBER).TOUCH$START = TRUE ;
                      ELSE
356   2                   PARAMETER$GROUP$TABLE(GROUP$NUMBER).TOUCH$START = FALSE ;
357   2               CALL DISPLAY$IGNITION ;
358   2               RETURN TRUE ;
359   2           END SET$IGNITION ;

360   1           SET$WIRE$FEED: PROCEDURE BYTE PUBLIC ;
361   2               DECLARE CHARACTER BYTE ;

362   2               CALL POSITION$CURSOR (PROMPT$FIELD) ;
363   2               CALL WRITE$MESSAGE (. (DELETE$LINE,
                                        'WIRE FEED ?', EOT)) ;
364   2               CHARACTER = CONSOLE$INPUT ;
365   2               DO WHILE ( (CHARACTER <> 'Y') AND (CHARACTER <> 'N') AND  (CHARACTER <> CARRIGE$RETURN) ) ;
366   3                   CHARACTER = CONSOLE$INPUT ;
367   3               END ;
368   2               IF CHARACTER = CARRIGE$RETURN
                      THEN
369   2                   RETURN TRUE ;
370   2               IF CHARACTER = 'Y'
                      THEN
371   2                   PARAMETER$GROUP$TABLE(GROUP$NUMBER).WIRE$FEED = TRUE ;
                      ELSE
372   2                   PARAMETER$GROUP$TABLE(GROUP$NUMBER).WIRE$FEED = FALSE ;
373   2               CALL DISPLAY$WIRE$FEED ;
374   2               RETURN TRUE ;
375   2           END SET$WIRE$FEED ;
```

```
                $EJECT
376   1         SET$WELDING$SPEED: PROCEDURE BYTE ;

377   2           CALL POSITION$CURSOR (PROMPT$FIELD) ;
378   2           CALL WRITE$MESSAGE (.(DELETE$LINE,
                                  'ENTER WELDING SPEED IN mm/Sec', EOT)) ;
379   2           OVER$RANGE,
                  UNDER$RANGE = TRUE ;
380   2           DO WHILE (OVER$RANGE OR UNDER$RANGE) ;
381   3             IF NOT (READ$REAL$NUMBER)
                    THEN
382   3               RETURN FALSE ;
383   3             CALL REAL$SUBTRACT (.MAX$WELDING$SPEED$FLOAT, .NUMBER$FLOAT) ;
384   3             IF RESULT$NEGATIVE
                    THEN
385   3               CALL OVER$RANGE$ERROR ;
                    ELSE
386   3               DO ;
387   4                 OVER$RANGE = FALSE ;
388   4                 CALL REAL$SUBTRACT (.NUMBER$FLOAT, .MIN$WELDING$SPEED$FLOAT) ;
389   4                 IF RESULT$NEGATIVE
                        THEN
390   4                   CALL UNDER$RANGE$ERROR ;
                        ELSE
391   4                   UNDER$RANGE = FALSE ;
392   4                 END ;
393   3             END ;
394   2           CALL REAL$DIVIDE (.TEN$THOUSAND$FLOAT, .NUMBER$FLOAT) ;
395   2           CALL REAL$DIVIDE (.RESULT, .MAIN$DIVISOR$FLOAT) ;
396   2           CALL STORE$RESULT (.PARAMETER$GROUP$TABLE(GROUP$NUMBER).WELDING$DIVISOR$FLOAT) ;
397   2           CALL CLEAR$ERROR$FIELD ;
398   2           CALL CLEAR$PROMPT$FIELD ;
399   2           CALL DISPLAY$WELDING$SPEED ;
400   2           RETURN TRUE ;
401   2         END SET$WELDING$SPEED ;
```

```
                $EJECT
402   1           SET$R$SPEED: PROCEDURE (R$ID) BYTE ;
403   2               DECLARE R$ID BYTE ;

404   2               CALL POSITION$CURSOR (PROMPT$FIELD) ;
405   2               CALL WRITE$MESSAGE (.(DELETE$LINE,
                                        'ENTER R', EOT)) ;
406   2               DO CASE R$ID ;
407   3                   CALL CONSOLE$OUTPUT ('0') ;
408   3                   CALL CONSOLE$OUTPUT ('1') ;
409   3                   CALL CONSOLE$OUTPUT ('2') ;
410   3                   CALL CONSOLE$OUTPUT ('3') ;
411   3               END ;
412   2               CALL WRITE$MESSAGE (.(' SPEED IN S/Sec', EOT)) ;
413   2               OVER$RANGE,

                      UNDER$RANGE = TRUE ;
414   2               DO WHILE (OVER$RANGE OR UNDER$RANGE) ;
415   3                   IF NOT (READ$REAL$NUMBER)
                          THEN
416   3                       RETURN FALSE ;
417   3                   CALL REAL$SUBTRACT (.MAX$R$SPEED$FLOAT, .NUMBER$FLOAT) ;
418   3                   IF RESULT$NEGATIVE
                          THEN
419   3                       CALL OVER$RANGE$ERROR ;
                          ELSE
420   3                       DO ;
421   4                           OVER$RANGE = FALSE ;
422   4                           CALL REAL$SUBTRACT (.NUMBER$FLOAT, .MIN$R$SPEED$FLOAT) ;
423   4                           IF RESULT$NEGATIVE
                                  THEN
424   4                               CALL UNDER$RANGE$ERROR ;
                                  ELSE
425   4                               UNDER$RANGE = FALSE ;
426   4                       END ;
427   3               END ;
428   2               CALL REAL$DIVIDE (.ONE$MILLION$FLOAT, .NUMBER$FLOAT) ;
429   2               CALL REAL$DIVIDE (.RESULT, .MAIN$DIVISOR$FLOAT) ;
430   2               CALL FIX$16$BIT (.RESULT) ;
431   2               DO CASE R$ID ;
432   3                   CALL STORE$RESULT$SINGLE (.PARAMETER$GROUP$TABLE(GROUP$NUMBER).R$0$DIVISOR$SINGLE) ;
433   3                   CALL STORE$RESULT$SINGLE (.PARAMETER$GROUP$TABLE(GROUP$NUMBER).R$1$DIVISOR$SINGLE) ;
434   3                   CALL STORE$RESULT$SINGLE (.PARAMETER$GROUP$TABLE(GROUP$NUMBER).R$2$DIVISOR$SINGLE) ;
435   3                   CALL STORE$RESULT$SINGLE (.PARAMETER$GROUP$TABLE(GROUP$NUMBER).R$3$DIVISOR$SINGLE) ;
436   3               END ;
437   2               CALL CLEAR$ERROR$FIELD ;
438   2               CALL CLEAR$PROMPT$FIELD ;
439   2               CALL DISPLAY$R$SPEED (R$ID) ;
440   2               RETURN TRUE ;
441   2           END SET$R$SPEED ;
```

```
                $EJECT
442   1         SET$PULSE$CURRENT: PROCEDURE BYTE ;

443   2           CALL POSITION$CURSOR (PROMPT$FIELD) ;
444   2           CALL WRITE$MESSAGE (.(DELETE$LINE,
                                  'ENTER PULSECURRENT IN Amps', EOT)) ;
445   2           OVER$RANGE,
                  UNDER$RANGE = TRUE ;
446   2           DO WHILE (OVER$RANGE OR UNDER$RANGE) ;
447   3             IF NOT (READ$REAL$NUMBER)
                    THEN
448   3                 RETURN FALSE ;
449   3               CALL REAL$SUBTRACT (.MAX$PULSE$CURRENT$FLOAT, .NUMBER$FLOAT) ;
450   3               IF RESULT$NEGATIVE
                      THEN
451   3                   CALL OVER$RANGE$ERROR ;
                      ELSE
452   3                   DO ;
453   4                     OVER$RANGE = FALSE ;
454   4                     CALL REAL$SUBTRACT (.NUMBER$FLOAT, .MIN$PULSE$CURRENT$FLOAT) ;
455   4                     IF RESULT$NEGATIVE
                            THEN
456   4                         CALL UNDER$RANGE$ERROR ;
                            ELSE
457   4                         UNDER$RANGE = FALSE ;
458   4                   END ;
459   3           END ;
460   2           CALL STORE$NUMBER (.PARAMETER$GROUP$TABLE(GROUP$NUMBER).PULSE$CURRENT$FLOAT,
                                     .NUMBER$FLOAT) ;

461   2           CALL CLEAR$ERROR$FIELD ;
462   2           CALL CLEAR$PROMPT$FIELD ;
463   2           CALL DISPLAY$PULSE$CURRENT ;
464   2           RETURN TRUE ;
465   2         END SET$PULSE$CURRENT ;
```

```
                $EJECT
466   1         SET$BACKGROUND$CURRENT: PROCEDURE BYTE ;

467   2             CALL POSITION$CURSOR (PROMPT$FIELD) ;
468   2             CALL WRITE$MESSAGE (.(DELETE$LINE,
                                  'ENTER BACKGROUND CURRENT IN Amps', EOT)) ;
469   2             OVER$RANGE,
                    UNDER$RANGE = TRUE ;
470   2             DO WHILE (OVER$RANGE OR UNDER$RANGE) ;
471   3                 IF NOT (READ$REAL$NUMBER)
                        THEN
472   3                     RETURN FALSE ;
473   3                 CALL REAL$SUBTRACT (.MAX$BACKGROUND$CURRENT$FLOAT, .NUMBER$FLOAT) ;
474   3                 IF RESULT$NEGATIVE
                        THEN
475   3                     CALL OVER$RANGE$ERROR ;
                        ELSE
476   3                     DO ;
477   4                         OVER$RANGE = FALSE ;
478   4                         CALL REAL$SUBTRACT (.NUMBER$FLOAT, .MIN$BACKGROUND$CURRENT$FLOAT) ;
479   4                         IF RESULT$NEGATIVE
                                THEN
480   4                             CALL UNDER$RANGE$ERROR ;
                                ELSE
481   4                             UNDER$RANGE = FALSE ;
482   4                     END ;
483   3             END ;
484   2             CALL STORE$NUMBER (.PARAMETER$GROUP$TABLE(GROUP$NUMBER).BACKGROUND$CURRENT$FLOAT,
                                  .NUMBER$FLOAT) ;

485   2             CALL CLEAR$ERROR$FIELD ;
486   2             CALL CLEAR$PROMPT$FIELD ;
487   2             CALL DISPLAY$BACKGROUND$CURRENT ;
488   2             RETURN TRUE ;
489   2         END SET$BACKGROUND$CURRENT ;
```

```
                $EJECT
490   1         SET$PULSE$ON$TIME: PROCEDURE BYTE ;

491   2           CALL POSITION$CURSOR (PROMPT$FIELD) ;
492   2           CALL WRITE$MESSAGE (.(DELETE$LINE,
                                  'ENTER PULSE ON TIME IN Secs', EOT)) ;
493   2           OVER$RANGE,
                  UNDER$RANGE = TRUE ;
494   2           DO WHILE (OVER$RANGE OR UNDER$RANGE) ;
495   3             IF NOT (READ$REAL$NUMBER)
                    THEN
496   3               RETURN FALSE ;
497   3             CALL REAL$SUBTRACT (.MAX$PULSE$ON$TIME$FLOAT, .NUMBER$FLOAT) ;
498   3             IF RESULT$NEGATIVE
                    THEN
499   3               CALL OVER$RANGE$ERROR ;
                    ELSE
500   3               DO ;
501   4                 OVER$RANGE = FALSE ;
502   4                 CALL REAL$SUBTRACT (.NUMBER$FLOAT, .MIN$PULSE$ON$TIME$FLOAT) ;
503   4                 IF RESULT$NEGATIVE
                        THEN
504   4                   CALL UNDER$RANGE$ERROR ;
                        ELSE
505   4                   UNDER$RANGE = FALSE ;
506   4               END ;
507   3           END ;
508   2           CALL STORE$NUMBER (.PARAMETER$GROUP$TABLE(GROUP$NUMBER).PULSE$ON$TIME$FLOAT,
                                .NUMBER$FLOAT) ;

509   2           CALL CLEAR$ERROR$FIELD ;
510   2           CALL CLEAR$PROMPT$FIELD ;
511   2           CALL DISPLAY$PULSE$ON$TIME ;
512   2           RETURN TRUE ;
513   2         END SET$PULSE$ON$TIME ;
```

```
        $EJECT
514  1      SET$PULSE$OFF$TIME: PROCEDURE BYTE ;

515  2          CALL POSITION$CURSOR (PROMPT$FIELD) ;
516  2          CALL WRITE$MESSAGE (.(DELETE$LINE,
                                 'ENTER PULSE OFF TIME IN Secs', EOT)) ;
517  2          OVER$RANGE,
                UNDER$RANGE = TRUE ;
518  2          DO WHILE (OVER$RANGE OR UNDER$RANGE) ;
519  3              IF NOT (READ$REAL$NUMBER)
                    THEN
520  3                  RETURN FALSE ;
521  3              CALL REAL$SUBTRACT (.MAX$PULSE$OFF$TIME$FLOAT, .NUMBER$FLOAT) ;
522  3              IF RESULT$NEGATIVE
                    THEN
523  3                  CALL OVER$RANGE$ERROR ;
                    ELSE
524  3                  DO ;
525  4                      OVER$RANGE = FALSE ;
526  4                      CALL REAL$SUBTRACT (.NUMBER$FLOAT, .MIN$PULSE$OFF$TIME$FLOAT) ;
527  4                      IF RESULT$NEGATIVE
                            THEN
528  4                          CALL UNDER$RANGE$ERROR ;
                            ELSE
529  4                          UNDER$RANGE = FALSE ;
530  4                  END ;
531  3          END ;
532  2          CALL STORE$NUMBER (.PARAMETER$GROUP$TABLE(GROUP$NUMBER).PULSE$OFF$TIME$FLOAT,
                                  .NUMBER$FLOAT) ;

533  2          CALL CLEAR$ERROR$FIELD ;
534  2          CALL CLEAR$PROMPT$FIELD ;
535  2          CALL DISPLAY$PULSE$OFF$TIME ;
536  2          RETURN TRUE ;
537  2      END SET$PULSE$OFF$TIME ;
```

```
        $EJECT
538  1      SET$FALL$TIME: PROCEDURE BYTE ;

539  2          CALL POSITION$CURSOR (PROMPT$FIELD) ;
540  2          CALL WRITE$MESSAGE (.(DELETE$LINE,
                                  'ENTER FALL TIME IN Secs', EOT)) ;
541  2          OVER$RANGE,
                UNDER$RANGE = TRUE ;
542  2          DO WHILE (OVER$RANGE OR UNDER$RANGE) ;
543  3              IF NOT (READ$REAL$NUMBER)
                    THEN
544  3                  RETURN FALSE ;
545  3              CALL REAL$SUBTRACT (.MAX$FALL$TIME$FLOAT, .NUMBER$FLOAT) ;
546  3              IF RESULT$NEGATIVE
                    THEN
547  3                  CALL OVER$RANGE$ERROR ;
                    ELSE
548  3                  DO ;
549  4                      OVER$RANGE = FALSE ;
550  4                      CALL REAL$SUBTRACT (.NUMBER$FLOAT, .MIN$FALL$TIME$FLOAT) ;
551  4                      IF RESULT$NEGATIVE
                            THEN
552  4                          CALL UNDER$RANGE$ERROR ;
                            ELSE
553  4                          UNDER$RANGE = FALSE ;
554  4                  END ;
555  3          END ;
556  2          CALL STORE$NUMBER (.PARAMETER$GROUP$TABLE(GROUP$NUMBER).FALL$TIME$FLOAT,
                                  .NUMBER$FLOAT) ;

557  2          CALL CLEAR$ERROR$FIELD ;
558  2          CALL CLEAR$PROMPT$FIELD ;
559  2          CALL DISPLAY$FALL$TIME ;
560  2          RETURN TRUE ;
561  2      END SET$FALL$TIME ;
```

```
        $EJECT
562  1      SET$PRE$TRAVERSE$TIME: PROCEDURE BYTE ;

563  2          CALL POSITION$CURSOR (PROMPT$FIELD) ;
564  2          CALL WRITE$MESSAGE (.(DELETE$LINE,
                                  'ENTER PRE-TRAVERSE TIME IN Secs', EOT)) ;
565  2          OVER$RANGE,
                UNDER$RANGE = TRUE ;
566  2          DO WHILE (OVER$RANGE OR UNDER$RANGE) ;
567  3              IF NOT (READ$REAL$NUMBER)
                    THEN
568  3                  RETURN FALSE ;
569  3              CALL REAL$SUBTRACT (.MAX$PRE$TRAVERSE$TIME$FLOAT, .NUMBER$FLOAT) ;
570  3              IF RESULT$NEGATIVE
                    THEN
571  3                  CALL OVER$RANGE$ERROR ;
                    ELSE
572  3                  DO ;
573  4                      OVER$RANGE = FALSE ;
574  4                      CALL REAL$SUBTRACT (.NUMBER$FLOAT, .MIN$PRE$TRAVERSE$TIME$FLOAT) ;
575  4                      IF RESULT$NEGATIVE
                            THEN
576  4                          CALL UNDER$RANGE$ERROR ;
                            ELSE
577  4                          UNDER$RANGE = FALSE ;
578  4                  END ;
579  3          END ;
580  2          CALL STORE$NUMBER (.PARAMETER$GROUP$TABLE(GROUP$NUMBER).PRE$TRAVERSE$TIME$FLOAT,
                                  .NUMBER$FLOAT) ;

581  2          CALL CLEAR$ERROR$FIELD ;
582  2          CALL CLEAR$PROMPT$FIELD ;
583  2          CALL DISPLAY$PRE$TRAVERSE$TIME ;
584  2          RETURN TRUE ;
585  2      END SET$PRE$TRAVERSE$TIME ;
```

```
                $EJECT
586   1           SET$PARAMETERS: PROCEDURE PUBLIC ;
587   2               CALL POSITION$CURSOR (PROG$FIELD) ;
588   2               CALL WRITE$MESSAGE (.(DELETE$LINE,
                                      'EDIT PARAMETER GROUP', EOT)) ;
589   2           A10: CALL POSITION$CURSOR (PROMPT$FIELD) ;
590   2               CALL WRITE$MESSAGE (.(DELETE$LINE,
                                      'WHICH PARAMETER GROUP ?', EOT)) ;
591   2               CALL POSITION$CURSOR (COMMAND$FIELD) ;
592   2               CALL CONSOLE$OUTPUT (DELETE$LINE) ;
593   2               CALL CONSOLE$OUTPUT ('#') ;
594   2               IF NOT (READ$BYTE$VALUE)
                      THEN
595   2                   DO ;
596   3                       CALL CLEAR$PROMPT$FIELD ;
597   3                       CALL CLEAR$ERROR$FIELD ;
598   3                       RETURN ;
599   3                   END ;
600   2               FLAG = TRUE ;
601   2               IF BYTE$VALUE <> 0
                      THEN
602   2                   DO I = 0 TO (BYTE$VALUE - 1) ;
603   3                       FLAG = FLAG AND PARAMETERS$SET$FLAG(I) ;
604   3                   END ;
605   2               IF NOT FLAG
                      THEN
606   2                   GOTO A10 ;
607   2               GROUP$NUMBER = BYTE$VALUE ;
608   2               CALL DISPLAY$PARAMETER$GROUP ;
609   2               DO WHILE NOT (SET$PULSE$CURRENT
                                  OR PARAMETERS$SET$FLAG(GROUP$NUMBER)) ;
610   3               END ;
611   2               DO WHILE NOT (SET$BACKGROUND$CURRENT
                                  OR PARAMETERS$SET$FLAG(GROUP$NUMBER)) ;
612   3               END ;
613   2               DO WHILE NOT (SET$PULSE$ON$TIME
                                  OR PARAMETERS$SET$FLAG(GROUP$NUMBER)) ;
614   3               END ;
615   2               DO WHILE NOT (SET$PULSE$OFF$TIME
                                  OR PARAMETERS$SET$FLAG(GROUP$NUMBER)) ;
616   3               END ;
617   2               DO WHILE NOT (SET$WELDING$SPEED
                                  OR PARAMETERS$SET$FLAG(GROUP$NUMBER)) ;
618   3               END ;
619   2               DO WHILE NOT (SET$PRE$TRAVERSE$TIME
                                  OR PARAMETERS$SET$FLAG(GROUP$NUMBER)) ;
620   3               END ;
621   2               DO WHILE NOT (SET$FALL$TIME
                                  OR PARAMETERS$SET$FLAG(GROUP$NUMBER)) ;
622   3               END ;
623   2               DO WHILE NOT (SET$IGNITION
                                  OR PARAMETERS$SET$FLAG(GROUP$NUMBER)) ;
624   3               END ;
625   2               DO WHILE NOT (SET$WIRE$FEED
                                  OR PARAMETERS$SET$FLAG(GROUP$NUMBER)) ;
626   3               END ;
627   2               DO WHILE NOT (SET$R$SPEED(0) OR
```

```
                                PARAMETERS$SET$FLAG(GROUP$NUMBER)) ;
628   3             END ;
629   2             DO WHILE NOT (SET$R$SPEED(1) OR
                                PARAMETERS$SET$FLAG(GROUP$NUMBER)) ;
630   3             END ;
631   2             DO WHILE NOT (SET$R$SPEED(2) OR
                                PARAMETERS$SET$FLAG(GROUP$NUMBER)) ;
632   3             END ;
633   2             DO WHILE NOT (SET$R$SPEED(3) OR
                                PARAMETERS$SET$FLAG(GROUP$NUMBER)) ;
634   3             END ;
635   2             PARAMETERS$SET$FLAG(GROUP$NUMBER) = TRUE ;
636   2             GOTO A10 ;
637   2         END SET$PARAMETERS ;

638   1     END SET$PARAMETERS$PROCEDURE ;



MODULE INFORMATION:

    CODE AREA SIZE     = 0E6EH   3694D
    VARIABLE AREA SIZE = 2737H  10039D
    MAXIMUM STACK SIZE = 000AH     10D
    1191 LINES READ
    0 PROGRAM ERROR(S)

END OF PL/M-80 COMPILATION
H  10039D
    MAXIMUM S
```

```
ISIS-II PL/M-80 V3.1 COMPILATION OF MODULE POSITIONPROCEDURE
OBJECT MODULE PLACED IN :F1:WELD2.OBJ
COMPILER INVOKED BY:  PLM80 :F1:WELD2.PLM DEBUG WORKFILES(:F1:,:F1:) PRINT(:F1:WELD2.LST)


                $TITLE('POSITION$PROCEDURE     VER 3.00       DATE 03/07/81')
  1             POSITION$PROCEDURE: DO ;
                  /* A PLM80 ROUTINE TO ALLOW THE USER TO MOVE THE WELDING TORCH FROM THE
                     HAND HELD CONTROLER.
                */
```

```
            $EJECT
            /* EXTERNALS */
            $INCLUDE (:F1:WELD.EQU)
        =   $NOLIST
        =
            $INCLUDE (:F1:WELD1.EDS)
        =   $NOLIST

 79  1      ADD: PROCEDURE (B$ADDRESS, A$ADDRESS) EXTERNAL ;
 80  2          DECLARE (A$ADDRESS, B$ADDRESS) ADDRESS ;
 81  2      END ADD ;

 82  1      SUBTRACT: PROCEDURE (B$ADDRESS, A$ADDRESS) EXTERNAL ;
 83  2          DECLARE (A$ADDRESS, B$ADDRESS) ADDRESS ;
 84  2      END SUBTRACT ;

 85  1      ADD$SINGLE$TO$DOUBLE: PROCEDURE (DOUBLE$ADDRESS, SINGLE$ADDRESS) EXTERNAL ;
 86  2          DECLARE (DOUBLE$ADDRESS, SINGLE$ADDRESS) ADDRESS ;
 87  2      END ADD$SINGLE$TO$DOUBLE ;

 88  1      MULTIPLY$SINGLE: PROCEDURE (B$ADDRESS, A$ADDRESS) EXTERNAL ;
 89  2          DECLARE (A$ADDRESS, B$ADDRESS) ADDRESS ;
 90  2      END MULTIPLY$SINGLE ;

 91  1      DIVIDE$SINGLE: PROCEDURE (B$ADDRESS, A$ADDRESS) EXTERNAL ;
 92  2          DECLARE (A$ADDRESS, B$ADDRESS) ADDRESS ;
 93  2      END DIVIDE$SINGLE ;

 94  1      NEGATE$SINGLE: PROCEDURE (A$ADDRESS) EXTERNAL ;
 95  2          DECLARE A$ADDRESS ADDRESS ;
 96  2      END NEGATE$SINGLE ;

 97  1      STORE$RESULT: PROCEDURE (STORE$ADDRESS) EXTERNAL ;
 98  2          DECLARE STORE$ADDRESS ADDRESS ;
 99  2      END STORE$RESULT ;

100  1      STORE$RESULT$SINGLE: PROCEDURE (STORE$ADDRESS) EXTERNAL ;
101  2          DECLARE STORE$ADDRESS ADDRESS ;
102  2      END STORE$RESULT$SINGLE ;

103  1      STORE$RESULT$SINGLE$ADDRESS: PROCEDURE (STORE$ADDRESS) EXTERNAL ;
104  2          DECLARE STORE$ADDRESS ADDRESS ;
105  2      END STORE$RESULT$SINGLE$ADDRESS ;

106  1      STORE$NUMBER$SINGLE: PROCEDURE (STORE$ADDRESS, SOURCE$ADDRESS) EXTERNAL ;
107  2          DECLARE (STORE$ADDRESS, SOURCE$ADDRESS) ADDRESS ;
108  2      END STORE$NUMBER$SINGLE ;

109  1      STORE$NUMBER: PROCEDURE (STORE$ADDRESS, SOURCE$ADDRESS) EXTERNAL ;
110  2          DECLARE (STORE$ADDRESS, SOURCE$ADDRESS) ADDRESS ;
111  2      END STORE$NUMBER ;

112  1      DECLARE (RESULT,
                      TWO,
                      TEN,
                      ONE     )       (4) BYTE EXTERNAL ;
113  1      DECLARE (RESULT$SINGLE,
```

```
                                ONE$SINGLE,
                                MINUS$ONE$SINGLE,
                                FOUR$SINGLE,
                                FIVE$SINGLE )   (2) BYTE EXTERNAL ;
114  1          DECLARE (X$UPPER$LIMIT,
                                X$LOWER$LIMIT,
                                Y$UPPER$LIMIT,
                                Y$LOWER$LIMIT,
                                Z$UPPER$LIMIT,
                                Z$LOWER$LIMIT  )  (4) BYTE EXTERNAL ;
115  1          DECLARE DELAY$COUNT ADDRESS EXTERNAL ;

116  1          SET$INTERRUPT$MASK: PROCEDURE EXTERNAL ;
117  2          END SET$INTERRUPT$MASK ;
```

```
          $EJECT
            /* PUBLIC DATA */
118   1       DECLARE SPEED$STATUS BYTE PUBLIC ;
119   1       DECLARE MOTORS$RUNNING BYTE PUBLIC ;
120   1       DECLARE (R$0$ENABLED,
                                 R$1$ENABLED,
                                 R$2$ENABLED,
                                 R$3$ENABLED,
                                 CONTROLLER$ACTIVE,
                                 ON$SEAM            ) BYTE PUBLIC ;
121   1       DECLARE (X$ORDINATE,
                                 Y$ORDINATE,
                                 Z$ORDINATE    ) (4) BYTE PUBLIC ;
122   1       DECLARE  R$ORDINATES  (4) STRUCTURE (ORDINATE (4) BYTE) PUBLIC ;
123   1       DECLARE R$DIVISOR$SINGLE (2) BYTE PUBLIC ;
124   1       DECLARE (X$POS$KEY,      X$NEG$KEY,
                 Y$POS$KEY,      Y$NEG$KEY,
                 Z$POS$KEY,      Z$NEG$KEY,
                 R$0$POS$KEY,    R$0$NEG$KEY,
                 R$1$POS$KEY,    R$1$NEG$KEY,
                 R$2$POS$KEY,    R$2$NEG$KEY,
                 R$3$POS$KEY,    R$3$NEG$KEY,
                 FAST$KEY,       SLOW$KEY,
                 START$WELD$KEY, STOP$WELD$KEY,
                 ENTER$KEY                      ) BYTE PUBLIC ;
125   1       DECLARE (ROTATION$ENABLED,
                                LINEAR$ENABLED    ) BYTE PUBLIC ;
```

```
          $EJECT
            /* LOCAL DATA */
126  1        DECLARE (KEY$ROW$1,
                          KEY$ROW$2,
                          KEY$ROW$3,
                          KEY$ROW$4,
                          KEY$ROW$5  ) BYTE ;
127  1        DECLARE POSITION$DIVISOR (10) STRUCTURE ( X (2) BYTE,
                                                        Y (2) BYTE,
                                                        Z (2) BYTE,
                                                        R (2) BYTE )
                     /* X$DIVISOR / Y$DIVISOR / Z$DIVISOR / R$DIVISOR */
                  DATA (  000H, 064H, 000H, 085H, 000H, 064H, 006H, 040H,
                          000H, 082H, 000H, 0ADH, 000H, 0B2H, 008H, 020H,
                          000H, 0A9H, 000H, 0E1H, 000H, 0A9H, 00AH, 090H,
                          000H, 0DCH, 001H, 025H, 000H, 0DCH, 00DH, 0C0H,
                          001H, 01EH, 001H, 07DH, 001H, 01EH, 011H, 0E0H,
                          001H, 073H, 001H, 0EFH, 001H, 073H, 017H, 030H,
                          001H, 0E3H, 002H, 084H, 001H, 0E3H, 01EH, 030H,
                          002H, 073H, 003H, 045H, 002H, 073H, 027H, 030H,
                          003H, 030H, 004H, 040H, 003H, 030H, 033H, 000H,
                          004H, 024H, 005H, 086H, 004H, 024H, 042H, 040H  ) ;
128  1        DECLARE TEMP$REG      (4) BYTE ;
```

```
        $EJECT
129  1      RUN$MOTORS: PROCEDURE PUBLIC ;
130  2          MOTORS$RUNNING = TRUE ;
131  2          CALL SET$DIRECTION$CONTROL ;
132  2          RST$6$5$MASK = FALSE ;
133  2          CALL SET$INTERRUPT$MASK ;
134  2          ENABLE ;
135  2          DO WHILE MOTORS$RUNNING ;
136  3          END ;
137  2      END RUN$MOTORS ;
```

```
                $EJECT
138   1         SET$UP$X: PROCEDURE (STEP$ADDRESS) PUBLIC ;
139   2         DECLARE STEP$ADDRESS ADDRESS ;
140   2         IF X$POSITIVE
                THEN
141   2             DO ;
142   3                 CALL ADD (.X$ORDINATE, STEP$ADDRESS) ;
143   3                 CALL STORE$RESULT (.TEMP$REG) ;
144   3                 IF X$UPPER$LIMIT$SET
                        THEN
145   3                     CALL SUBTRACT (.X$UPPER$LIMIT, .TEMP$REG) ;
                        ELSE
146   3                     RESULT(0) = FALSE ;
147   3             END ;
                ELSE
148   2             DO ;
149   3                 CALL SUBTRACT (.X$ORDINATE, STEP$ADDRESS) ;
150   3                 CALL STORE$RESULT (.TEMP$REG) ;
151   3                 IF X$LOWER$LIMIT$SET
                        THEN
152   3                     CALL SUBTRACT (.TEMP$REG, .X$LOWER$LIMIT) ;
                        ELSE
153   3                     RESULT(0) = FALSE ;
154   3             END ;
155   2         IF RESULT$NEGATIVE
                THEN
156   2             DO ;
157   3                 X$ENABLED = FALSE ;
158   3                 CALL CONSOLE$OUTPUT (BELL) ;
159   3             END ;
                ELSE
160   2             DO ;
161   3                 CALL SUBTRACT (STEP$ADDRESS, .ONE) ;
162   3                 OUTPUT (COUNTER$MODE) = SETTING (SELECT$0
                                                       AND MODE$0
                                                       AND READ$LOAD$LSB$MSB
                                                       AND BINARY) ;
163   3                 OUTPUT (X$COUNTER) = RESULT(3) ;
164   3                 OUTPUT (X$COUNTER) = RESULT(2) ;
165   3                 CALL STORE$NUMBER (.X$ORDINATE, .TEMP$REG) ;
166   3             END ;
167   2         END SET$UP$X ;
```

```
        $EJECT
168  1      SET$UP$Y: PROCEDURE (STEP$ADDRESS) PUBLIC ;
169  2      DECLARE STEP$ADDRESS ADDRESS ;
170  2      IF Y$POSITIVE
            THEN
171  2          DO ;
172  3              CALL ADD (.Y$ORDINATE, STEP$ADDRESS) ;
173  3              CALL STORE$RESULT (.TEMP$REG) ;
174  3              IF Y$UPPER$LIMIT$SET
                    THEN
175  3                  CALL SUBTRACT (.Y$UPPER$LIMIT, .TEMP$REG) ;
                    ELSE
176  3                  RESULT(0) = FALSE ;
177  3          END ;
            ELSE
178  2          DO ;
179  3              CALL SUBTRACT (.Y$ORDINATE, STEP$ADDRESS) ;
180  3              CALL STORE$RESULT (.TEMP$REG) ;
181  3              IF Y$LOWER$LIMIT$SET
                    THEN
182  3                  CALL SUBTRACT (.TEMP$REG, .Y$LOWER$LIMIT) ;
                    ELSE
183  3                  RESULT(0) = FALSE ;
184  3          END ;
185  2      IF RESULT$NEGATIVE
            THEN
186  2          DO ;
187  3              Y$ENABLED = FALSE ;
188  3              CALL CONSOLE$OUTPUT (BELL) ;
189  3          END ;
            ELSE
190  2          DO ;
191  3              CALL SUBTRACT (STEP$ADDRESS, .ONE) ;
192  3              OUTPUT (COUNTER$MODE) = SETTING (SELECT$1
                                                     AND MODE$0
                                                     AND READ$LOAD$LSB$MSB
                                                     AND BINARY ) ;
193  3              OUTPUT (Y$COUNTER) = RESULT(3) ;
194  3              OUTPUT (Y$COUNTER) = RESULT(2) ;
195  3              CALL STORE$NUMBER (.Y$ORDINATE, .TEMP$REG) ;
196  3          END ;
197  2      END SET$UP$Y ;
```

```
                $EJECT
198   1         SET$UP$Z: PROCEDURE (STEP$ADDRESS) PUBLIC ;
199   2             DECLARE STEP$ADDRESS ADDRESS ;
200   2             IF Z$POSITIVE
                    THEN
201   2                 DO ;
202   3                     CALL ADD (.Z$ORDINATE, STEP$ADDRESS) ;
203   3                     CALL STORE$RESULT (.TEMP$REG) ;
204   3                     IF Z$UPPER$LIMIT$SET
                            THEN
205   3                         CALL SUBTRACT (.Z$UPPER$LIMIT, .TEMP$REG) ;
                            ELSE
206   3                         RESULT(0) = FALSE ;
207   3                 END ;
                    ELSE
208   2                 DO ;
209   3                     CALL SUBTRACT (.Z$ORDINATE, STEP$ADDRESS) ;
210   3                     CALL STORE$RESULT (.TEMP$REG) ;
211   3                     IF Z$LOWER$LIMIT$SET
                            THEN
212   3                         CALL SUBTRACT (.TEMP$REG, .Z$LOWER$LIMIT) ;
                            ELSE
213   3                         RESULT(0) = FALSE ;
214   3                 END ;
215   2             IF RESULT$NEGATIVE
                    THEN
216   2                 DO ;
217   3                     Z$ENABLED = FALSE ;
218   3                     CALL CONSOLE$OUTPUT (BELL) ;
219   3                 END ;
                    ELSE
220   2                 DO ;
221   3                     CALL SUBTRACT (STEP$ADDRESS, .ONE) ;
222   3                     OUTPUT (COUNTER$MODE) = SETTING (SELECT$2
                                                           AND MODE$0
                                                           AND READ$LOAD$LSB$MSB
                                                           AND BINARY ) ;
223   3                     OUTPUT (Z$COUNTER) = RESULT(3) ;
224   3                     OUTPUT (Z$COUNTER) = RESULT(2) ;
225   3                     CALL STORE$NUMBER (.Z$ORDINATE, .TEMP$REG) ;
226   3                 END ;
227   2         END SET$UP$Z ;
```

```
                $EJECT
228   1         STROBE$CONTROL$KEYS: PROCEDURE PUBLIC ;
229   2             OUTPUT (CONTROLLER$STROBE) = BIT$0 ;
230   2             KEY$ROW$1 = INPUT (CONTROLLER$KEYS) ;
231   2             OUTPUT (CONTROLLER$STROBE) = BIT$1 ;
232   2             KEY$ROW$2 = INPUT (CONTROLLER$KEYS) ;
233   2             OUTPUT (CONTROLLER$STROBE) = BIT$2 ;
234   2             KEY$ROW$3 = INPUT (CONTROLLER$KEYS) ;
235   2             OUTPUT (CONTROLLER$STROBE) = BIT$3 ;
236   2             KEY$ROW$4 = INPUT (CONTROLLER$KEYS) ;
237   2             OUTPUT (CONTROLLER$STROBE) = BIT$4 ;
238   2             KEY$ROW$5 = INPUT (CONTROLLER$KEYS) ;
239   2             IF KEY$ROW$1
                    THEN    FAST$KEY = FALSE ;
241   2                ELSE    FAST$KEY = TRUE ;
242   2             IF ROR (KEY$ROW$1, 1)
                    THEN    Z$POS$KEY = FALSE ;
244   2                ELSE    Z$POS$KEY = TRUE ;
245   2             IF ROR (KEY$ROW$1, 2)
                    THEN    Y$POS$KEY = FALSE ;
247   2                ELSE    Y$POS$KEY = TRUE ;
248   2             IF ROR (KEY$ROW$1, 3)
                    THEN    X$POS$KEY = FALSE ;
250   2                ELSE    X$POS$KEY = TRUE ;
251   2             IF KEY$ROW$2
                    THEN    SLOW$KEY = FALSE ;
253   2                ELSE    SLOW$KEY = TRUE ;
254   2             IF ROR (KEY$ROW$2, 1)
                    THEN    Z$NEG$KEY = FALSE ;
256   2                ELSE    Z$NEG$KEY = TRUE ;
257   2             IF ROR (KEY$ROW$2, 2)
                    THEN    Y$NEG$KEY = FALSE ;
259   2                ELSE    Y$NEG$KEY = TRUE ;
260   2             IF ROR (KEY$ROW2, 3)
                    THEN    X$NEG$KEY = FALSE ;
262   2                ELSE    X$NEG$KEY = TRUE ;
263   2             IF KEY$ROW$3
                    THEN    R$0$POS$KEY = FALSE ;
265   2                ELSE    R$0$POS$KEY = TRUE ;
266   2             IF ROR (KEY$ROW$3, 1)
                    THEN    R$1$POS$KEY = FALSE ;
268   2                ELSE    R$1$POS$KEY = TRUE ;
269   2             IF ROR (KEY$ROW$3, 2)
                    THEN    R$2$POS$KEY = FALSE ;
271   2                ELSE    R$2$POS$KEY = TRUE ;
272   2             IF ROR (KEY$ROW$3, 3)
                    THEN    R$3$POS$KEY = FALSE ;
274   2                ELSE    R$3$POS$KEY = TRUE ;
275   2             IF KEY$ROW$4
                    THEN    R$0$NEG$KEY = FALSE ;
277   2                ELSE    R$0$NEG$KEY = TRUE ;
278   2             IF ROR (KEY$ROW$4, 1)
                    THEN    R$1$NEG$KEY = FALSE ;
280   2                ELSE    R$1$NEG$KEY = TRUE ;
281   2             IF ROR (KEY$ROW$4, 2)
                    THEN    R$2$NEG$KEY = FALSE ;
283   2                ELSE    R$2$NEG$KEY = TRUE ;
```

```
284  2            IF ROR (KEY$ROW$4, 3)
                  THEN    R$3$NEG$KEY = FALSE ;
286  2            ELSE    R$3$NEG$KEY = TRUE ;
287  2            IF KEY$ROW$5
                  THEN    ENTER$KEY = FALSE ;
289  2            ELSE    ENTER$KEY = TRUE ;
290  2            IF ROR (KEY$ROW$5, 1)
                  THEN    START$WELD$KEY = FALSE ;
292  2            ELSE    START$WELD$KEY = TRUE ;
293  2            IF ROR (KEY$ROW$5, 2)
                  THEN    STOP$WELD$KEY = FALSE ;
295  2            ELSE    STOP$WELD$KEY = TRUE ;
296  2        END STROBE$CONTROL$KEYS ;
```

```
                $EJECT
297   1         VERIFY$CONTROL$KEYS: PROCEDURE BYTE PUBLIC ;
298   2             DECLARE STATUS BYTE ;
299   2             STATUS = TRUE ;
300   2             OUTPUT (CONTROLLER$STROBE) = BIT$0 ;
301   2             IF (INPUT(CONTROLLER$KEYS) <> KEY$ROW$1)
                    THEN
302   2                 STATUS = FALSE ;
303   2             OUTPUT (CONTROLLER$STROBE) = BIT$1 ;
304   2             IF (INPUT(CONTROLLER$KEYS) <> KEY$ROW$2)
                    THEN
305   2                 STATUS = FALSE ;
306   2             OUTPUT (CONTROLLER$STROBE) = BIT$2 ;
307   2             IF (INPUT(CONTROLLER$KEYS) <> KEY$ROW$3)
                    THEN
308   2                 STATUS = FALSE ;
309   2             OUTPUT (CONTROLLER$STROBE) = BIT$3 ;
310   2             IF (INPUT(CONTROLLER$KEYS) <> KEY$ROW$4)
                    THEN
311   2                 STATUS = FALSE ;
312   2             OUTPUT (CONTROLLER$STROBE) = BIT$4 ;
313   2             IF (INPUT(CONTROLLER$KEYS) <> KEY$ROW$5)
                    THEN
314   2                 STATUS = FALSE ;
315   2             RETURN STATUS ;
316   2         END VERIFY$CONTROL$KEYS ;

317   1         RELEASE$KEYS: PROCEDURE PUBLIC ;
318   2             KEY$ROW$1,
                              KEY$ROW$2,
                              KEY$ROW$3,
                              KEY$ROW$4,
                              KEY$ROW$5   = 0FFH ;
319   2             DO WHILE NOT (VERIFY$CONTROL$KEYS) ;
320   3             END ;
321   2         END RELEASE$KEYS ;
```

```
        $EJECT

322  1      SET$CONTROLLER$INDICATORS: PROCEDURE PUBLIC ;
323  2          OUTPUT (CONTROLLER$INDICATORS) = ( (R$0$ENABLED      OR BIT$0) AND
                                                  (R$1$ENABLED      OR BIT$1) AND
                                                  (R$2$ENABLED      OR BIT$2) AND
                                                  (R$3$ENABLED      OR BIT$3) AND
                                                  (CONTROLLER$ACTIVE OR BIT$4) AND
                                                  (ON$SEAM          OR BIT$5)    ) ;
324  2      END SET$CONTROLLER$INDICATORS ;
```

```
                $EJECT
325   1           STEP$LINEAR: PROCEDURE PUBLIC ;
326   2               IF X$POS$KEY XOR X$NEG$KEY
                      THEN
327   2                   X$ENABLED = TRUE ;
                      ELSE
328   2                   X$ENABLED = FALSE ;
329   2               IF Y$POS$KEY XOR Y$NEG$KEY
                      THEN
330   2                   Y$ENABLED = TRUE ;
                      ELSE
331   2                   Y$ENABLED = FALSE ;
332   2               IF Z$POS$KEY XOR Z$NEG$KEY
                      THEN
333   2                   Z$ENABLED = TRUE ;
                      ELSE
334   2                   Z$ENABLED = FALSE ;
335   2               IF X$NEG$KEY
                      THEN
336   2                   X$POSITIVE = FALSE ;
                      ELSE
337   2                   X$POSITIVE = TRUE ;
338   2               IF Y$NEG$KEY
                      THEN
339   2                   Y$POSITIVE = FALSE ;
                      ELSE
340   2                   Y$POSITIVE = TRUE ;
341   2               IF Z$NEG$KEY
                      THEN
342   2                   Z$POSITIVE = FALSE ;
                      ELSE
343   2                   Z$POSITIVE = TRUE ;
344   2               IF X$ENABLED
                      THEN
345   2                   CALL SET$UP$X (.TEN) ;
346   2               IF Y$ENABLED
                      THEN
347   2                   CALL SET$UP$Y (.TEN) ;
348   2               IF Z$ENABLED
                      THEN
349   2                   CALL SET$UP$Z (.TEN) ;
350   2               CALL RUN$MOTORS ;
351   2           END STEP$LINEAR ;
```

```
        $EJECT
352  1      ROTATE: PROCEDURE (R$NUMBER, DISPLACEMENT$ADDRESS) PUBLIC ;
353  2          DECLARE (R$NUMBER,
                        DIRECTION ) BYTE ;
354  2          DECLARE DISPLACEMENT$ADDRESS ADDRESS ;
355  2          DECLARE (I, J)               ADDRESS ;
356  2          DECLARE I$BYTE  (2) BYTE  AT (.I)    ;
357  2          DECLARE DISPLACEMENT$SINGLE BASED DISPLACEMENT$ADDRESS (2) BYTE ;
358  2          IF ROL (DISPLACEMENT$SINGLE(0), 1)
                THEN
359  2              DO ;
360  3                  DIRECTION = FALSE ;
361  3                  CALL NEGATE$SINGLE (.DISPLACEMENT$SINGLE) ;
362  3                  CALL STORE$RESULT$SINGLE$ADDRESS (.I) ;
363  3              END ;
                ELSE
364  2              DO ;
365  3                  DIRECTION = TRUE ;
366  3                  I$BYTE(0) = DISPLACEMENT$SINGLE(1) ;
367  3                  I$BYTE(1) = DISPLACEMENT$SINGLE(0) ;
368  3              END ;
369  2          CALL ADD$SINGLE$TO$DOUBLE (.R$ORDINATES(R$NUMBER).ORDINATE, DISPLACEMENT$ADDRESS) ;
370  2          CALL STORE$RESULT (.R$ORDINATES(R$NUMBER).ORDINATE) ;
371  2          DO J = 1 TO I ;
372  3              OUTPUT (R$DIVIDER) = R$DIVISOR$SINGLE(1) ;
373  3              OUTPUT (R$DIVIDER) = R$DIVISOR$SINGLE(0) ;
374  3              R$ENABLED = TRUE ;
375  3              R$POSITIVE = DIRECTION ;
376  3              MOTORS$RUNNING = TRUE ;
377  3              CALL SET$DIRECTION$CONTROL ;
378  3              RST$6$5$MASK = FALSE ;
379  3              CALL SET$INTERRUPT$MASK ;
380  3              ENABLE ;
381  3              CALL TIME (1) ;
382  3              DO CASE R$NUMBER ;
383  4                  OUTPUT (ROTATION) = 01H ;
384  4                  OUTPUT (ROTATION) = 02H ;
385  4                  OUTPUT (ROTATION) = 04H ;
386  4                  OUTPUT (ROTATION) = 08H ;
387  4              END ;
388  3              CALL TIME (1) ;
389  3              OUTPUT (ROTATION) = 00H ;
390  3              DO WHILE MOTORS$RUNNING ;
391  4              END ;
392  3          END ;
393  2      END ROTATE ;
```

```
                $EJECT
394   1         STEP$ROTATION: PROCEDURE (AUTO$DISABLE) PUBLIC ;
395   2             DECLARE AUTO$DISABLE BYTE ;
396   2             IF ((R$0$POS$KEY XOR R$0$NEG$KEY) AND R$0$ENABLED)
                    THEN
397   2                 DO ;
398   3                     IF AUTO$DISABLE
                            THEN
399   3                         DO ;
400   4                             R$1$ENABLED, R$2$ENABLED, R$3$ENABLED = FALSE ;
401   4                             CALL SET$CONTROLLER$INDICATORS ;
402   4                         END ;
403   3                     IF R$0$POS$KEY
                            THEN
404   3                         CALL ROTATE (0, .ONE$SINGLE) ;
                            ELSE
405   3                         CALL ROTATE (0, .MINUS$ONE$SINGLE) ;
406   3                 END ;
407   2             IF ((R$1$POS$KEY XOR R$1$NEG$KEY) AND R$1$ENABLED)
                    THEN
408   2                 DO ;
409   3                     IF AUTO$DISABLE
                            THEN
410   3                        DO ;
411   4                            R$0$ENABLED, R$2$ENABLED, R$3$ENABLED = FALSE ;
412   4                            CALL SET$CONTROLLER$INDICATORS ;
413   4                        END ;
414   3                    IF R$1$POS$KEY
                           THEN
415   3                         CALL ROTATE (1, .ONE$SINGLE) ;
                           ELSE
416   3                         CALL ROTATE (1, .MINUS$ONE$SINGLE) ;
417   3                 END ;
418   2             IF ((R$2$POS$KEY XOR R$2$NEG$KEY) AND R$2$ENABLED)
                    THEN
419   2                 DO ;
420   3                     IF AUTO$DISABLE
                            THEN
421   3                         DO ;
422   4                             R$0$ENABLED, R$1$ENABLED, R$3$ENABLED = FALSE ;
423   4                             CALL SET$CONTROLLER$INDICATORS ;
424   4                         END ;
425   3                     IF R$2$POS$KEY
                            THEN
426   3                         CALL ROTATE (2, .ONE$SINGLE) ;
                            ELSE
427   3                         CALL ROTATE (2, .MINUS$ONE$SINGLE) ;
428   3                 END ;
429   2             IF ((R$3$POS$KEY XOR R$3$NEG$KEY) AND R$3$ENABLED)
                    THEN
430   2                 DO ;
431   3                     IF AUTO$DISABLE
                            THEN
432   3                         DO ;
433   4                             R$0$ENABLED, R$1$ENABLED, R$2$ENABLED = FALSE ;
434   4                             CALL SET$CONTROLLER$INDICATORS ;
435   4                         END ;
```

```
436  3                  IF R$3$POS$KEY
                        THEN
437  3                      CALL ROTATE (3, .ONE$SINGLE) ;
                        ELSE
438  3                      CALL ROTATE (3, .MINUS$ONE$SINGLE) ;
439  3              END ;
440  2      END STEP$ROTATION ;
```

```
                $EJECT
441   1         ACCELERATE: PROCEDURE PUBLIC ;
442   2             IF (SPEED$STATUS > 1)
                    THEN
443   2                 DO ;
444   3                     SPEED$STATUS = SPEED$STATUS - 1 ;
445   3                     OUTPUT (X$DIVIDER) = POSITION$DIVISOR(SPEED$STATUS).X(1) ;
446   3                     OUTPUT (X$DIVIDER) = POSITION$DIVISOR(SPEED$STATUS).X(0) ;
447   3                     OUTPUT (Y$DIVIDER) = POSITION$DIVISOR(SPEED$STATUS).Y(1) ;
448   3                     OUTPUT (Y$DIVIDER) = POSITION$DIVISOR(SPEED$STATUS).Y(0) ;
449   3                     OUTPUT (Z$DIVIDER) = POSITION$DIVISOR(SPEED$STATUS).Z(1) ;
450   3                     OUTPUT (Z$DIVIDER) = POSITION$DIVISOR(SPEED$STATUS).Z(0) ;
451   3                     R$DIVISOR$SINGLE(0) = POSITION$DIVISOR(SPEED$STATUS).R(0) ;
452   3                     R$DIVISOR$SINGLE(1) = POSITION$DIVISOR(SPEED$STATUS).R(1) ;
453   3                     DELAY$COUNT = 50 ;
454   3                     DELAY$FLAG = TRUE ;
455   3                     RST$5$5$MASK = FALSE ;
456   3                     CALL SET$INTERRUPT$MASK ;
457   3                 END ;
458   2         END ACCELERATE ;

459   1         DECELERATE: PROCEDURE PUBLIC ;
460   2             IF (SPEED$STATUS < 10)
                    THEN
461   2                 DO ;
462   3                     SPEED$STATUS = SPEED$STATUS + 1 ;
463   3                     OUTPUT (X$DIVIDER) = POSITION$DIVISOR(SPEED$STATUS).X(1) ;
464   3                     OUTPUT (X$DIVIDER) = POSITION$DIVISOR(SPEED$STATUS).X(0) ;
465   3                     OUTPUT (Y$DIVIDER) = POSITION$DIVISOR(SPEED$STATUS).Y(1) ;
466   3                     OUTPUT (Y$DIVIDER) = POSITION$DIVISOR(SPEED$STATUS).Y(0) ;
467   3                     OUTPUT (Z$DIVIDER) = POSITION$DIVISOR(SPEED$STATUS).Z(1) ;
468   3                     OUTPUT (Z$DIVIDER) = POSITION$DIVISOR(SPEED$STATUS).Z(0) ;
469   3                     R$DIVISOR$SINGLE(0) = POSITION$DIVISOR(SPEED$STATUS).R(0) ;
470   3                     R$DIVISOR$SINGLE(1) = POSITION$DIVISOR(SPEED$STATUS).R(1) ;
471   3                     DELAY$COUNT = 50 ;
472   3                     DELAY$FLAG = TRUE ;
473   3                     RST$5$5$MASK = FALSE ;
474   3                     CALL SET$INTERRUPT$MASK ;
475   3                 END ;
476   2         END DECELERATE ;
```

```
                $EJECT
477   1         MOVE: PROCEDURE (AUTO$DISABLE) PUBLIC ;
478   2             DECLARE AUTO$DISABLE BYTE ;
479   2             DO FOREVER ;
480   3                 CALL STROBE$CONTROL$KEYS ;
481   3                 DO WHILE NOT (X$POS$KEY      OR X$NEG$KEY
                                OR Y$POS$KEY      OR Y$NEG$KEY
                                OR Z$POS$KEY      OR Z$NEG$KEY
                                OR R$0$POS$KEY    OR R$0$NEG$KEY
                                OR R$1$POS$KEY    OR R$1$NEG$KEY
                                OR R$2$POS$KEY    OR R$2$NEG$KEY
                                OR R$3$POS$KEY    OR R$3$NEG$KEY
                                OR FAST$KEY       OR SLOW$KEY
                                OR START$WELD$KEY OR STOP$WELD$KEY
                                OR ENTER$KEY                      ) ;
482   4                     CALL STROBE$CONTROL$KEYS ;
483   4                 END ;
484   3                 IF ( ( START$WELD$KEY AND ( NOT(STOP$WELD$KEY OR ENTER$KEY)) )
                                OR
                                  ( STOP$WELD$KEY AND ( NOT(START$WELD$KEY OR ENTER$KEY)) )
                                OR
                           ( ENTER$KEY AND ( NOT(START$WELD$KEY OR STOP$WELD$KEY)) ) )
                        THEN
485   3                     DO ;
486   4                         CALL TIME (200) ;
487   4                         IF VERIFY$CONTROL$KEYS
                                THEN
488   4                             DO ;
489   5                                 CALL DISPLAY$X ;
490   5                                 CALL DISPLAY$Y ;
491   5                                 CALL DISPLAY$Z ;
492   5                                 CALL DISPLAY$R(0) ;
493   5                                 CALL DISPLAY$R(1) ;
494   5                                 CALL DISPLAY$R(2) ;
495   5                                 CALL DISPLAY$R(3) ;
496   5                                 RETURN ;
497   5                             END ;
498   4                     END ;
                        ELSE
499   3                     DO ;
500   4                         IF (NOT DELAY$FLAG)
                                THEN
501   4                             DO ;
502   5                                 IF FAST$KEY
                                        THEN
503   5                                     CALL ACCELERATE ;
504   5                                 IF SLOW$KEY
                                        THEN
505   5                                     CALL DECELERATE ;
506   5                             END ;
507   4                         IF (R$0$POS$KEY OR R$0$NEG$KEY OR
                                    R$1$POS$KEY OR R$1$NEG$KEY OR
                                    R$2$POS$KEY OR R$2$NEG$KEY OR
                                    R$3$POS$KEY OR R$3$NEG$KEY    )
                                THEN
508   4                             DO ;
509   5                                 IF ROTATION$ENABLED
```

```
                                THEN
510  5                              DO ;
511  6                                  IF AUTO$DISABLE
                                        THEN
512  6                                      DO ;
513  7                                          LINEAR$ENABLED = FALSE ;
514  7                                          CALL STEP$ROTATION (TRUE) ;
515  7                                      END ;
                                        ELSE
516  6                                      CALL STEP$ROTATION (FALSE) ;
517  6                                  END ;
518  5                          END ;
                           ELSE
519  4                            DO ;
520  5                            IF (   X$POS$KEY OR X$NEG$KEY
                                    OR Y$POS$KEY OR Y$NEG$KEY
                                    OR Z$POS$KEY OR Z$NEG$KEY)
                                THEN
521  5                              DO ;
522  6                                  IF LINEAR$ENABLED
                                        THEN
523  6                                      DO ;
524  7                                          IF AUTO$DISABLE
                                                THEN
525  7                                              DO ;
526  8                                                  ROTATION$ENABLED,
                                                        R$0$ENABLED,
                                                        R$1$ENABLED,
                                                        R$2$ENABLED,
                                                        R$3$ENABLED      = FALSE ;
527  8                                                  CALL SET$CONTROLLER$INDICATORS ;
528  8                                              END ;
529  7                                          CALL STEP$LINEAR ;
530  7                                      END ;
531  6                                  END ;
532  5                          END ;
533  4                    END ;
534  3                END ;
535  2        END MOVE ;

536  1        POSITION: PROCEDURE PUBLIC ;
537  2            SPEED$STATUS = 3 ;
538  2            OUTPUT (X$DIVIDER) = POSITION$DIVISOR(SPEED$STATUS).X(1) ;
539  2            OUTPUT (X$DIVIDER) = POSITION$DIVISOR(SPEED$STATUS).X(0) ;
540  2            OUTPUT (Y$DIVIDER) = POSITION$DIVISOR(SPEED$STATUS).Y(1) ;
541  2            OUTPUT (Y$DIVIDER) = POSITION$DIVISOR(SPEED$STATUS).Y(0) ;
542  2            OUTPUT (Z$DIVIDER) = POSITION$DIVISOR(SPEED$STATUS).Z(1) ;
543  2            OUTPUT (Z$DIVIDER) = POSITION$DIVISOR(SPEED$STATUS).Z(0) ;
544  2            R$DIVISOR$SINGLE(0) = POSITION$DIVISOR(SPEED$STATUS).R(0) ;
545  2            R$DIVISOR$SINGLE(1) = POSITION$DIVISOR(SPEED$STATUS).R(1) ;
546  2            ROTATION$ENABLED,
                          R$0$ENABLED,
                          R$1$ENABLED,
                          R$2$ENABLED,
                          R$3$ENABLED,
                          CONTROLLER$ACTIVE,
                          LINEAR$ENABLED      = TRUE ;
```

```
547   2             ON$SEAM             = FALSE ;
548   2             CALL SET$CONTROLLER$INDICATORS ;
549   2             DO FOREVER ;
550   3                 CALL MOVE (FALSE) ;
551   3                 IF ENTER$KEY
                        THEN
552   3                     DO ;
553   4                         CALL DISPLAY$X ;
554   4                         CALL DISPLAY$Y ;
555   4                         CALL DISPLAY$Z ;
556   4                         CALL DISPLAY$R(0) ;
557   4                         CALL DISPLAY$R(1) ;
558   4                         CALL DISPLAY$R(2) ;
559   4                         CALL DISPLAY$R(3) ;
560   4                         R$0$ENABLED,
                                R$1$ENABLED,
                                R$2$ENABLED,
                                R$3$ENABLED,
                                CONTROLLER$ACTIVE,
                                ON$SEAM             = FALSE ;
561   4                         CALL SET$CONTROLLER$INDICATORS ;
562   4                         RETURN ;
563   4                     END ;
564   3             END ;
565   2         END POSITION ;
566   1     END POSITION$PROCEDURE ;
```

MODULE INFORMATION:

```
    CODE AREA SIZE     = 0A0AH   2570D
    VARIABLE AREA SIZE = 0055H     85D
    MAXIMUM STACK SIZE = 0008H      8D
    1069 LINES READ
    0 PROGRAM ERROR(S)
```

END OF PL/M-80 COMPILATION

```
ISIS-II PL/M-80 V3.1 COMPILATION OF MODULE WELDPROCEDURE
OBJECT MODULE PLACED IN :F1:WELD11.OBJ
COMPILER INVOKED BY:  PLM80 :F1:WELD11.PLM DEBUG WORKFILES(:F1:,:F1:) PRINT(:F1:WELD11.LST)


            $TITLE('WELD$PROCEDURE    VER 3.00    DATE 08/07/81')
  1         WELD$PROCEDURE: DO ;

            /* A PLM80 PROCEDURE FOR THE CONTROL OF THREE DIMENSIONAL TIG WELDS.
               THE PATH IS DEFINED BY THE TORCH DISPLACEMENTS IN DISPLACEMENT$TABLE.      */

            /* EXTERNALS */

            $INCLUDE (:F1:WELD.EQU)
        =   $NOLIST
        =
  3   1         DECLARE RESET$ARC$FLIP$FLOP LITERALLY 'RSTFF' ;
            $INCLUDE (:F1:MATH.EQU)
        =   $NOLIST

  5   1         MOTOR$CALCULATIONS: PROCEDURE (TABLE$ADDRESS, DIVISOR$FLOAT$ADDRESS, UPDATE) EXTERNAL ;
  6   2             DECLARE (TABLE$ADDRESS, DIVISOR$FLOAT$ADDRESS) ADDRESS ;
  7   2             DECLARE UPDATE BYTE ;
  8   2         END MOTOR$CALCULATIONS ;

  9   1         REAL$SUBTRACT: PROCEDURE (B$ADDRESS, A$ADDRESS) EXTERNAL ;
 10   2             DECLARE (A$ADDRESS, B$ADDRESS) ADDRESS ;
 11   2         END REAL$SUBTRACT ;

 12   1         NEGATE$SINGLE: PROCEDURE (A$ADDRESS) EXTERNAL ;
 13   2             DECLARE A$ADDRESS ADDRESS ;
 14   2         END NEGATE$SINGLE ;

 15   1         ADD$SINGLE: PROCEDURE (B$ADDRESS, A$ADDRESS) EXTERNAL ;
 16   2             DECLARE (A$ADDRESS, B$ADDRESS) ADDRESS ;
 17   2         END ADD$SINGLE ;

 18   1         STORE$NUMBER$SINGLE: PROCEDURE (STORE$ADDRESS, SOURCE$ADDRESS) EXTERNAL ;
 19   2             DECLARE (STORE$ADDRESS, SOURCE$ADDRESS) ADDRESS ;
 20   2         END STORE$NUMBER$SINGLE ;

 21   1         FIX$8$BIT: PROCEDURE (DATA$ADDRESS) EXTERNAL ;
 22   2             DECLARE DATA$ADDRESS ADDRESS ;
 23   2         END FIX$8$BIT ;

 24   1         FIX$SINGLE$ROUND: PROCEDURE (DATA$ADDRESS) EXTERNAL ;
 25   2             DECLARE DATA$ADDRESS ADDRESS ;
 26   2         END FIX$SINGLE$ROUND ;

 27   1         STORE$RESULT$SINGLE$ADDRESS: PROCEDURE (STORE$ADDRESS) EXTERNAL ;
 28   2             DECLARE STORE$ADDRESS ADDRESS ;
 29   2         END STORE$RESULT$SINGLE$ADDRESS ;

 30   1         STORE$NUMBER: PROCEDURE (STORE$ADDRESS, SOURCE$ADDRESS) EXTERNAL ;
 31   2             DECLARE (STORE$ADDRESS, SOURCE$ADDRESS) ADDRESS ;
 32   2         END STORE$NUMBER ;
```

```
33  1   REAL$ADD: PROCEDURE (B$ADDRESS, A$ADDRESS) EXTERNAL ;
34  2       DECLARE (A$ADDRESS, B$ADDRESS) ADDRESS ;
35  2   END REAL$ADD ;

36  1   REAL$MULTIPLY: PROCEDURE (B$ADDRESS, A$ADDRESS) EXTERNAL ;
37  2       DECLARE (A$ADDRESS, B$ADDRESS) ADDRESS ;
38  2   END REAL$MULTIPLY ;

39  1   REAL$DIVIDE: PROCEDURE (B$ADDRESS, A$ADDRESS) EXTERNAL ;
40  2       DECLARE (A$ADDRESS, B$ADDRESS) ADDRESS ;
41  2   END REAL$DIVIDE ;

42  1   REAL$NEGATE: PROCEDURE (A$ADDRESS) EXTERNAL ;
43  2       DECLARE A$ADDRESS ADDRESS ;
44  2   END REAL$NEGATE ;

45  1   SUBTRACT$SINGLE: PROCEDURE (B$ADDRESS, A$ADDRESS) EXTERNAL ;
46  2       DECLARE (A$ADDRESS, B$ADDRESS) ADDRESS ;
47  2   END SUBTRACT$SINGLE ;

48  1   SUBTRACT: PROCEDURE (B$ADDRESS, A$ADDRESS) EXTERNAL ;
49  2       DECLARE (A$ADDRESS, B$ADDRESS) ADDRESS ;
50  2   END SUBTRACT ;

51  1   FLOAT: PROCEDURE (A$ADDRESS) EXTERNAL ;
52  2       DECLARE A$ADDRESS ADDRESS ;
53  2   END FLOAT ;

54  1   FLOAT$SINGLE: PROCEDURE (A$ADDRESS) EXTERNAL ;
55  2       DECLARE A$ADDRESS ADDRESS ;
56  2   END FLOAT$SINGLE ;

57  1   FIX$16$BIT: PROCEDURE (A$ADDRESS) EXTERNAL ;
58  2       DECLARE A$ADDRESS ADDRESS ;
59  2   END FIX$16$BIT ;

60  1   SQUARE$ROOT: PROCEDURE (A$ADDRESS) EXTERNAL ;
61  2       DECLARE A$ADDRESS ADDRESS ;
62  2   END SQUARE$ROOT ;

63  1   STORE$RESULT: PROCEDURE (STORE$ADDRESS) EXTERNAL ;
64  2       DECLARE STORE$ADDRESS ADDRESS ;
65  2   END STORE$RESULT ;

66  1   STORE$RESULT$SINGLE: PROCEDURE (STORE$ADDRESS) EXTERNAL ;
67  2       DECLARE (STORE$ADDRESS) ADDRESS ;
68  2   END STORE$RESULT$SINGLE ;

69  1   MULT$BY$2$PI: PROCEDURE (DATA$ADDRESS) EXTERNAL ;
70  2       DECLARE DATA$ADDRESS ADDRESS ;
71  2   END MULT$BY$2$PI ;

72  1   TANGENT: PROCEDURE (DATA$ADDRESS) EXTERNAL ;
73  2       DECLARE DATA$ADDRESS ADDRESS ;
74  2   END TANGENT ;
```

```
 75  1          COSINE: PROCEDURE (DATA$ADDRESS) EXTERNAL ;
 76  2              DECLARE DATA$ADDRESS ADDRESS ;
 77  2          END COSINE ;

 78  1          SINE: PROCEDURE (DATA$ADDRESS) EXTERNAL ;
 79  2              DECLARE DATA$ADDRESS ADDRESS ;
 80  2          END SINE ;

 81  1          DECLARE MATH$STATUS BYTE EXTERNAL ;
 82  1          DECLARE (FOUR,
                         FIVE,
                         FOUR$FLOAT,
                         FIVE$FLOAT,
                         ONE$FLOAT,
                         MINUS$ONE$FLOAT,
                         FIVE$HUNDRED$FLOAT,
                         FIFTY$FLOAT,
                         ONE$MILLION$FLOAT,
                         ZERO$FLOAT,
                         ONE$HUNDRED$FLOAT,
                         ONE$THOUSAND$FLOAT,
                         FOUR$HUNDRED$FLOAT,
                         RESULT          ) (4) BYTE EXTERNAL ;
 83  1          DECLARE (ONE$SINGLE,
                         MINUS$ONE$SINGLE,
                         ZERO$SINGLE,
                         RESULT$SINGLE  ) (2) BYTE EXTERNAL ;
 84  1          SET$INTERRUPT$MASK: PROCEDURE EXTERNAL ;
 85  2          END SET$INTERRUPT$MASK ;

 86  1          SET$POWER$SOURCE$CONTROL: PROCEDURE EXTERNAL ;
 87  2          END SET$POWER$SOURCE$CONTROL ;

 88  1          RESET$ARC$FLIP$FLOP: PROCEDURE EXTERNAL ;
 89  2          END RESET$ARC$FLIP$FLOP ;

 90  1          SET$DIRECTION$CONTROL: PROCEDURE EXTERNAL ;
 91  2          END SET$DIRECTION$CONTROL ;

 92  1          SET$CONTROLLER$INDICATORS: PROCEDURE EXTERNAL ;
 93  2          END SET$CONTROLLER$INDICATORS ;

 94  1          POSITION: PROCEDURE EXTERNAL ;
 95  2          END POSITION ;

 96  1          START$MOTORS: PROCEDURE EXTERNAL ;
 97  2          END START$MOTORS ;

 98  1          ROTATE: PROCEDURE (R$NUMBER, DISPLACEMENT$ADDRESS) EXTERNAL ;
 99  2              DECLARE R$NUMBER BYTE ;
100  2              DECLARE DISPLACEMENT$ADDRESS ADDRESS ;
101  2          END ROTATE ;

102  1          POSITION$CURSOR: PROCEDURE (COL$NUMBER, LINE$NUMBER) EXTERNAL ;
103  2              DECLARE (COL$NUMBER, LINE$NUMBER) BYTE ;
104  2          END POSITION$CURSOR ;
```

```
105  1          DISPLAY$BYTE: PROCEDURE (BYTE$VALUE) EXTERNAL ;
106  2              DECLARE BYTE$VALUE BYTE ;
107  2          END DISPLAY$BYTE ;

108  1          CONSOLE$INPUT: PROCEDURE BYTE EXTERNAL ;
109  2          END CONSOLE$INPUT ;

110  1          WRITE$MESSAGE: PROCEDURE (MESSAGE$ADDRESS) EXTERNAL ;
111  2              DECLARE MESSAGE$ADDRESS ADDRESS ;
112  2          END WRITE$MESSAGE ;

113  1          HOME$TORCH: PROCEDURE EXTERNAL ;
114  2          END HOME$TORCH ;

115  1          CLEAR$PROMPT$FIELD: PROCEDURE EXTERNAL ;
116  2          END CLEAR$PROMPT$FIELD ;

117  1          DECLARE DISPLACEMENT$TABLE (MAX$POINTS) STRUCTURE (IDENTIFIER BYTE,
                                                                  DISPLACEMENT$1$SINGLE (2) BYTE,
                                                                  DISPLACEMENT$2$SINGLE (2) BYTE,
                                                                  DISPLACEMENT$3$SINGLE (2) BYTE ) EXTERNAL ;
118  1          DECLARE PARAMETER$GROUP$TABLE (256) STRUCTURE (PULSE$CURRENT$FLOAT (4) BYTE,
                                                              BACKGROUND$CURRENT$FLOAT (4) BYTE,
                                                              PULSE$ON$TIME$FLOAT (4) BYTE ,
                                                              PULSE$OFF$TIME$FLOAT (4) BYTE,
                                                              WELDING$DIVISOR$FLOAT (4) BYTE,
                                                              PRE$TRAVERSE$TIME$FLOAT (4) BYTE,
                                                              FALL$TIME$FLOAT (4) BYTE,
                                                              R$0$DIVISOR$SINGLE (2) BYTE,
                                                              R$1$DIVISOR$SINGLE (2) BYTE,
                                                              R$2$DIVISOR$SINGLE (2) BYTE,
                                                              R$3$DIVISOR$SINGLE (2) BYTE,
                                                              TOUCH$START BYTE,
                                                              WIRE$FEED   BYTE          ) EXTERNAL ;
119  1          DECLARE MAIN$DIVISOR$FLOAT   (4) BYTE EXTERNAL ;
120  1          DECLARE SEQUENCE$TABLE (256) BYTE EXTERNAL ;
121  1          DECLARE R$ORDINATES (4) STRUCTURE (ORDINATE (4) BYTE) EXTERNAL ;
122  1          DECLARE (X$ENABLED,
                        Y$ENABLED,
                        Z$ENABLED ) BYTE EXTERNAL ;
123  1          DECLARE  ON$SEAM     BYTE EXTERNAL ;
124  1          DECLARE PARAMETERS$SET$FLAG (256) BYTE EXTERNAL ;
125  1          DECLARE RST$7$5$MASK              BYTE EXTERNAL ;
126  1          DECLARE RST$5$5$MASK              BYTE EXTERNAL ;
127  1          DECLARE (PULSE$ON$FLAG,
                        PULSE$OFF$FLAG,
                        DELAY$FLAG,
                        FALL$FLAG,
                        FALL$DELAY$FLAG,
                        PRE$TRAVERSE$FLAG ) BYTE EXTERNAL ;
128  1          DECLARE (START$ENABLE,
                        STOP$ENABLE,
                        GAS$ENABLE,
                        WIRE$ENABLE      ) BYTE EXTERNAL ;
129  1          DECLARE (RTC$DIVISOR$FLOAT,
                        PULSE$CURRENT$FLOAT,
                        BACKGROUND$CURRENT$FLOAT,
```

```
                        CURRENT$INCREMENT$FLOAT   ) (4) BYTE EXTERNAL ;
130   1         DECLARE DISPLACEMENT$ERROR              BYTE EXTERNAL ;
131   1         DECLARE SEAM$COUNT                      BYTE EXTERNAL ;
132   1         DECLARE POINT$COUNT                     BYTE EXTERNAL ;
133   1         DECLARE MOTORS$RUNNING                  BYTE EXTERNAL ;
134   1         DECLARE R$DIVISOR$SINGLE              (2) BYTE EXTERNAL ;
135   1         DECLARE PATH$SET                        BYTE EXTERNAL ;
```

```
        $EJECT
        /* PUBLIC DATA */
136  1      DECLARE RATIO$0    (4) BYTE PUBLIC
                   DATA (03H, 80H, 00H, 00H) ;
137  1      DECLARE RATIO$1    (4) BYTE PUBLIC
                   DATA (03H, 80H, 00H, 00H) ;
138  1      DECLARE  ARC$DIED                BYTE PUBLIC ;
139  1      DECLARE BACKGROUND$CURRENT$BYTE          BYTE PUBLIC ;
140  1      DECLARE PULSE$CURRENT$BYTE               BYTE PUBLIC ;
141  1      DECLARE (DELAY$COUNT,        DELAY$COUNTER,
                    FALL$COUNT,         FALL$COUNTER,
                    FALL$DELAY$COUNT,   FALL$DELAY$COUNTER,
                    PULSE$ON$COUNT,     PULSE$ON$COUNTER,
                    PULSE$OFF$COUNT,    PULSE$OFF$COUNTER,
                    PRE$TRAVERSE$COUNT, PRE$TRAVERSE$COUNTER ) ADDRESS PUBLIC ;
142  1      DECLARE REMAINDER$SINGLE         (2) BYTE PUBLIC ;
143  1      DECLARE POSITIONING$DIVISOR$FLOAT (4) BYTE PUBLIC
                   DATA (07H, 0A0H, 00H, 00H) ;
144  1      DECLARE POSITIONING$DIVISOR$SINGLE (2) BYTE PUBLIC
                   DATA (00H, 64H) ;
145  1      DECLARE FALL$COUNT$FLOAT         (4) BYTE PUBLIC ;
```

```
                $EJECT
                /* LOCAL DATA */
146   1           DECLARE UPDATE              LITERALLY 'TRUE',
                          NO$UPDATE           LITERALLY 'FALSE' ;

147   1           DECLARE (X$SQUARED$FLOAT,
                           Y$SQUARED$FLOAT,
                           DISPLACEMENT$FLOAT,
                           X$VECTOR$FLOAT,
                           Y$VECTOR$FLOAT,
                           Z$VECTOR$FLOAT      )   (4) BYTE ;

148   1           DECLARE VECTOR$SINGLE STRUCTURE (X$VECTOR$SINGLE (2) BYTE,
                                                   Y$VECTOR$SINGLE (2) BYTE,
                                                   Z$VECTOR$SINGLE (2) BYTE ) ;
149   1           DECLARE SEAM$TIME$FLOAT                          (4) BYTE ;
150   1           DECLARE (THETA$FLOAT,
                           PHI$FLOAT,
                           COS$THETA$FLOAT,
                           TAN$THETA$FLOAT,
                           TAN$SQUARED$THETA$FLOAT,
                           TAN$PHI$FLOAT             )          (4) BYTE ;
151   1           DECLARE RATIO$0$FLOAT                          (4) BYTE
                          DATA (03H, 80H, 00H, 00H) ;
152   1           DECLARE RATIO$1$FLOAT                          (4) BYTE
                          DATA (03H, 80H, 00H, 00H) ;
153   1           DECLARE DIVISOR$ADDRESS                       ADDRESS ;
154   1           DECLARE (R$COUNT,
                           M$COUNT,
                           CHARACTER,
                           R$ID,
                           IDENTIFIER) BYTE ;
155   1           DECLARE (P$COUNT,
                           I,
                           J       ) ADDRESS ;
156   1           DECLARE PARAMETER$ERROR BYTE ;
157   1           DECLARE (RATIO$FLOAT,
                           RATIO$DIVISOR$FLOAT) (4) BYTE ;
158   1           DECLARE (IGNITION$ERROR,
                           IGNITE,
                           ARC$FAIL,
                           FIRST$SEGMENT         )         BYTE ;
159   1           DECLARE IGNITION$CURRENT$FLOAT (7) STRUCTURE (I (4) BYTE)
                          DATA (002H, 0B0H, 000H, 000H,
                                006H, 0C8H, 000H, 000H,
                                004H, 0A0H, 000H, 000H,
                                005H, 0A0H, 000H, 000H,
                                005H, 0F0H, 000H, 000H,
                                006H, 0A0H, 000H, 000H,
                                006H, 0C8H, 000H, 000H  ) ;
160   1           DECLARE IGNITION$CURRENT$BYTE                  (7) BYTE ;
```

```
$EJECT

161  1      RE$POSITION: PROCEDURE ;
162  2          IF P$COUNT > 0
                THEN
163  2              DO ;
164  3                  P$COUNT = P$COUNT - 1 ;
165  3                  IF ROR((IDENTIFIER := DISPLACEMENT$TABLE(P$COUNT).IDENTIFIER), 2)
                        THEN
166  3                      DO ;
167  4                          CALL NEGATE$SINGLE (.DISPLACEMENT$TABLE(P$COUNT).DISPLACEMENT$1$SINGLE) ;
168  4                          CALL STORE$RESULT$SINGLE (.VECTOR$SINGLE.X$VECTOR$SINGLE) ;
169  4                          CALL NEGATE$SINGLE (.DISPLACEMENT$TABLE(P$COUNT).DISPLACEMENT$2$SINGLE) ;
170  4                          CALL STORE$RESULT$SINGLE (.VECTOR$SINGLE.Y$VECTOR$SINGLE) ;
171  4                          CALL NEGATE$SINGLE (.DISPLACEMENT$TABLE(P$COUNT).DISPLACEMENT$3$SINGLE) ;
172  4                          CALL STORE$RESULT$SINGLE (.VECTOR$SINGLE.Z$VECTOR$SINGLE) ;
173  4                          CALL MOTOR$CALCULATIONS (.VECTOR$SINGLE.X$VECTOR$SINGLE,
                                                         .POSITIONING$DIVISOR$FLOAT,
                                                          UPDATE                        ) ;
174  4                          CALL START$MOTORS ;
175  4                          DO WHILE MOTORS$RUNNING ;
176  5                          END ;
177  4                      END ;
                        ELSE
178  3                      DO ;
179  4                          CALL NEGATE$SINGLE (.DISPLACEMENT$TABLE(P$COUNT).DISPLACEMENT$1$SINGLE) ;
180  4                          IF ROR (IDENTIFIER, 3)
                                THEN
181  4                              R$ID = 0 ;
182  4                          IF ROR (IDENTIFIER, 4)
                                THEN
183  4                              R$ID = 1 ;
184  4                          IF ROR (IDENTIFIER, 5)
                                THEN
185  4                              R$ID = 2 ;
186  4                          IF ROR (IDENTIFIER, 6)
                                THEN
187  4                              R$ID = 3 ;
188  4                          CALL ROTATE (R$ID, .RESULT$SINGLE) ;
189  4                      END ;
190  3              END ;
191  2      END RE$POSITION ;

192  1      ARC$OFF: PROCEDURE BYTE PUBLIC ;
193  2          IF INPUT(0)
                THEN
194  2              RETURN TRUE ;
                ELSE
195  2              RETURN FALSE ;
196  2      END ARC$OFF ;
```

```
                $EJECT
197   1         DISPLACE$TORCH: PROCEDURE ;
198   2           DECLARE IDENTIFIER BYTE ;
199   2           IF ROR (IDENTIFIER := DISPLACEMENT$TABLE(P$COUNT).IDENTIFIER, 2)
                  THEN
200   2             DO ;
201   3               IF ON$SEAM
                      THEN
202   3                 CALL MOTOR$CALCULATIONS (.DISPLACEMENT$TABLE(P$COUNT).DISPLACEMENT$1$SINGLE,
                        .PARAMETER$GROUP$TABLE(SEQUENCE$TABLE(W$COUNT)).WELDING$DIVISOR$FLOAT,
                         UPDATE) ;
                      ELSE
203   3                 CALL MOTOR$CALCULATIONS (.DISPLACEMENT$TABLE(P$COUNT).DISPLACEMENT$1$SINGLE,
                        .POSITIONING$DIVISOR$FLOAT,
                         UPDATE                    ) ;
204   3               DO WHILE MOTORS$RUNNING ;
205   4               END ;
206   3               IF ARC$DIED
                      THEN
207   3                 DO ;
208   4                   CALL NEGATE$SINGLE (.DISPLACEMENT$TABLE(P$COUNT).DISPLACEMENT$1$SINGLE) ;
209   4                   CALL STORE$RESULT$SINGLE (.VECTOR$SINGLE.X$VECTOR$SINGLE) ;
210   4                   CALL NEGATE$SINGLE (.DISPLACEMENT$TABLE(P$COUNT).DISPLACEMENT$2$SINGLE) ;
211   4                   CALL STORE$RESULT$SINGLE (.VECTOR$SINGLE.Y$VECTOR$SINGLE) ;
212   4                   CALL NEGATE$SINGLE (.DISPLACEMENT$TABLE(P$COUNT).DISPLACEMENT$3$SINGLE) ;
213   4                   CALL STORE$RESULT$SINGLE (.VECTOR$SINGLE.Z$VECTOR$SINGLE) ;
214   4                   CALL MOTOR$CALCULATIONS (.VECTOR$SINGLE.X$VECTOR$SINGLE,
                                                   .POSITIONING$DIVISOR$FLOAT,
                                                    UPDATE                    ) ;
215   4                   RETURN ;
216   4                 END ;
217   3               CALL START$MOTORS ;
218   3             END ;
                    ELSE
219   2               DO ;
220   3                 DO WHILE MOTORS$RUNNING ;
221   4                 END ;
222   3                 IF ARC$DIED
                        THEN
223   3                   RETURN ;
224   3                 R$DIVISOR$SINGLE(0) = DISPLACEMENT$TABLE(P$COUNT).DISPLACEMENT$2$SINGLE(0) ;
225   3                 R$DIVISOR$SINGLE(1) = DISPLACEMENT$TABLE(P$COUNT).DISPLACEMENT$2$SINGLE(1) ;
226   3                 IF ROR (IDENTIFIER, 3)
                        THEN
227   3                   R$ID = 0 ;
228   3                 IF ROR (IDENTIFIER, 4)
                        THEN
229   3                   R$ID = 1 ;
230   3                 IF ROR (IDENTIFIER, 5)
                        THEN
231   3                   R$ID = 2 ;
232   3                 IF ROR (IDENTIFIER, 6)
                        THEN
233   3                   R$ID = 3 ;
234   3                 CALL ROTATE (R$ID, .DISPLACEMENT$TABLE(P$COUNT).DISPLACEMENT$1$SINGLE) ;
235   3               END ;
236   2               P$COUNT = P$COUNT + 1 ;
```

```
237  2              FIRST$SEGMENT = FALSE ;
238  2        END DISPLACE$TORCH ;
```

```
        $EJECT
239  1      CALCULATE$SEAM$TIME: PROCEDURE ;
240  2          DECLARE I ADDRESS ;
241  2          DECLARE FLAG BYTE ;
242  2          CALL STORE$NUMBER (.SEAM$TIME$FLOAT, .ZERO$FLOAT) ;
243  2          FLAG = TRUE ;
244  2          I = P$COUNT ;
245  2          DO WHILE FLAG ;
246  3              IF ROR (IDENTIFIER := DISPLACEMENT$TABLE(I).IDENTIFIER, 2)
                    THEN
247  3                  DO ;
248  4                      CALL FLOAT$SINGLE (.DISPLACEMENT$TABLE(I).DISPLACEMENT$1$SINGLE) ;
249  4                      CALL REAL$MULTIPLY (.RESULT, .RESULT) ;
250  4                      CALL STORE$RESULT (.X$SQUARED$FLOAT) ;
251  4                      CALL FLOAT$SINGLE (.DISPLACEMENT$TABLE(I).DISPLACEMENT$2$SINGLE) ;
252  4                      CALL REAL$MULTIPLY (.RESULT, .FIVE$FLOAT) ;
253  4                      CALL REAL$DIVIDE   (.RESULT, .FOUR$FLOAT) ;
254  4                      CALL REAL$MULTIPLY (.RESULT, .RESULT) ;
255  4                      CALL STORE$RESULT  (.Y$SQUARED$FLOAT) ;
256  4                      CALL FLOAT$SINGLE (.DISPLACEMENT$TABLE(I).DISPLACEMENT$3$SINGLE) ;
257  4                      CALL REAL$MULTIPLY (.RESULT, .RESULT) ;
258  4                      CALL REAL$ADD (.RESULT, .Y$SQUARED$FLOAT) ;
259  4                      CALL REAL$ADD (.RESULT, .X$SQUARED$FLOAT) ;
260  4                      CALL SQUARE$ROOT (.RESULT) ;
261  4                      CALL REAL$MULTIPLY (.RESULT,
                                .PARAMETER$GROUP$TABLE(SEQUENCE$TABLE(W$COUNT)).WELDING$DIVISOR$FLOAT) ;
262  4                      CALL REAL$MULTIPLY (.RESULT, .MAIN$DIVISOR$FLOAT) ;
263  4                      CALL REAL$DIVIDE (.RESULT, .ONE$MILLION$FLOAT) ;
264  4                      CALL REAL$ADD (.RESULT, .SEAM$TIME$FLOAT) ;
265  4                      CALL STORE$RESULT (.SEAM$TIME$FLOAT) ;
266  4                  END ;
                    ELSE
267  3                  DO ;
268  4                      CALL FLOAT$SINGLE (.DISPLACEMENT$TABLE(I).DISPLACEMENT$1$SINGLE) ;
269  4                      CALL STORE$RESULT (.DISPLACEMENT$FLOAT) ;
270  4                      IF ROR (IDENTIFIER, 3)
                            THEN
271  4                          DIVISOR$ADDRESS =
                                .PARAMETER$GROUP$TABLE(SEQUENCE$TABLE(W$COUNT)).R$0$DIVISOR$SINGLE ;
272  4                      IF ROR (IDENTIFIER, 4)
                            THEN
273  4                          DIVISOR$ADDRESS =
                                .PARAMETER$GROUP$TABLE(SEQUENCE$TABLE(W$COUNT)).R$1$DIVISOR$SINGLE ;
274  4                      IF ROR (IDENTIFIER, 5)
                            THEN
275  4                          DIVISOR$ADDRESS =
                                .PARAMETER$GROUP$TABLE(SEQUENCE$TABLE(W$COUNT)).R$2$DIVISOR$SINGLE ;
276  4                      IF ROR (IDENTIFIER, 6)
                            THEN
277  4                          DIVISOR$ADDRESS =
                                .PARAMETER$GROUP$TABLE(SEQUENCE$TABLE(W$COUNT)).R$3$DIVISOR$SINGLE ;
278  4                      CALL FLOAT$SINGLE (DIVISOR$ADDRESS) ;
279  4                      CALL REAL$DIVIDE (.DISPLACEMENT$FLOAT, .RESULT) ;
280  4                      CALL REAL$DIVIDE (.RESULT, .ONE$MILLION$FLOAT) ;
281  4                      CALL REAL$DIVIDE (.RESULT, .MAIN$DIVISOR$FLOAT) ;
282  4                      CALL REAL$ADD (.RESULT, .SEAM$TIME$FLOAT) ;
283  4                      CALL STORE$RESULT (.SEAM$TIME$FLOAT) ;
```

```
284  4                  END ;
285  3              I = I + 1 ;
286  3              IF ROR (DISPLACEMENT$TABLE(I).IDENTIFIER, 1)
                    THEN
287  3                  FLAG = FALSE ;
288  3              END ;
289  2          END CALCULATE$SEAM$TIME ;
```

```
                $EJECT
290   1         TRACE$PATH: PROCEDURE ;
291   2           ARC$DIED = FALSE ;
292   2           DO WHILE P$COUNT < POINT$COUNT ;
293   3             IF (IDENTIFIER := DISPLACEMENT$TABLE(P$COUNT).IDENTIFIER)
                    THEN
294   3               ON$SEAM = TRUE ;
295   3             IF ROR (IDENTIFIER, 1)
                    THEN
296   3               ON$SEAM = FALSE ;
297   3             CALL DISPLACE$TORCH ;
298   3             CALL SET$CONTROLLER$INDICATORS ;
299   3           END ;
300   2           CALL HOME$TORCH ;
301   2           IF DISPLACEMENT$ERROR
                  THEN
302   2             DO ;
303   3               CALL POSITION$CURSOR (ERROR$FIELD) ;
304   3               CALL WRITE$MESSAGE   (.(DELETE$LINE,
                                              'PATH CLOSURE ERROR', BELL, EOT)) ;
305   3             END ;
306   2         END TRACE$PATH ;

307   1         GAS$PRE$PURGE: PROCEDURE ;
308   2           RETURN ;
309   2           GAS$ENABLE = TRUE ;
310   2           START$ENABLE,
                  STOP$ENABLE = FALSE ;
311   2           CALL SET$POWER$SOURCE$CONTROL ;
312   2           DELAY$COUNT = GAS$PRE$PURGE$COUNT ;
313   2           DELAY$FLAG = TRUE ;
314   2           DO WHILE DELAY$FLAG ;
315   3           END ;
316   2         END GAS$PRE$PURGE ;
```

```
        $EJECT
317  1      IGNITE$H$F: PROCEDURE ;
318  2          IGNITION$ERROR = FALSE ;
319  2          CALL GAS$PRE$PURGE ;
320  2          START$ENABLE = TRUE ;
321  2          STOP$ENABLE  = FALSE ;
322  2          CALL SET$POWER$SOURCE$CONTROL ;
323  2          OUTPUT (CURRENT$CONTROL) = PULSE$CURRENT$BYTE ;
324  2          DELAY$FLAG = FALSE ;
325  2          DELAY$COUNT = 500 ;
326  2          DELAY$FLAG = TRUE ;
327  2          DO WHILE DELAY$FLAG ;                /*DELAY FOR 5 Secs*/
328  3          END ;
329  2          DELAY$COUNT = IGNITION$TIME$OUT$COUNT ;
330  2          DELAY$FLAG = TRUE ;
331  2          DO WHILE (ARC$OFF XOR (NOT DELAY$FLAG)) ;
332  3          END ;
333  2          IF NOT (DELAY$FLAG)
                THEN
334  2              DO ;
335  3                  IGNITION$ERROR = TRUE ;
336  3                  START$ENABLE,
                        GAS$ENABLE     = FALSE ;
337  3                  CALL SET$POWER$SOURCE$CONTROL ;
338  3              END ;
                ELSE
339  2              DO ;
340  3                  ARC$DIED = FALSE ;
341  3              END ;
342  2      END IGNITE$H$F ;
```

```
                $EJECT
343   1         IGNITE$TOUCH: PROCEDURE ;
344   2             DECLARE IGNITION$VECTOR STRUCTURE (X$SINGLE (2) BYTE,
                                                     Y$SINGLE (2) BYTE,
                                                     Z$SINGLE (2) BYTE ) ;
345   2             IGNITION$ERROR = FALSE ;
346   2             CALL FLOAT (.R$ORDINATES(1)) ;
347   2             CALL REAL$DIVIDE (.RESULT, .FOUR$HUNDRED$FLOAT) ;
348   2             CALL REAL$DIVIDE (.RESULT, .RATIO$1$FLOAT) ;
349   2             CALL MULT$BY$2$PI (.RESULT) ;
350   2             CALL STORE$RESULT (.THETA$FLOAT) ;
351   2             CALL TANGENT(.RESULT) ;
352   2             CALL STORE$RESULT (.TAN$THETA$FLOAT) ;
353   2             CALL REAL$MULTIPLY (.RESULT, .RESULT) ;
354   2             CALL STORE$RESULT (.TAN$SQUARED$THETA$FLOAT) ;
355   2             CALL FLOAT (.R$ORDINATES(0)) ;
356   2             CALL REAL$DIVIDE (.RESULT, .FOUR$HUNDRED$FLOAT) ;
357   2             CALL REAL$DIVIDE (.RESULT, .RATIO$0$FLOAT) ;
358   2             CALL MULT$BY$2$PI (.RESULT) ;
359   2             CALL STORE$RESULT (.PHI$FLOAT) ;
360   2             CALL TANGENT (.RESULT) ;
361   2             CALL STORE$RESULT (.TAN$PHI$FLOAT) ;
362   2             CALL REAL$MULTIPLY (.RESULT, .RESULT) ;
363   2             CALL REAL$ADD (.RESULT, .TAN$SQUARED$THETA$FLOAT) ;
364   2             CALL REAL$ADD (.RESULT, .ONE$FLOAT) ;
365   2             CALL SQUARE$ROOT (.RESULT) ;
366   2             CALL REAL$DIVIDE (.ONE$THOUSAND$FLOAT, .RESULT) ;
367   2             CALL STORE$RESULT (.Z$VECTOR$FLOAT) ;
368   2             CALL REAL$MULTIPLY (.Z$VECTOR$FLOAT, .TAN$THETA$FLOAT) ;
369   2             IF MATH$NEGATIVE
                      THEN
370   2                 CALL REAL$NEGATE (.RESULT) ;
371   2             CALL STORE$RESULT (.X$VECTOR$FLOAT) ;
372   2             CALL REAL$MULTIPLY (.Z$VECTOR$FLOAT, .TAN$PHI$FLOAT) ;
373   2             IF MATH$NEGATIVE
                      THEN
374   2                 CALL REAL$NEGATE (.RESULT) ;
375   2             CALL STORE$RESULT (.Y$VECTOR$FLOAT) ;
376   2             CALL COSINE (.THETA$FLOAT) ;
377   2             CALL STORE$RESULT (.COS$THETA$FLOAT) ;
378   2             CALL COSINE (.PHI$FLOAT) ;
379   2             CALL REAL$MULTIPLY (.RESULT, .COS$THETA$FLOAT) ;
380   2             IF MATH$NEGATIVE
                      THEN
381   2                 CALL REAL$MULTIPLY (.Z$VECTOR$FLOAT, .ONE$FLOAT) ;
                      ELSE
382   2                 CALL REAL$MULTIPLY (.Z$VECTOR$FLOAT, .MINUS$ONE$FLOAT) ;
383   2             CALL FIX$16$BIT (.RESULT) ;
384   2             CALL STORE$RESULT$SINGLE (.IGNITION$VECTOR.Z$SINGLE) ;
385   2             CALL SINE (.THETA$FLOAT) ;
386   2             IF MATH$NEGATIVE
                      THEN
387   2                 CALL REAL$MULTIPLY (.X$VECTOR$FLOAT, .ONE$FLOAT) ;
                      ELSE
388   2                 CALL REAL$MULTIPLY (.X$VECTOR$FLOAT, .MINUS$ONE$FLOAT) ;
389   2             CALL FIX$16$BIT (.RESULT) ;
390   2             CALL STORE$RESULT$SINGLE (.IGNITION$VECTOR.X$SINGLE) ;
```

```
391  2            CALL SINE (.PHI$FLOAT) ;
392  2            IF MATH$NEGATIVE
                  THEN
393  2              CALL REAL$MULTIPLY (.Y$VECTOR$FLOAT, .ONE$FLOAT) ;
                  ELSE
394  2              CALL REAL$MULTIPLY (.Y$VECTOR$FLOAT, .MINUS$ONE$FLOAT) ;
395  2            CALL FIX$16$BIT (.RESULT) ;
396  2            CALL STORE$RESULT$SINGLE (.IGNITION$VECTOR.Y$SINGLE) ;
397  2            CALL MOTOR$CALCULATIONS (.IGNITION$VECTOR.X$SINGLE,
                                           .FIVE$HUNDRED$FLOAT,
                                            NO$UPDATE              ) ;
398  2            CALL GAS$PRE$PURGE ;
399  2            START$ENABLE = TRUE ;
400  2            STOP$ENABLE = FALSE ;
401  2            CALL SET$POWER$SOURCE$CONTROL ;
402  2            DELAY$FLAG = FALSE ;
403  2            DELAY$COUNT = 500 ;
404  2            DELAY$FLAG = TRUE ;
405  2            DO WHILE DELAY$FLAG ;          /*DELAY FOR 5 Secs*/
406  3            END ;
407  2            OUTPUT (CURRENT$CONTROL) = IGNITION$CURRENT$BYTE(0) ;
408  2            CALL START$MOTORS ;
409  2            DELAY$FLAG = FALSE ;
410  2            DELAY$COUNT = 5 ;
411  2            DELAY$FLAG = TRUE ;
412  2            DO WHILE DELAY$FLAG ;          /*DELAY FOR 50 mS*/
413  3            END ;
414  2            DO WHILE (MOTORS$RUNNING AND ARC$OFF) ;
415  3            END ;
416  2            X$ENABLED,
                  Y$ENABLED,
                  Z$ENABLED    = FALSE ;
417  2            CALL SET$DIRECTION$CONTROL ;
418  2            CALL TIME (100) ;         /* DELAY 10mS */
419  2            IF ARC$OFF
                  THEN
420  2              IGNITION$ERROR = TRUE ;
421  2            DELAY$FLAG = FALSE ;
422  2            DELAY$COUNT = 1000 ;      /*ELECTRODE WARM UP = 10 Sec*/
423  2            DELAY$FLAG = TRUE ;
424  2            OUTPUT (CURRENT$CONTROL) = IGNITION$CURRENT$BYTE(1) ;
425  2            IF (IGNITION$VECTOR.X$SINGLE(0) OR IGNITION$VECTOR.X$SINGLE(1)) <> 0
                  THEN
426  2              DO ;
427  3                REMAINDER$SINGLE(1) = INPUT (X$COUNTER) ;
428  3                REMAINDER$SINGLE(0) = INPUT (X$COUNTER) ;
429  3                IF ROL (IGNITION$VECTOR.X$SINGLE(0), 1)
                       THEN
430  3                  DO ;
431  4                    CALL ADD$SINGLE (.REMAINDER$SINGLE, .IGNITION$VECTOR.X$SINGLE) ;
432  4                    CALL NEGATE$SINGLE (.RESULT$SINGLE) ;
433  4                  END ;
                       ELSE
434  3                  CALL SUBTRACT$SINGLE (.REMAINDER$SINGLE, .IGNITION$VECTOR.X$SINGLE) ;
435  3                CALL STORE$RESULT$SINGLE (.IGNITION$VECTOR.X$SINGLE) ;
436  3              END ;
437  2            IF (IGNITION$VECTOR.Y$SINGLE(0) OR IGNITION$VECTOR.Y$SINGLE(1)) <> 0
```

```
                    THEN
438  2                  DO ;
439  3                      REMAINDER$SINGLE(1) = INPUT (Y$COUNTER) ;
440  3                      REMAINDER$SINGLE(0) = INPUT (Y$COUNTER) ;
441  3                      IF ROL (IGNITION$VECTOR.Y$SINGLE(0), 1)
                            THEN
442  3                          DO ;
443  4                              CALL ADD$SINGLE (.REMAINDER$SINGLE, .IGNITION$VECTOR.Y$SINGLE) ;
444  4                              CALL NEGATE$SINGLE (.RESULT$SINGLE) ;
445  4                          END ;
                            ELSE
446  3                          CALL SUBTRACT$SINGLE (.REMAINDER$SINGLE, .IGNITION$VECTOR.Y$SINGLE) ;
447  3                      CALL STORE$RESULT$SINGLE (.IGNITION$VECTOR.Y$SINGLE) ;
448  3                  END ;
449  2              IF (IGNITION$VECTOR.Z$SINGLE(0) OR IGNITION$VECTOR.Z$SINGLE(1)) <> 0
                    THEN
450  2                  DO ;
451  3                      REMAINDER$SINGLE(1) = INPUT (Z$COUNTER) ;
452  3                      REMAINDER$SINGLE(0) = INPUT (Z$COUNTER) ;
453  3                      IF ROL (IGNITION$VECTOR.Z$SINGLE(0), 1)
                            THEN
454  3                          DO ;
455  4                              CALL ADD$SINGLE (.REMAINDER$SINGLE, .IGNITION$VECTOR.Z$SINGLE) ;
456  4                              CALL NEGATE$SINGLE (.RESULT$SINGLE) ;
457  4                          END ;
                            ELSE
458  3                          CALL SUBTRACT$SINGLE (.REMAINDER$SINGLE, .IGNITION$VECTOR.Z$SINGLE) ;
459  3                      CALL STORE$RESULT$SINGLE (.IGNITION$VECTOR.Z$SINGLE) ;
460  3                  END ;
461  2              CALL MOTOR$CALCULATIONS (.IGNITION$VECTOR,
                                            .FIVE$HUNDRED$FLOAT,
                                             NO$UPDATE                  ) ;
462  2              DO WHILE DELAY$FLAG ;           /* ELECTRODE WARM UP TIME */
463  3              END ;
464  2              OUTPUT (CURRENT$CONTROL) = IGNITION$CURRENT$BYTE(2) ;
465  2              CALL START$MOTORS ;
466  2              DELAY$FLAG = FALSE ;
467  2              DELAY$COUNT = 25 ;              /* 250 mS DELAY */
468  2              DELAY$FLAG = TRUE ;
469  2              DO WHILE (DELAY$FLAG AND MOTORS$RUNNING) ;
470  3              END ;
471  2              OUTPUT (CURRENT$CONTROL) = IGNITION$CURRENT$BYTE(3) ;
472  2              DELAY$FLAG = FALSE ;
473  2              DELAY$COUNT = 25 ;              /* 250 mS DELAY */
474  2              DELAY$FLAG = TRUE ;
475  2              DO WHILE (DELAY$FLAG AND MOTORS$RUNNING) ;
476  3              END ;
477  2              OUTPUT (CURRENT$CONTROL) = IGNITION$CURRENT$BYTE(4) ;
478  2              DELAY$FLAG = FALSE ;
479  2              DELAY$COUNT = 25 ;              /* 250 mS DELAY */
480  2              DELAY$FLAG = TRUE ;
481  2              DO WHILE (DELAY$FLAG AND MOTORS$RUNNING) ;
482  3              END ;
483  2              OUTPUT (CURRENT$CONTROL) = IGNITION$CURRENT$BYTE(5) ;
484  2              DELAY$FLAG = FALSE ;
485  2              DELAY$COUNT = 25 ;              /* 250 mS DELAY */
486  2              DELAY$FLAG = TRUE ;
```

```
487  2            DO WHILE (DELAY$FLAG AND MOTORS$RUNNING) ;
488  3            END ;
489  2            OUTPUT (CURRENT$CONTROL) = IGNITION$CURRENT$BYTE(6) ;
490  2            DELAY$FLAG = FALSE ;
491  2            DELAY$COUNT = 25 ;            /* 250 mS DELAY */
492  2            DELAY$FLAG = TRUE ;
493  2            DO WHILE DELAY$FLAG ;
494  3            END ;
495  2            OUTPUT (CURRENT$CONTROL) = PULSE$CURRENT$BYTE ;
496  2            DO WHILE MOTORS$RUNNING ;
497  3            END ;
498  2            IF ARC$OFF
                  THEN
499  2               IGNITION$ERROR = TRUE ;
                  ELSE
500  2               DO ;
501  3                  ARC$DIED = FALSE ;
502  3               END ;
503  2         END IGNITE$TOUCH ;
```

```
                $EJECT
504   1         WELD$SEAM: PROCEDURE ;
505   2             DECLARE I BYTE ;

506   2             CALL STORE$NUMBER (.PULSE$CURRENT$FLOAT,
                                  .PARAMETER$GROUP$TABLE(SEQUENCE$TABLE(W$COUNT)).PULSE$CURRENT$FLOAT) ;
507   2             CALL REAL$DIVIDE (.PULSE$CURRENT$FLOAT, .CURRENT$INCREMENT$FLOAT) ;
508   2             CALL FIX$8$BIT (.RESULT) ;
509   2             PULSE$CURRENT$BYTE = RESULT$SINGLE(1) ;
510   2             DO I = 0 TO 6 ;
511   3                 CALL REAL$DIVIDE (.IGNITION$CURRENT$FLOAT(I), .CURRENT$INCREMENT$FLOAT) ;
512   3                 CALL FIX$8$BIT (.RESULT) ;
513   3                 IGNITION$CURRENT$BYTE(I) = RESULT$SINGLE(1) ;
514   3             END ;
515   2             CALL STORE$NUMBER (.BACKGROUND$CURRENT$FLOAT,
                                  .PARAMETER$GROUP$TABLE(SEQUENCE$TABLE(W$COUNT)).BACKGROUND$CURRENT$FLOAT) ;
516   2             CALL REAL$DIVIDE (.BACKGROUND$CURRENT$FLOAT, .CURRENT$INCREMENT$FLOAT) ;
517   2             CALL FIX$8$BIT (.RESULT) ;
518   2             BACKGROUND$CURRENT$BYTE = RESULT$SINGLE(1) ;
519   2             CALL REAL$MULTIPLY (.PARAMETER$GROUP$TABLE(SEQUENCE$TABLE(W$COUNT)).PULSE$ON$TIME$FLOAT,
                                  .ONE$MILLION$FLOAT) ;
520   2             CALL REAL$DIVIDE (.RESULT, .RTC$DIVISOR$FLOAT) ;
521   2             CALL FIX$SINGLE$ROUND (.RESULT) ;
522   2             CALL STORE$RESULT$SINGLE$ADDRESS (.PULSE$ON$COUNT) ;
523   2             PULSE$ON$COUNTER = 0 ;
524   2             CALL REAL$MULTIPLY (.PARAMETER$GROUP$TABLE(SEQUENCE$TABLE(W$COUNT)).PULSE$OFF$TIME$FLOAT,
                                  .ONE$MILLION$FLOAT) ;
525   2             CALL REAL$DIVIDE (.RESULT, .RTC$DIVISOR$FLOAT) ;
526   2             CALL FIX$SINGLE$ROUND (.RESULT) ;
527   2             CALL STORE$RESULT$SINGLE$ADDRESS (.PULSE$OFF$COUNT) ;
528   2             PULSE$OFF$COUNTER = 0 ;
529   2             CALL REAL$MULTIPLY (.PARAMETER$GROUP$TABLE(SEQUENCE$TABLE(W$COUNT)).PRE$TRAVERSE$TIME$FLOAT,
                                  .ONE$MILLION$FLOAT) ;
530   2             CALL REAL$DIVIDE (.RESULT, .RTC$DIVISOR$FLOAT) ;
531   2             CALL FIX$SINGLE$ROUND (.RESULT) ;
532   2             CALL STORE$RESULT$SINGLE$ADDRESS (.PRE$TRAVERSE$COUNT) ;
533   2             PRE$TRAVERSE$COUNTER = 0 ;
534   2             CALL REAL$MULTIPLY (.PARAMETER$GROUP$TABLE(SEQUENCE$TABLE(W$COUNT)).FALL$TIME$FLOAT,
                                  .ONE$MILLION$FLOAT) ;
535   2             CALL REAL$DIVIDE (.RESULT, .RTC$DIVISOR$FLOAT) ;
536   2             CALL STORE$RESULT (.FALL$COUNT$FLOAT) ;
537   2             CALL FIX$SINGLE$ROUND (.RESULT) ;
538   2             CALL STORE$RESULT$SINGLE$ADDRESS (.FALL$COUNT) ;
539   2             FALL$COUNTER = 0 ;
540   2             IGNITE = TRUE ;
541   2             DO WHILE IGNITE ;
542   3                 CALL CALCULATE$SEAM$TIME ;
543   3                 CALL REAL$SUBTRACT (.SEAM$TIME$FLOAT,
                                      .PARAMETER$GROUP$TABLE(SEQUENCE$TABLE(W$COUNT)).FALLTIME$FLOAT) ;
544   3                 IF RESULT$NEGATIVE
                        THEN
545   3                     DO ;
546   4                         CALL POSITION$CURSOR (PROMPT$FIELD) ;
547   4                         CALL WRITE$MESSAGE (.(DELETE$LINE,
                                                    'FALL TIME TOO LONG ON WELD SEAM ',
                                                     BELL, EOT)) ;
548   4                         CALL DISPLAY$BYTE (W$COUNT) ;
```

```
549  4                      CALL STORE$NUMBER (.RESULT, .ZERO$FLOAT) ;
550  4                  END ;
551  3              CALL REAL$MULTIPLY (.RESULT, .ONE$MILLION$FLOAT) ;
552  3              CALL REAL$DIVIDE   (.RESULT, .RTC$DIVISOR$FLOAT) ;
553  3              CALL FIX$SINGLE$ROUND (.RESULT) ;
554  3              CALL STORE$RESULT$SINGLE$ADDRESS (.FALL$DELAY$COUNT) ;
555  3              FALL$DELAY$COUNTER = 0 ;
556  3              DELAY$COUNTER = 0 ;
557  3              DO WHILE MOTORS$RUNNING ;
558  4              END ;
559  3              RST$5$5$MASK = FALSE ;
560  3              CALL SET$INTERRUPT$MASK ;
561  3              ON$SEAM = TRUE ;
562  3              CALL SET$CONTROLLER$INDICATORS ;
563  3              IGNITION$ERROR = TRUE ;
564  3              DO WHILE IGNITION$ERROR ;
565  4                  IF PARAMETER$GROUP$TABLE(SEQUENCE$TABLE(W$COUNT)).TOUCH$START
                        THEN
566  4                      CALL IGNITE$TOUCH ;
                        ELSE
567  4                      CALL IGNITE$H$F ;
568  4                  IF IGNITION$ERROR
                        THEN
569  4                      DO ;
570  5                      GAS$ENABLE,
                            START$ENABLE = FALSE ;
571  5                      STOP$ENABLE = TRUE ;
572  5                      CALL SET$POWER$SOURCE$CONTROL ;
573  5                      CALL POSITION$CURSOR (PROMPT$FIELD) ;
574  5                      CALL WRITE$MESSAGE (.(DELETE$LINE,
                                                  'ARC FAILED TO LIGHT, TRY AGAIN ?',
                                                   BELL, EOT )) ;
575  5                      IF CONSOLE$INPUT = 'N'
                            THEN
576  5                          DO ;
577  6                          ARC$FAIL = TRUE ;
578  6                          CALL CLEAR$PROMPT$FIELD ;
579  6                          RETURN ;
580  6                      END ;
581  5                  CALL CLEAR$PROMPT$FIELD ;
582  5              END ;
583  4          END ;
584  3          PULSE$ON$FLAG = TRUE ;
585  3          PRE$TRAVERSE$FLAG = TRUE ;
586  3          IF PARAMETER$GROUP$TABLE(SEQUENCE$TABLE(W$COUNT)).WIRE$FEED
               THEN
587  3              DO ;
588  4                  WIRE$ENABLE = TRUE ;
589  4                  CALL SET$POWER$SOURCE$CONTROL ;
590  4              END ;
591  3          DO WHILE PRE$TRAVERSE$FLAG ;
592  4          END ;
593  3          FALL$DELAY$FLAG = TRUE ;
594  3          FIRST$SEGMENT = TRUE ;
595  3          DO WHILE ( (P$COUNT < POINT$COUNT) AND (NOT ARC$DIED) AND
                          (NOT ROR(DISPLACEMENT$TABLE(P$COUNT).IDENTIFIER, 1)) ) ;
596  4              CALL DISPLACE$TORCH ;
```

```
597  4                END ;
598  3                DO WHILE MOTORS$RUNNING ;
599  4                END ;
600  3                IF ARC$DIED
                      THEN
601  3                    DO ;
602  4                        DISABLE ;
603  4                        OUTPUT (CURRENT$CONTROL) = 0 ;
604  4                        FALL$DELAY$FLAG,
                              PULSE$ON$FLAG,
                              PULSE$OFF$FLAG,
                              PRE$TRAVERSE$FLAG,
                              FALL$FLAG,
                              DELAY$FLAG       = FALSE ;
605  4                        FALL$DELAY$COUNTER,
                              PULSE$ON$COUNTER,
                              PULSE$OFF$COUNTER,
                              PRE$TRAVERSE$COUNTER,
                              FALL$COUNTER,
                              DELAY$COUNTER    = 0 ;
606  4                        RST$7$5$MASK,
                              RST$5$5$MASK = TRUE ;
607  4                        CALL SET$INTERRUPT$MASK ;
608  4                        ENABLE ;
609  4                        STOP$ENABLE = TRUE ;
610  4                        START$ENABLE,
                              WIRE$ENABLE = FALSE ;
611  4                        CALL SET$POWER$SOURCE$CONTROL ;
612  4                        IF (NOT FIRST$SEGMENT)
                              THEN
613  4                            CALL RE$POSITION ;
614  4                        IGNITE = TRUE ;
615  4                    END ;
                         ELSE
616  3                    DO ;
617  4                        RST$7$5$MASK = TRUE ;
618  4                        CALL SET$INTERRUPT$MASK ;
619  4                        START$ENABLE = FALSE ;
620  4                        STOP$ENABLE  = TRUE ;
621  4                        CALL SET$POWER$SOURCE$CONTROL ;
622  4                        IGNITE = FALSE ;
623  4                        DELAY$COUNT = POST$PURGE$COUNT ;
624  4                        DELAY$COUNTER = 0 ;
625  4                        DELAY$FLAG = TRUE ;
626  4                        DO WHILE DELAY$FLAG ;
627  5                        END ;
628  4                        GAS$ENABLE = FALSE ;
629  4                        STOP$ENABLE = FALSE ;
630  4                        CALL SET$POWER$SOURCE$CONTROL ;
631  4                        ON$SEAM = FALSE ;
632  4                        CALL SET$CONTROLLER$INDICATORS ;
633  4                    END ;
634  3            END ;
635  2        END WELD$SEAM ;
```

```
        $EJECT
636  1      WELD: PROCEDURE ;
637  2          ARC$DIED = FALSE ;
638  2          DO WHILE P$COUNT < POINT$COUNT ;
639  3              IF (IDENTIFIER := DISPLACEMENT$TABLE(P$COUNT).IDENTIFIER)
                    THEN
640  3                  CALL WELD$SEAM ;
                    ELSE
641  3                  CALL DISPLACE$TORCH ;
642  3              IF ARC$FAIL
                    THEN
643  3                  CALL TRACE$PATH ;
644  3          END ;
645  2          CALL HOME$TORCH ;
646  2          IF DISPLACEMENT$ERROR
                THEN
647  2              DO ;
648  3                  CALL POSITION$CURSOR (ERROR$FIELD) ;
649  3                  CALL WRITE$MESSAGE (.(DELETE$LINE,
                                              'WELD PROGRAM PATH CLOSURE ERROR',
                                                BELL, EOT)) ;
650  3              END ;
651  2      END WELD ;
```

```
                $EJECT
652   1         WELD$SEQUENCE: PROCEDURE PUBLIC ;
653   2             CALL POSITION$CURSOR (PROG$FIELD) ;
654   2             CALL WRITE$MESSAGE (.(DELETE$LINE, 'WELD$SEQUENCE', EOT)) ;
655   2             IF NOT (PATH$SET)
                    THEN
656   2                 DO ;
657   3                     CALL POSITION$CURSOR (ERROR$FIELD) ;
658   3                     CALL WRITE$MESSAGE (.(DELETE$LINE,
                                                'NO PATH IN MEMORY', BELL, EOT)) ;
659   3                     RETURN ;
660   3                 END ;
661   2             PARAMETER$ERROR,
                    ARC$FAIL        = FALSE ;
662   2             IF SEAM$COUNT <> 0
                    THEN
663   2                 DO J = 0 TO (SEAM$COUNT - 1) ;
664   3                     IF NOT (PARAMETERS$SET$FLAG(SEQUENCE$TABLE(J)))
                            THEN
665   3                         PARAMETER$ERROR = TRUE ;
666   3                 END ;
667   2             IF PARAMETER$ERROR
                    THEN
668   2                 DO ;
669   3                     CALL POSITION$CURSOR (ERROR$FIELD) ;
670   3                     CALL WRITE$MESSAGE (.(DELETE$LINE,
                                                'PARAMETER SEQUENCE ERROR', EOT)) ;
671   3                     RETURN ;
672   3                 END ;
673   2                 ON$SEAM = FALSE ;
674   2                 P$COUNT, W$COUNT = 0 ;
675   2                 DO WHILE P$COUNT < POINT$COUNT ;
676   3                     IF ROR((IDENTIFIER := DISPLACEMENT$TABLE(P$COUNT).IDENTIFIER), 3)
                            THEN
677   3                         DO ;
678   4                             IF ON$SEAM
                                    THEN
679   4                                 DIVISOR$ADDRESS =
                                          .PARAMETER$GROUP$TABLE(SEQUENCE$TABLE(W$COUNT)).R$0$DIVISOR$SINGLE ;
                                    ELSE
680   4                                 DIVISOR$ADDRESS = .POSITIONING$DIVISOR$SINGLE ;
681   4                             END ;
682   3                       IF ROR (IDENTIFIER, 4)
                              THEN
683   3                           DO ;
684   4                               IF ON$SEAM
                                      THEN
685   4                                   DIVISOR$ADDRESS =
                                            .PARAMETER$GROUP$TABLE(SEQUENCE$TABLE(W$COUNT)).R$1$DIVISOR$SINGLE ;
                                      ELSE
686   4                                   DIVISOR$ADDRESS = .POSITIONING$DIVISOR$SINGLE ;
687   4                               END ;
688   3                       IF ROR (IDENTIFIER, 5)
                              THEN
689   3                           DO ;
690   4                               IF ON$SEAM
                                      THEN
```

```
691  4                          DIVISOR$ADDRESS =
                                .PARAMETER$GROUP$TABLE(SEQUENCE$TABLE(W$COUNT)).R$2$DIVISOR$SINGLE ;
                            ELSE
692  4                          DIVISOR$ADDRESS = .POSITIONING$DIVISOR$SINGLE ;
693  4                      END ;
694  3                  IF ROR (IDENTIFIER, 6)
                        THEN
695  3                      DO ;
696  4                          IF ON$SEAM
                                THEN
697  4                          DIVISOR$ADDRESS =
                                .PARAMETER$GROUP$TABLE(SEQUENCE$TABLE(W$COUNT)).R$3$DIVISOR$SINGLE ;
                            ELSE
698  4                          DIVISOR$ADDRESS = .POSITIONING$DIVISOR$SINGLE ;
699  4                      END ;
700  3                  IF NOT ( ROR(IDENTIFIER, 2))
                        THEN
701  3                      CALL STORE$NUMBER$SINGLE (.DISPLACEMENT$TABLE(P$COUNT).DISPLACEMENT$2$SINGLE,
                                                     DIVISOR$ADDRESS) ;
702  3                  IF IDENTIFIER
                        THEN
703  3                      ON$SEAM = TRUE ;
704  3                  IF ROR (IDENTIFIER, 1)
                        THEN
705  3                      DO ;
706  4                          ON$SEAM = FALSE ;
707  4                          W$COUNT = W$COUNT + 1 ;
708  4                      END ;
709  3                  P$COUNT = P$COUNT + 1 ;
710  3              END ;
711  2          CALL HOME$TORCH ;
712  2          DO WHILE DISPLACEMENT$ERROR ;
713  3              CALL POSITION$CURSOR (PROMPT$FIELD) ;
714  3              CALL WRITE$MESSAGE (.(DELETE$LINE,
                                         'MOVE TORCH TO REST POSITION', EOT)) ;
715  3              CALL POSITION ;
716  3              CALL HOME$TORCH ;
717  3          END ;
718  2          P$COUNT = 0 ;
719  2          W$COUNT, R$COUNT = 0 ;
720  2          CALL POSITION$CURSOR (PROMPT$FIELD) ;
721  2          CALL WRITE$MESSAGE (.(DELETE$LINE,
                                      'TRACE OR WELD PATH', EOT )) ;
722  2          CHARACTER = CONSOLE$INPUT ;
723  2          DO WHILE ((CHARACTER <> 'T') AND (CHARACTER <> 'W')) ;
724  3              CHARACTER = CONSOLE$INPUT ;
725  3          END ;
726  2          IF CHARACTER = 'T'
               THEN
727  2              CALL TRACE$PATH ;
               ELSE
728  2              CALL WELD ;
729  2          CALL CLEAR$PROMPT$FIELD ;
730  2          END WELD$SEQUENCE ;
731  1      END WELD$PROCEDURE ;
```

```
MODULE INFORMATION:

    CODE AREA SIZE     = 10FCH   4348D
    VARIABLE AREA SIZE = 0087H    135D
    MAXIMUM STACK SIZE = 000AH     10D
    1285 LINES READ
    0 PROGRAM ERROR(S)

END OF PL/M-80 COMPILATION
```

```
ISIS-II PL/M-80 V3.1 COMPILATION OF MODULE REALTIMEINTERRUPTPROCEDURE
OBJECT MODULE PLACED IN :F1:WELD12.OBJ
COMPILER INVOKED BY:  PLM80 :F1:WELD12.PLM DEBUG WORKFILES(:F1:,:F1:) PRINT(:F1:WELD12.LST)



           $TITLE('REAL$TIME$INTERRUPT$PROCEDURE  VER 2.00  DATE 07/04/81')
  1        REAL$TIME$INTERRUPT$PROCEDURE: DO ;

           /* A PLM80 PROCEDURE TO HANDLE TIMING OF REAL TIME EVENTS. */

           /* EXTERNALS */
           $INCLUDE (:F1:WELD.EQU)
       =   $NOLIST
       =
           $INCLUDE (:F1:MATH.EQU)
       =   $NOLIST
           $INCLUDE (:F1:WELD1.EDS)
       =   $NOLIST
           $INCLUDE (:F1:MATH.EDS)
       =   $NOLIST
183    1       SET$POWER$SOURCE$CONTROL: PROCEDURE EXTERNAL ;
184    2       END SET$POWER$SOURCE$CONTROL ;

185    1       DECLARE WIRE$ENABLE BYTE EXTERNAL ;
186    1       DECLARE (PULSE$CURRENT$FLOAT,
                        BACK$GROUND$CURRENT$FLOAT,
                        CURRENT$INCREMENT$FLOAT,
                        FALL$COUNT$FLOAT          )    (4) BYTE EXTERNAL ;
187    1       DECLARE (PULSE$ON$COUNT,
                        PULSE$OFF$COUNT,
                        FALL$COUNT,
                        FALL$DELAY$COUNT,
                        PRE$TRAVERSE$COUNT,
                        PULSE$ON$COUNTER,
                        PULSE$OFF$COUNTER,
                        FALL$COUNTER,
                        FALL$DELAY$COUNTER,
                        PRE$TRAVERSE$COUNTER )                  ADDRESS EXTERNAL ;
188    1       DECLARE (PULSE$CURRENT$BYTE,
                        BACKGROUND$CURRENT$BYTE )               BYTE EXTERNAL ;
189    1       DECLARE MATH$IN$USE                             BYTE EXTERNAL ;
190    1       DECLARE (DELAY$COUNTER,
                        DELAY$COUNT     ) ADDRESS EXTERNAL ;
```

```
                $EJECT
191   1         REAL$TIME$INTERRUPT: PROCEDURE PUBLIC ;
192   2             OUTPUT (REAL$TIME$CLOCK) = RTC$DIVISOR$SINGLE(1) ;
193   2             OUTPUT (REAL$TIME$CLOCK) = RTC$DIVISOR$SINGLE(0) ;
194   2             IF PRE$TRAVERSE$FLAG
                    THEN
195   2                 DO ;
196   3                     PRE$TRAVERSE$COUNTER = PRE$TRAVERSE$COUNTER + 1 ;
197   3                     IF PRE$TRAVERSE$COUNTER > PRE$TRAVERSE$COUNT
                            THEN
198   3                         DO ;
199   4                             PRE$TRAVERSE$COUNTER = 0 ;
200   4                             PRE$TRAVERSE$FLAG = FALSE ;
201   4                         END ;
202   3                 END ;
203   2             IF FALL$DELAY$FLAG
                    THEN
204   2                 DO ;
205   3                     FALL$DELAY$COUNTER = FALL$DELAY$COUNTER + 1 ;
206   3                     IF FALL$DELAY$COUNTER > FALL$DELAY$COUNT
                            THEN
207   3                         DO ;
208   4                             FALL$DELAY$COUNTER = 0 ;
209   4                             FALL$DELAY$FLAG = FALSE ;
210   4                             FALL$FLAG = TRUE ;
211   4                             WIRE$ENABLE = FALSE ;
212   4                             CALL SET$POWER$SOURCE$CONTROL ;
213   4                         END ;
214   3                 END ;
215   2             IF FALL$FLAG
                    THEN
216   2                 DO ;
217   3                     FALL$COUNTER = FALL$COUNTER + 1 ;
218   3                     IF FALL$COUNTER > FALL$COUNT
                            THEN
219   3                         DO ;
220   4                             FALL$COUNTER = 0 ;
221   4                             FALL$FLAG,
                                    PULSE$ON$FLAG,
                                    PULSE$OFF$FLAG = FALSE ;
222   4                             PULSE$ON$COUNTER = 0 ;
223   4                             PULSE$OFF$COUNTER = 0 ;
224   4                             OUTPUT (CURRENT$CONTROL) = 0 ;
225   4                         END;
226   3                 END;
227   2             IF PULSE$OFF$FLAG
                    THEN
228   2                 DO ;
229   3                     PULSE$OFF$COUNTER = PULSE$OFF$COUNTER + 1 ;
230   3                     IF PULSE$OFF$COUNTER > PULSE$OFF$COUNT
                            THEN
231   3                         DO ;
232   4                             PULSE$OFF$COUNTER = 0 ;
233   4                             PULSE$OFF$FLAG = FALSE ;
234   4                             PULSE$ON$FLAG = TRUE ;

235   4                             IF FALL$FLAG
```

```
                                THEN
236  4                              ; /* DUMMY */
                                ELSE
237  4                              OUTPUT (CURRENT$CONTROL) = PULSE$CURRENT$BYTE ;
238  4                      END ;
239  3                  END ;
240  2              IF PULSE$ON$FLAG
                    THEN
241  2                  DO ;
242  3                      PULSE$ON$COUNTER = PULSE$ON$COUNTER + 1 ;
243  3                      IF PULSE$ON$COUNTER > PULSE$ON$COUNT
                            THEN
244  3                          DO ;
245  4                              PULSE$ON$COUNTER = 0 ;
246  4                              PULSE$ON$FLAG = FALSE ;
247  4                              PULSE$OFF$FLAG = TRUE ;
248  4                              OUTPUT (CURRENT$CONTROL) = BACKGROUND$CURRENT$BYTE ;
249  4                          END ;
                            ELSE
250  3                          DO ;
251  4                              IF (FALL$FLAG AND NOT(MATH$IN$USE))
                                    THEN
252  4                                  DO ;
253  5                                      CALL REAL$SUBTRACT (.PULSE$CURRENT$FLOAT, .BACKGROUND$CURRENT$FLOAT) ;
254  5                                      OUTPUT (ARITHMETIC$DATA) = LOW (FALL$COUNTER) ;
255  5                                      OUTPUT (ARITHMETIC$DATA) = HIGH (FALL$COUNTER) ;
256  5                                      OUTPUT (ARITHMETIC$COMMAND) = FLTS ;
257  5                                      CALL WAIT$FOR$ARITHMETIC$PROCESSOR ;
258  5                                      CALL PUSH$DATA$TO$APU (.RESULT) ;
259  5                                      OUTPUT (ARITHMETIC$COMMAND) = FMUL ;
260  5                                      CALL WAIT$FOR$ARITHMETIC$PROCESSOR ;
261  5                                      CALL POP$RESULT$FROM$APU ;
262  5                                      CALL REAL$DIVIDE (.RESULT, .FALL$COUNT$FLOAT) ;
263  5                                      CALL REAL$SUBTRACT (.PULSE$CURRENT$FLOAT, .RESULT) ;
264  5                                      CALL REAL$DIVIDE (.RESULT, .CURRENT$INCREMENT$FLOAT) ;
265  5                                      CALL FIX$8$BIT (.RESULT) ;
266  5                                      OUTPUT (CURRENT$CONTROL) = RESULT$SINGLE (1) ;
267  5                                  END ;
268  4                          END ;
269  3                  END ;
270  2              IF DELAY$FLAG
                    THEN
271  2                  DO ;
272  3                      DELAY$COUNTER = DELAY$COUNTER + 1 ;
273  3                      IF DELAY$COUNTER > DELAY$COUNT
                            THEN
274  3                          DO ;
275  4                              DELAY$COUNTER = 0 ;
276  4                              DELAY$FLAG = FALSE ;
277  4                          END ;
278  3                  END ;
279  2          END REAL$TIME$INTERRUPT ;
280  1      END REAL$TIME$INTERRUPT$PROCEDURE ;
```

MODULE INFORMATION:

```
    CODE AREA SIZE     = 016CH    364D
    VARIABLE AREA SIZE = 0000H     0D
    MAXIMUM STACK SIZE = 0002H     2D
    689 LINES READ
    0 PROGRAM ERROR(S)

END OF PL/M-80 COMPILATION
```

```
ISIS-II PL/M-80 V3.1 COMPILATION OF MODULE SEQUENCEPROCEDURE
OBJECT MODULE PLACED IN :F1:WELD18.OBJ
COMPILER INVOKED BY:  PLM80 :F1:WELD18.PLM DEBUG WORKFILES(:F1:,:F1:) PRINT(:F1:WELD18.LST)


          $TITLE ('SEQUENCE$PROCEDURE    VER 1.00     DATE 27/07/81')
 1        SEQUENCE$PROCEDURE: DO;

          /* A PLM80 PROGRAM TO ALLOW THE USER TO SET THE SEQUENCE IN WHICH GROUPS OF
             WELDING PARAMETERS WILL BE USED DURING WELDING.
             EACH WELD SEAM MAY BE ASSIGNED A GROUP OF PARAMETERS BY ENTERING THE
             CORRESPONDING GROUP NUMBER INTO THE RELEVANT POSITION IN THE ARRAY 'SEQUENCE$TABLE'.
          */

          /* EXTERNALS */
          $INCLUDE (:F1:WELD.EQU)
      =   $NOLIST
      =    *
 3  1        WRITE$MESSAGE: PROCEDURE (MESSAGE$ADDRESS) EXTERNAL ;
 4  2            DECLARE MESSAGE$ADDRESS ADDRESS ;
 5  2        END WRITE$MESSAGE ;

 6  1        POSITION$CURSOR: PROCEDURE (COLUMN$NUMBER, LINE$NUMBER) EXTERNAL ;
 7  2            DECLARE (COLUMN$NUMBER, LINE$NUMBER) BYTE ;
 8  2        END POSITION$CURSOR ;

 9  1        DISPLAY$BYTE: PROCEDURE (BYTE$VALUE) EXTERNAL ;
10  2            DECLARE BYTE$VALUE BYTE ;
11  2        END DISPLAY$BYTE ;

12  1        READ$BYTE$VALUE: PROCEDURE BYTE EXTERNAL ;
13  2        END READ$BYTE$VALUE ;

14  1        DISPLAY$X: PROCEDURE EXTERNAL ;
15  2        END DISPLAY$X ;

16  1        DISPLAY$Y: PROCEDURE EXTERNAL ;
17  2        END DISPLAY$Y ;

18  1        DISPLAY$Z: PROCEDURE EXTERNAL ;
19  2        END DISPLAY$Z ;

20  1        DISPLAY$R: PROCEDURE (R$NUMBER) EXTERNAL ;
21  2            DECLARE R$NUMBER BYTE ;
22  2        END DISPLAY$R ;

23  1        DISPLAY$CURRENT$RANGE: PROCEDURE EXTERNAL ;
24  2        END DISPLAY$CURRENT$RANGE ;

25  1        DISPLAY$CURRENT$INCREMENT: PROCEDURE EXTERNAL ;
26  2        END DISPLAY$CURRENT$INCREMENT ;

27  1        DISPLAY$PATH$NAME: PROCEDURE EXTERNAL ;
28  2        END DISPLAY$PATH$NAME ;

29  1        DISPLAY$REST$POSITION: PROCEDURE EXTERNAL ;
```

```
30  2         END DISPLAY$REST$POSITION ;

31  1         CONSOLE$OUTPUT: PROCEDURE (BYTE$VALUE) EXTERNAL ;
32  2            DECLARE BYTE$VALUE BYTE ;
33  2         END CONSOLE$OUTPUT ;

34  1         DECLARE BYTE$VALUE BYTE EXTERNAL ;
35  1         DECLARE SIGN$ON$MESSAGE BYTE EXTERNAL ;



        /* PUBLIC DATA */
36  1         DECLARE SEQUENCE$TABLE (60) BYTE PUBLIC ;



        /* LOCAL DATA */
37  1         DECLARE (I,
                    J,
                    SEAM ) BYTE ;
```

```
                $EJECT
 38   1         SEQUENCE: PROCEDURE PUBLIC ;
 39   2            CALL WRITE$MESSAGE (.(CLEAR$SCREEN,
                                         CURSOR$HOME,
                                         'WELDING PARAMETER SEQUENCE',
                                          CR$LF, CR$LF,
                    ' SEAM GROUP 1 SEAM GROUP 1 SEAM GROUP 1 SEAM GROUP 1 SEAM GROUP 1 SEAM GROUP', EOT)) ;
 40   2            DO I = 0 TO 5 ;
 41   3               DO J = 0 TO 9 ;
 42   4                  CALL POSITION$CURSOR ( ((I$13)+2), (J+3) ) ;
 43   4                  CALL DISPLAY$BYTE ( (10$I)+J ) ;
 44   4                  CALL WRITE$MESSAGE (.('  ', EOT)) ;
 45   4                  CALL DISPLAY$BYTE (SEQUENCE$TABLE((10$I)+J)) ;
 46   4               END ;
 47   3            END ;
 48   2            DO FOREVER ;
 49   3               CALL POSITION$CURSOR (PROMPT$FIELD) ;
 50   3               CALL WRITE$MESSAGE (.(DELETE$LINE,
                                            'SEAM ? ', EOT)) ;
 51   3               IF NOT (READ$BYTE$VALUE)
                      THEN
 52   3                  DO ;
 53   4                     CALL WRITE$MESSAGE (.SIGN$ON$MESSAGE) ;
 54   4                     CALL DISPLAY$CURRENT$RANGE ;
 55   4                     CALL DISPLAY$CURRENT$INCREMENT ;
 56   4                     CALL DISPLAY$PATH$NAME ;
 57   4                     CALL DISPLAY$REST$POSITION ;
 58   4                     CALL DISPLAY$X ;
 59   4                     CALL DISPLAY$Y ;
 60   4                     CALL DISPLAY$Z ;
 61   4                     CALL DISPLAY$R(0) ;
 62   4                     CALL DISPLAY$R(1) ;
 63   4                     CALL DISPLAY$R(2) ;
 64   4                     CALL DISPLAY$R(3) ;
 65   4                     RETURN ;
 66   4                  END ;
 67   3               IF (BYTE$VALUE < 60)
                      THEN
 68   3                  DO ;
 69   4                     SEAM = BYTE$VALUE ;
 70   4                     CALL POSITION$CURSOR (PROMPT$FIELD) ;
 71   4                     CALL WRITE$MESSAGE (.('SEAM ', EOT)) ;
 72   4                     CALL DISPLAY$BYTE (SEAM) ;
 73   4                     CALL WRITE$MESSAGE (.('  PARAMETER GROUP ? ', EOT)) ;
 74   4                     IF NOT (READ$BYTE$VALUE)
                            THEN
 75   4                        CALL CONSOLE$OUTPUT (BELL) ;
                            ELSE
 76   4                        DO ;
 77   5                           SEQUENCE$TABLE (SEAM) = BYTE$VALUE ;
 78   5                           CALL POSITION$CURSOR ( (((SEAM/10)$13)+7), ((SEAM MOD 10)+3) ) ;
 79   5                           CALL DISPLAY$BYTE (BYTE$VALUE) ;
 80   5                        END ;
 81   4                  END ;
                      ELSE
 82   3                  CALL CONSOLE$OUTPUT (BELL) ;
 83   3            END ;
```

```
 84  2         END SEQUENCE ;
 85  1     END SEQUENCE$PROCEDURE ;


MODULE INFORMATION:

    CODE AREA SIZE     = 01D3H    467D
    VARIABLE AREA SIZE = 003FH     63D
    MAXIMUM STACK SIZE = 0006H      6D
    340 LINES READ
    0 PROGRAM ERROR(S)

END OF PL/M-80 COMPILATION
```

```
ISIS-II PL/M-80 V3.1 COMPILATION OF MODULE PATHPROCEDURE
OBJECT MODULE PLACED IN :F1:WELD9.OBJ
COMPILER INVOKED BY:  PLM80 :F1:WELD9.PLM DEBUG WORKFILES(:F1:,:F1:) PRINT(:F1:WELD9.LST)



                $TITLE ('PATH$PROCEDURE  VER 3.00  DATE 20/08/81')
  1             PATH$PROCEDURE: DO ;

                /* EXTERNALS */
                $INCLUDE (:F1:WELD.EQU)
          =     $NOLIST
          =
                $INCLUDE (:F1:MATH.EQU)
          =     $NOLIST
                $INCLUDE (:F1:WELD1.EDS)
          =     $NOLIST
 80   1           MULTIPLY: PROCEDURE (B$ADDRESS, A$ADDRESS) EXTERNAL ;
 81   2               DECLARE (A$ADDRESS, B$ADDRESS) ADDRESS ;
 82   2           END MULTIPLY ;

 83   1           DIVIDE: PROCEDURE (B$ADDRESS, A$ADDRESS) EXTERNAL ;
 84   2               DECLARE (A$ADDRESS, B$ADDRESS) ADDRESS ;
 85   2           END DIVIDE ;

 86   1           SUBTRACT: PROCEDURE (B$ADDRESS, A$ADDRESS) EXTERNAL ;
 87   2               DECLARE (A$ADDRESS, B$ADDRESS) ADDRESS ;
 88   2           END SUBTRACT ;

 89   1           PUSH$DATA$TO$APU: PROCEDURE (DATA$ADDRESS) EXTERNAL ;
 90   2               DECLARE DATA$ADDRESS ADDRESS ;
 91   2           END PUSH$DATA$TO$APU ;

 92   1           WAIT$FOR$ARITHMETIC$PROCESSOR: PROCEDURE EXTERNAL ;
 93   2           END WAIT$FOR$ARITHMETIC$PROCESSOR ;

 94   1           SUBTRACT$SINGLE: PROCEDURE (B$ADDRESS, A$ADDRESS) EXTERNAL ;
 95   2               DECLARE (A$ADDRESS, B$ADDRESS) ADDRESS ;
 96   2           END SUBTRACT$SINGLE ;

 97   1           STORE$RESULT: PROCEDURE (STORE$ADDRESS) EXTERNAL ;
 98   2               DECLARE STORE$ADDRESS ADDRESS ;
 99   2           END STORE$RESULT ;

100   1           STORE$RESULT$SINGLE: PROCEDURE (STORE$ADDRESS) EXTERNAL ;
101   2               DECLARE STORE$ADDRESS ADDRESS ;
102   2           END STORE$RESULT$SINGLE ;

103   1           STORE$NUMBER$SINGLE: PROCEDURE (STORE$ADDRESS, DATA$ADDRESS) EXTERNAL ;
104   2               DECLARE (STORE$ADDRESS, DATA$ADDRESS) ADDRESS ;
105   2           END STORE$NUMBER$SINGLE ;

106   1           STORE$NUMBER: PROCEDURE (STORE$ADDRESS, DATA$ADDRESS) EXTERNAL ;
107   2               DECLARE (STORE$ADDRESS, DATA$ADDRESS) ADDRESS ;
108   2           END STORE$NUMBER ;

109   1           NEGATE: PROCEDURE (A$ADDRESS) EXTERNAL ;
```

```
110  2          DECLARE A$ADDRESS ADDRESS ;
111  2        END NEGATE ;

112  1        NEGATE$SINGLE: PROCEDURE (A$ADDRESS) EXTERNAL ;
113  2          DECLARE A$ADDRESS ADDRESS ;
114  2        END NEGATE$SINGLE ;

115  1        FIX$SINGLE: PROCEDURE (A$ADDRESS) EXTERNAL ;
116  2          DECLARE A$ADDRESS ADDRESS ;
117  2        END FIX$SINGLE ;

118  1        DECLARE (FOUR,
                      FIVE,
                      RESULT,
                      THREE$TWO$SEVEN$SIX$SEVEN,
                      POSITIONING$DIVISOR$FLOAT ) (4) BYTE EXTERNAL ;
119  1        DECLARE (TWO$THOUSAND$SINGLE,
                      TWO$SINGLE,
                      RESULT$SINGLE       ) (2) BYTE EXTERNAL ;
120  1        DECLARE MATH$STATUS BYTE EXTERNAL ;

121  1        SET$UP$X: PROCEDURE (DISPLACEMENT$ADDRESS) EXTERNAL ;
122  2          DECLARE DISPLACEMENT$ADDRESS ADDRESS ;
123  2        END SET$UP$X ;

124  1        SET$UP$Y: PROCEDURE (DISPLACEMENT$ADDRESS) EXTERNAL ;
125  2          DECLARE DISPLACEMENT$ADDRESS ADDRESS ;
126  2        END SET$UP$Y ;

127  1        SET$UP$Z: PROCEDURE (DISPLACEMENT$ADDRESS) EXTERNAL ;
128  2          DECLARE DISPLACEMENT$ADDRESS ADDRESS ;
129  2        END SET$UP$Z ;

130  1        ROTATE: PROCEDURE (R$NUMBER, DISPLACEMENT$SINGLE$ADDRESS) EXTERNAL ;
131  2          DECLARE R$NUMBER BYTE ;
132  2          DECLARE DISPLACEMENT$SINGLE$ADDRESS ADDRESS ;
133  2        END ROTATE ;

134  1        RELEASE$KEYS: PROCEDURE EXTERNAL ;
135  2        END RELEASE$KEYS ;

136  1        RUN$MOTORS: PROCEDURE EXTERNAL ;
137  2        END RUN$MOTORS ;

138  1        POSITION: PROCEDURE EXTERNAL ;
139  2        END POSITION ;

140  1        MOVE: PROCEDURE (AUTO$DISABLE) EXTERNAL ;
141  2          DECLARE AUTO$DISABLE BYTE ;
142  2        END MOVE ;

143  1        SET$CONTROLLER$INDICATORS: PROCEDURE EXTERNAL ;
144  2        END SET$CONTROLLER$INDICATORS ;

145  1        DECLARE MOTORS$RUNNING BYTE EXTERNAL ;
146  1        DECLARE (X$ORDINATE,
                      Y$ORDINATE,
```

```
                        Z$ORDINATE )  (4) BYTE EXTERNAL ;
147   1           DECLARE R$ORDINATES (4) STRUCTURE (ORDINATE (4) BYTE) EXTERNAL ;
148   1           DECLARE (R$0$ENABLED,
                           R$1$ENABLED,
                           R$2$ENABLED,
                           R$3$ENABLED,
                           CONTROLLER$ACTIVE,
                           ON$SEAM           ) BYTE EXTERNAL ;
149   1           DECLARE (ROTATION$ENABLED,
                           LINEAR$ENABLED,
                           FAST$KEY,
                           SLOW$KEY,
                           ENTER$KEY,
                           START$WELD$KEY,
                           STOP$WELD$KEY              ) BYTE EXTERNAL ;

                  /* A PLM80 PROCEDURE TO ALLOW THE USER TO ENTER A WELDING PATH INTO SYSTEM MEMORY.
                     THE PATH IS ENTERED ON A POINT BY POINT BASIS, THE TORCH BEING MOVED FROM ONE POINT TO
                     THE NEXT USING THE HAND HELD CONTROLLER..
                     ALL PATHS MUST START AND FINISH AT THE SAME POINT i.e. THE REST POSITION.
                     THE REST POSITION IS DEFINED BY THE USER WHEN HE ENTERS A PATH.
                  */
```

```
            $EJECT

            /* PUBLIC DATA */
150   1         DECLARE DISPLACEMENT$TABLE (1000) STRUCTURE (IDENTIFIER BYTE,
                                                           DISPLACEMENT$1$SINGLE (2) BYTE,
                                                           DISPLACEMENT$2$SINGLE (2) BYTE,
                                                           DISPLACEMENT$3$SINGLE (2) BYTE) PUBLIC ;
            /* IDENTIFIER        BIT$0 = START WELD
                                 BIT$1 = STOP  WELD
                                 BIT$2 = LINEAR$ENABLE
                                 BIT$3 = R0$ENABLE
                                 BIT$4 = R1$ENABLE
                                 BIT$5 = R2$ENABLE
                                 BIT$6 = R3$ENABLE
            */
151   1         DECLARE PATH$NAME (16) BYTE PUBLIC ;
152   1         DECLARE (REST$POSITION$X,
                         REST$POSITION$Y,
                         REST$POSITION$Z,
                         REST$POSITION$R$0,
                         REST$POSITION$R$1,
                         REST$POSITION$R$2,
                         REST$POSITION$R$3  ) (4) BYTE PUBLIC ;
153   1         DECLARE (SEAM$COUNT,
                         DISPLACEMENT$ERROR)    BYTE PUBLIC ;
154   1         DECLARE  POINT$COUNT         ADDRESS PUBLIC ;
155   1         DECLARE DISPLACEMENT$SINGLE (2) BYTE PUBLIC ;
156   1         DECLARE DISPLACEMENT        (4) BYTE PUBLIC ;
157   1         DECLARE ZERO$DISPLACEMENT       BYTE PUBLIC ;
158   1         DECLARE PATH$SET                BYTE PUBLIC ;

            /* LOCAL DATA */
159   1         DECLARE (LAST$X$ORDINATE,
                         LAST$Y$ORDINATE,
                         LAST$Z$ORDINATE,
                         LAST$R$0$ORDINATE,
                         LAST$R$1$ORDINATE,
                         LAST$R$2$ORDINATE,
                         LAST$R$3$ORDINATE  ) (4) BYTE ;
160   1         DECLARE (R$0$DISPLACEMENT$SINGLE,
                         R$1$DISPLACEMENT$SINGLE,
                         R$2$DISPLACEMENT$SINGLE,
                         R$3$DISPLACEMENT$SINGLE) (2) BYTE ;
161   1         DECLARE (I,
                         IDENTIFIER           ) BYTE ;
162   1         DECLARE DISPLACEMENT$DIRECTION   BYTE ;
163   1         DECLARE (WELD$SEAM,
                         WELDING$FINISHED     ) BYTE ;
164   1         DECLARE ZERO$DISPLACEMENT$FLAG   BYTE ;
```

```
                $EJECT

165   1         SET$PATH$NAME: PROCEDURE ;
166   2           DECLARE I BYTE ;

167   2           CALL POSITION$CURSOR (PROMPT$FIELD) ;
168   2           CALL WRITE$MESSAGE (.(DELETE$LINE,
                                  'ENTER PATH NAME', EOT)) ;
169   2           CALL POSITION$CURSOR (COMMAND$FIELD) ;
170   2           CALL WRITE$MESSAGE (.(DELETE$LINE, EOT)) ;
171   2           CALL READ$LINE ;
172   2           DO I = 0 TO 15 ;
173   3             IF EDIT$BUFFER(I) = CARRIGE$RETURN
                    THEN
174   3               PATH$NAME(I) = SPACE ;
                    ELSE
175   3               PATH$NAME(I) = EDIT$BUFFER(I) ;
176   3             END ;
177   2           PATH$NAME$SET = TRUE ;
178   2           CALL DISPLAY$PATH$NAME ;
179   2           CALL CLEAR$PROMPT$FIELD ;
180   2         END SET$PATH$NAME ;

181   1         DISPLAY$REST$POSITION: PROCEDURE PUBLIC ;
182   2           CALL POSITION$CURSOR (REST$POSITION$FIELD) ;
183   2           CALL CONSOLE$OUTPUT (DELETE$LINE) ;
184   2           IF REST$POSITION$SET
                    THEN
185   2             DO ;
186   3               CALL WRITE$MESSAGE (.(BACKGROUND$ON,
                                      'START  X= ', EOT)) ;
187   3               CALL DISPLAY$IN$NORMALISED$UNITS (.REST$POSITION$X) ;
188   3               CALL WRITE$MESSAGE (.('    Y= ', EOT)) ;
189   3               CALL MULTIPLY (.REST$POSITION$Y, .FIVE) ;
190   3               CALL DIVIDE (.RESULT, .FOUR) ;
191   3               CALL DISPLAY$IN$NORMALISED$UNITS (.RESULT) ;
192   3               CALL WRITE$MESSAGE (.('    Z= ', EOT)) ;
193   3               CALL DISPLAY$IN$NORMALISED$UNITS (.REST$POSITION$Z) ;
194   3               CALL WRITE$MESSAGE (.(CR$LF, '    R0= ', EOT)) ;
195   3               CALL CONVERT$TO$ASCII (.REST$POSITION$R$0) ;
196   3               CALL CONSOLE$OUTPUT (ASCII(0)) ;
197   3               CALL DISPLAY$ASCII$BUFFER (1, 10, ENABLE$BLANKING) ;
198   3               CALL WRITE$MESSAGE (.('  R1= ', EOT)) ;
199   3               CALL CONVERT$TO$ASCII (.REST$POSITION$R$1) ;
200   3               CALL CONSOLE$OUTPUT (ASCII(0)) ;
201   3               CALL DISPLAY$ASCII$BUFFER (1, 10, ENABLE$BLANKING) ;
202   3               CALL WRITE$MESSAGE (.('  R2= ', EOT)) ;
203   3               CALL CONVERT$TO$ASCII (.REST$POSITION$R$2) ;
204   3               CALL CONSOLE$OUTPUT (ASCII(0)) ;
205   3               CALL DISPLAY$ASCII$BUFFER (1, 10, ENABLE$BLANKING) ;
206   3               CALL WRITE$MESSAGE (.('  R3= ', EOT)) ;
207   3               CALL CONVERT$TO$ASCII (.REST$POSITION$R$3) ;
208   3               CALL CONSOLE$OUTPUT (ASCII(0)) ;
209   3               CALL DISPLAY$ASCII$BUFFER (1, 10, ENABLE$BLANKING) ;
210   3               CALL CONSOLE$OUTPUT (BACKGROUND$OFF) ;
211   3             END ;
212   2         END DISPLAY$REST$POSITION ;
```

```
213  1          ERROR: PROCEDURE ;
214  2              DISPLACEMENT$ERROR = TRUE ;
215  2              CALL POSITION$CURSOR (ERROR$FIELD) ;
216  2              CALL WRITE$MESSAGE (.(DELETE$LINE, 'DISPLACEMENT ERROR', EOT)) ;
217  2          END ERROR ;

218  1          CALCULATE$DISPLACEMENT: PROCEDURE (THIS$ORDINATE$ADDR, LAST$ORDINATE$ADDR) ;
219  2              DECLARE (THIS$ORDINATE$ADDR, LAST$ORDINATE$ADDR) ADDRESS ;
220  2              CALL SUBTRACT (THIS$ORDINATE$ADDR, LAST$ORDINATE$ADDR) ;
221  2              IF MATH$OVERFLOW
                     THEN
222  2                  CALL ERROR ;
223  2              IF MATH$ZERO
                     THEN
224  2                  ZERO$DISPLACEMENT = TRUE ;
                     ELSE
225  2                  ZERO$DISPLACEMENT = FALSE ;
226  2              IF MATH$NEGATIVE
                     THEN
227  2                  DO ;
228  3                      CALL PUSH$DATA$TO$APU (.RESULT) ;
229  3                      CALL PUSH$DATA$TO$APU (.THREE$TWO$SEVEN$SIX$SEVEN) ;
230  3                      OUTPUT (ARITHMETIC$COMMAND) = CHSD ;
231  3                      CALL WAIT$FOR$ARITHMETIC$PROCESSOR ;
232  3                  END ;
                     ELSE
233  2                  DO ;
234  3                      CALL PUSH$DATA$TO$APU (.THREE$TWO$SEVEN$SIX$SEVEN) ;
235  3                      CALL PUSH$DATA$TO$APU (.RESULT) ;
236  3                  END ;
237  2              OUTPUT (ARITHMETIC$COMMAND) = DSUB ;
238  2              CALL WAIT$FOR$ARITHMETIC$PROCESSOR ;
239  2              MATH$STATUS = INPUT (ARITHMETIC$STATUS) ;
240  2              IF MATH$NEGATIVE
                     THEN
241  2                  CALL ERROR ;
242  2              CALL STORE$RESULT (.DISPLACEMENT) ;
243  2              DISPLACEMENT$SINGLE(0) = RESULT (2) ;
244  2              DISPLACEMENT$SINGLE(1) = RESULT (3) ;
245  2          END CALCULATE$DISPLACEMENT ;

246  1          CHECK$DISPLACEMENT: PROCEDURE (THIS$ORDINATE$ADDR, LAST$ORDINATE$ADDR) PUBLIC ;
247  2              DECLARE (THIS$ORDINATE$ADDR, LAST$ORDINATE$ADDR) ADDRESS ;
248  2              CALL CALCULATE$DISPLACEMENT (THIS$ORDINATE$ADDR, LAST$ORDINATE$ADDR) ;
249  2              IF ROL (DISPLACEMENT(0), 1)
                     THEN
250  2                  DO ;
251  3                      DISPLACEMENT$DIRECTION = FALSE ;
252  3                      CALL NEGATE$SINGLE (.DISPLACEMENT$SINGLE) ;
253  3                      CALL STORE$RESULT$SINGLE (.DISPLACEMENT$SINGLE) ;
254  3                      CALL NEGATE (.DISPLACEMENT) ;
255  3                      CALL STORE$RESULT (.DISPLACEMENT) ;
256  3                  END ;
                     ELSE
257  2                  DISPLACEMENT$DIRECTION = TRUE ;
258  2              CALL SUBTRACT$SINGLE (.TWO$THOUSAND$SINGLE, .DISPLACEMENT$SINGLE) ;
```

```
259  2          IF MATH$NEGATIVE
                THEN
260  2             DISPLACEMENT$ERROR = TRUE ;
261  2          CALL SUBTRACT$SINGLE (.DISPLACEMENT$SINGLE, .TWO$SINGLE) ;
262  2          IF ( MATH$NEGATIVE AND (NOT ZERO$DISPLACEMENT) )
                THEN
263  2             DISPLACEMENT$ERROR = TRUE ;
264  2       END CHECK$DISPLACEMENT ;

265  1       HOME$TORCH: PROCEDURE PUBLIC ;
266  2          DO WHILE MOTORS$RUNNING ;
267  3          END ;
268  2          CALL FIX$SINGLE (.POSITIONING$DIVISOR$FLOAT) ;
269  2          OUTPUT (X$DIVIDER) = RESULT$SINGLE(1) ;
270  2          OUTPUT (X$DIVIDER) = RESULT$SINGLE(0) ;
271  2          OUTPUT (Y$DIVIDER) = RESULT$SINGLE(1) ;
272  2          OUTPUT (Y$DIVIDER) = RESULT$SINGLE(0) ;
273  2          OUTPUT (Z$DIVIDER) = RESULT$SINGLE(1) ;
274  2          OUTPUT (Z$DIVIDER) = RESULT$SINGLE(0) ;
275  2          DISPLACEMENT$ERROR = FALSE ;
276  2          CALL CHECK$DISPLACEMENT (.REST$POSITION$X, .X$ORDINATE) ;
277  2          IF (NOT (DISPLACEMENT$ERROR OR ZERO$DISPLACEMENT))
                THEN
278  2             DO ;
279  3                X$POSITIVE = DISPLACEMENT$DIRECTION ;
280  3                X$ENABLED = TRUE ;
281  3                CALL SET$UP$X (.DISPLACEMENT) ;
282  3             END ;
283  2          CALL CHECK$DISPLACEMENT (.REST$POSITION$Y, .Y$ORDINATE) ;
284  2          IF (NOT (DISPLACEMENT$ERROR OR ZERO$DISPLACEMENT))
                THEN
285  2             DO ;
286  3                Y$POSITIVE = DISPLACEMENT$DIRECTION ;
287  3                Y$ENABLED  = TRUE ;
288  3                CALL SET$UP$Y (.DISPLACEMENT) ;
289  3             END ;
290  2          CALL CHECK$DISPLACEMENT (.REST$POSITION$Z, .Z$ORDINATE) ;
291  2          IF (NOT (DISPLACEMENT$ERROR OR ZERO$DISPLACEMENT))
                THEN
292  2            DO ;
293  3               Z$POSITIVE = DISPLACEMENT$DIRECTION ;
294  3               Z$ENABLED  = TRUE ;
295  3               CALL SET$UP$Z (.DISPLACEMENT) ;
296  3            END ;
297  2          CALL CALCULATE$DISPLACEMENT (.REST$POSITION$R$0, .R$ORDINATES(0).ORDINATE) ;
298  2          CALL STORE$NUMBER$SINGLE (.R$0$DISPLACEMENT$SINGLE,
                                          .DISPLACEMENT$SINGLE      ) ;
299  2          CALL CALCULATE$DISPLACEMENT (.REST$POSITION$R$1, .R$ORDINATES(1).ORDINATE) ;
300  2          CALL STORE$NUMBER$SINGLE (.R$1$DISPLACEMENT$SINGLE,
                                          .DISPLACEMENT$SINGLE      ) ;
301  2          CALL CALCULATE$DISPLACEMENT (.REST$POSITION$R$2, .R$ORDINATES(2).ORDINATE) ;
302  2          CALL STORE$NUMBER$SINGLE (.R$2$DISPLACEMENT$SINGLE,
                                          .DISPLACEMENT$SINGLE      ) ;
303  2          CALL CALCULATE$DISPLACEMENT (.REST$POSITION$R$3, .R$ORDINATES(3).ORDINATE) ;
304  2          CALL STORE$NUMBER$SINGLE (.R$3$DISPLACEMENT$SINGLE,
                                          .DISPLACEMENT$SINGLE      ) ;
305  2          CALL RUN$MOTORS ;
```

```
306  2          CALL ROTATE (0, .R$0$DISPLACEMENT$SINGLE) ;
307  2          CALL ROTATE (1, .R$1$DISPLACEMENT$SINGLE) ;
308  2          CALL ROTATE (2, .R$2$DISPLACEMENT$SINGLE) ;
309  2          CALL ROTATE (3, .R$3$DISPLACEMENT$SINGLE) ;
310  2      END HOME$TORCH ;

311  1      ENTER$PATH$POINT: PROCEDURE ;
312  2          ZERO$DISPLACEMENT$FLAG = TRUE ;
313  2          IF (ROTATION$ENABLED XOR LINEAR$ENABLED)
                 THEN
314  2              DO ;
315  3                  DISPLACEMENT$ERROR = FALSE ;
316  3                  IF ROTATION$ENABLED
                        THEN
317  3                      DO ;
318  4                          IF R$0$ENABLED
                                THEN
319  4                              DO ;
320  5                                  CALL CALCULATE$DISPLACEMENT (.R$ORDINATES(0).ORDINATE,
                                                                     .LAST$R$0$ORDINATE) ;
321  5                                  IDENTIFIER = 0000$1000B ;
322  5                              END ;
323  4                          IF R$1$ENABLED
                                THEN
324  4                              DO ;
325  5                                  CALL CALCULATE$DISPLACEMENT (.R$ORDINATES(1).ORDINATE,
                                                                     .LAST$R$1$ORDINATE) ;
326  5                                  IDENTIFIER = 0001$0000B ;
327  5                              END ;
328  4                          IF R$2$ENABLED
                                THEN
329  4                              DO ;
330  5                                  CALL CALCULATE$DISPLACEMENT (.R$ORDINATES(2).ORDINATE,
                                                                     .LAST$R$2$ORDINATE) ;
331  5                          IDENTIFIER = 0010$0000B ;
332  5                              END ;
333  4                          IF R$3$ENABLED
                                THEN
334  4                              DO ;
335  5                                  CALL CALCULATE$DISPLACEMENT (.R$ORDINATES(3).ORDINATE,
                                                                     .LAST$R$3$ORDINATE) ;
336  5                                  IDENTIFIER = 0100$0000B ;
337  5                              END ;
338  4                          ZERO$DISPLACEMENT$FLAG = ZERO$DISPLACEMENT ;
339  4                          CALL STORE$NUMBER$SINGLE (
                                    .DISPLACEMENT$TABLE(POINT$COUNT).DISPLACEMENT$1$SINGLE,
                                    .DISPLACEMENT$SINGLE) ;
340  4                          IF (NOT (DISPLACEMENT$ERROR OR ZERO$DISPLACEMENT))
                                THEN
341  4                              DO ;
342  5                                  DISPLACEMENT$TABLE(POINT$COUNT).IDENTIFIER =
                                       (DISPLACEMENT$TABLE(POINT$COUNT).IDENTIFIER) OR (IDENTIFIER) ;
343  5                                  POINT$COUNT = POINT$COUNT + 1 ;
344  5                              END ;
345  4                      END ;
                        ELSE
346  3                      DO ;
```

```
347  4                          CALL CALCULATE$DISPLACE$MENT (.X$ORDINATE, .LAST$X$ORDINATE) ;
348  4                          CALL STORE$NUMBER$SINGLE (.DISPLACEMENT$TABLE(POINT$COUNT).DISPLACEMENT$1$SINGLE,
                                                          .DISPLACEMENT$SINGLE) ;
349  4                          ZERO$DISPLACEMENT$FLAG = ZERO$DISPLACEMENT$FLAG AND
                                                          ZERO$DISPLACEMENT ;
350  4                          CALL CALCULATE$DISPLACEMENT (.Y$ORDINATE, .LAST$Y$ORDINATE) ;
351  4                          CALL STORE$NUMBER$SINGLE (.DISPLACEMENT$TABLE(POINT$COUNT).DISPLACEMENT$2$SINGLE,
                                                          .DISPLACEMENT$SINGLE) ;
352  4                          ZERO$DISPLACEMENT$FLAG = ZERO$DISPLACEMENT$FLAG AND
                                                          ZERO$DISPLACEMENT ;
353  4                          CALL CALCULATE$DISPLACEMENT (.Z$ORDINATE, .LAST$Z$ORDINATE) ;
354  4                          CALL STORE$NUMBER$SINGLE (.DISPLACEMENT$TABLE(POINT$COUNT).DISPLACEMENT$3$SINGLE,
                                                          .DISPLACEMENT$SINGLE) ;
355  4                          ZERO$DISPLACEMENT$FLAG = ZERO$DISPLACEMENT$FLAG AND
                                                          ZERO$DISPLACEMENT ;
356  4                          IF (NOT (DISPLACEMENT$ERROR OR ZERO$DISPLACEMENT$FLAG))
                                THEN
357  4                              DO ;
358  5                                  DISPLACEMENT$TABLE(POINT$COUNT).IDENTIFIER =
                                       (DISPLACEMENT$TABLE(POINT$COUNT).IDENTIFIER) OR (0000$0100B) ;
359  5                                  POINT$COUNT = POINT$COUNT + 1 ;
360  5                              END ;
361  4                      END ;
362  3                  CALL POSITION$CURSOR (ERROR$FIELD) ;
363  3                  IF DISPLACEMENT$ERROR
                        THEN
364  3                      CALL WRITE$MESSAGE (.(DELETE$LINE,
                                                  'DISPLACEMENT TOO LARGE', BLINK$ON,
                                                   BLINK$OFF, BELL, EOT      )) ;
                        ELSE
365  3                      DO ;
366  4                          CALL CONSOLE$OUTPUT (DELETE$LINE) ;
367  4                          CALL STORE$NUMBER (.LAST$X$ORDINATE, .X$ORDINATE) ;
368  4                          CALL STORE$NUMBER (.LAST$Y$ORDINATE, .Y$ORDINATE) ;
369  4                          CALL STORE$NUMBER (.LAST$Z$ORDINATE, .Z$ORDINATE) ;
370  4                          CALL STORE$NUMBER (.LAST$R$0$ORDINATE, .R$ORDINATES(0).ORDINATE) ;
371  4                          CALL STORE$NUMBER (.LAST$R$1$ORDINATE, .R$ORDINATES(1).ORDINATE) ;
372  4                          CALL STORE$NUMBER (.LAST$R$2$ORDINATE, .R$ORDINATES(2).ORDINATE) ;
373  4                          CALL STORE$NUMBER (.LAST$R$3$ORDINATE, .R$ORDINATES(3).ORDINATE) ;
374  4                      END ;
375  3              END ;
376  2      END ENTER$PATH$POINT ;
```

```
                $EJECT
377   1         PATH: PROCEDURE PUBLIC ;
378   2             CALL POSITION$CURSOR (PROG$FIELD) ;
379   2             CALL WRITE$MESSAGE (.(DELETE$LINE, 'PATH', EOT)) ;
380   2             CALL SET$PATH$NAME ;
381   2             CALL POSITION$CURSOR (PROMPT$FIELD) ;
382   2             CALL WRITE$MESSAGE (.(DELETE$LINE,
                                  'MOVE TORCH TO REST POSITION', EOT)) ;
383   2             CALL POSITION ;
384   2             CALL STORE$NUMBER (.REST$POSITION$X, .X$ORDINATE) ;
385   2             CALL STORE$NUMBER (.LAST$X$ORDINATE, .X$ORDINATE) ;
386   2             CALL STORE$NUMBER (.REST$POSITION$Y, .Y$ORDINATE) ;
387   2             CALL STORE$NUMBER (.LAST$Y$ORDINATE, .Y$ORDINATE) ;
388   2             CALL STORE$NUMBER (.REST$POSITION$Z, .Z$ORDINATE) ;
389   2             CALL STORE$NUMBER (.LAST$Z$ORDINATE, .Z$ORDINATE) ;
390   2             CALL STORE$NUMBER (.REST$POSITION$R$0, .R$ORDINATES(0).ORDINATE) ;
391   2             CALL STORE$NUMBER (.LAST$R$0$ORDINATE, .R$ORDINATES(0).ORDINATE) ;
392   2             CALL STORE$NUMBER (.REST$POSITION$R$1, .R$ORDINATES(1).ORDINATE) ;
393   2             CALL STORE$NUMBER (.LAST$R$1$ORDINATE, .R$ORDINATES(1).ORDINATE) ;
394   2             CALL STORE$NUMBER (.REST$POSITION$R$2, .R$ORDINATES(2).ORDINATE) ;
395   2             CALL STORE$NUMBER (.LAST$R$2$ORDINATE, .R$ORDINATES(2).ORDINATE) ;
396   2             CALL STORE$NUMBER (.REST$POSITION$R$3, .R$ORDINATES(3).ORDINATE) ;
397   2             CALL STORE$NUMBER (.LAST$R$3$ORDINATE, .R$ORDINATES(3).ORDINATE) ;
398   2             REST$POSITION$SET = TRUE ;
399   2             CALL DISPLAY$REST$POSITION ;
400   2             CALL POSITION$CURSOR (PROMPT$FIELD) ;
401   2             CALL WRITE$MESSAGE (.(DELETE$LINE,
                                  'ENTER PATH', EOT )) ;
402   2             WELD$SEAM = FALSE ;
403   2             ROTATION$ENABLED,
                    R$0$ENABLED,
                    R$1$ENABLED,
                    R$2$ENABLED,
                    R$3$ENABLED,
                    CONTROLLER$ACTIVE,
                    LINEAR$ENABLED    = TRUE ;
404   2             WELDING$FINISHED,
                    ON$SEAM           = FALSE ;
405   2             CALL SET$CONTROLLER$INDICATORS ;
406   2             PATH$SET = FALSE ;
407   2             POINT$COUNT = 0 ;
408   2             SEAM$COUNT  = 0 ;
409   2             DO I = 0 TO (MAX$POINTS - 1) ;
410   3                 DISPLACEMENT$TABLE(I).IDENTIFIER = 0 ;
411   3             END ;
412   2             DO WHILE POINT$COUNT < (MAX$POINTS - 3) ;
413   3                 DISPLACEMENT$ERROR = FALSE ;
414   3                 CALL MOVE (TRUE) ;
415   3                 IF (ENTER$KEY AND (NOT WELDING$FINISHED))
                        THEN
416   3                     DO ;
417   4                         CALL ENTER$PATH$POINT ;
418   4                         CALL RELEASE$KEYS ;
419   4                         IF (NOT DISPLACEMENT$ERROR)
                                THEN
420   4                             DO ;
421   5                                 LINEAR$ENABLED,
```

```
                                ROTATION$ENABLED,
                                R$0$ENABLED,
                                R$1$ENABLED,
                                R$2$ENABLED,
                                R$3$ENABLED      = TRUE ;
422  5                          CALL SET$CONTROLLER$INDICATORS ;
423  5                      END ;
424  4                  END ;
425  3              IF (START$WELD$KEY AND (NOT (WELDING$FINISHED OR WELD$SEAM)))
                    THEN
426  3                  DO ;
427  4                      CALL ENTER$PATH$POINT ;
428  4                      IF (NOT DISPLACEMENT$ERROR)
                            THEN
429  4                          DO ;
430  5                              WELD$SEAM = TRUE ;
431  5                              IF ROR (IDENTIFIER:= DISPLACEMENT$TABLE(POINT$COUNT).IDENTIFIER, 1)
                                    THEN
432  5                                  DISPLACEMENT$TABLE(POINT$COUNT).IDENTIFIER =
                                                        IDENTIFIER AND 1111$1101B ;
                                    ELSE
433  5                                  DISPLACEMENT$TABLE(POINT$COUNT).IDENTIFIER =
                                                        IDENTIFIER OR 0000$0001B ;
434  5                              LINEAR$ENABLED,
                                    ROTATION$ENABLED,
                                    R$0$ENABLED,
                                    R$1$ENABLED,
                                    R$2$ENABLED,
                                    R$3$ENABLED,
                                    ON$SEAM = TRUE ;
435  5                              CALL SET$CONTROLLER$INDICATORS ;
436  5                          END ;
437  4                      CALL RELEASE$KEYS ;
438  4                  END ;
439  3              IF STOP$WELD$KEY
                    THEN
440  3                  DO ;
441  4                      CALL ENTER$PATH$POINT ;
442  4                      CALL RELEASE$KEYS ;
443  4                      IF (NOT DISPLACEMENT$ERROR)
                            THEN
444  4                          DO ;
445  5                              IF (NOT WELD$SEAM)
                                    THEN
446  5                                  DO ;
447  6                                      IF (NOT WELDING$FINISHED)
                                            THEN
448  6                                          DO ;
449  7                                              CONTROLLER$ACTIVE = FALSE ;
450  7                                              CALL SET$CONTROLLER$INDICATORS ;
451  7                                              CALL POSITION$CURSOR (PROMPT$FIELD) ;
452  7                                              CALL WRITE$MESSAGE (.(DELETE$LINE,
                                                                        'ANY MORE WELDS ?',
                                                                        BELL, EOT )) ;
453  7                                              IF CONSOLE$INPUT = 'N'
                                                    THEN
454  7                                                  DO ;
```

```
455  8                                  WELDING$FINISHED = TRUE ;
456  8                                  CALL DISPLAY$X ;
457  8                                  CALL DISPLAY$Y ;
458  8                                  CALL DISPLAY$Z ;
459  8                                  CALL DISPLAY$R(0) ;
460  8                                  CALL DISPLAY$R(1) ;
461  8                                  CALL DISPLAY$R(2) ;
462  8                                  CALL DISPLAY$R(3) ;
463  8                                  CALL POSITION$CURSOR (PROMPT$FIELD) ;
464  8                                  CALL WRITE$MESSAGE (.(DELETE$LINE,
                                                    'MOVE TORCH TO REST POSITION',
                                                EOT)) ;
465  8                                   LINEAR$ENABLED,
                                         ROTATION$ENABLED,
                                         R$0$ENABLED,
                                         R$1$ENABLED,
                                         R$2$ENABLED,
                                         R$3$ENABLED    = TRUE ;
466  8                                 END ;
                                     ELSE
467  7                                   POINT$COUNT = POINT$COUNT - 1 ;
468  7                                 CONTROLLER$ACTIVE = TRUE ;
469  7                                 CALL SET$CONTROLLER$INDICATORS ;
470  7                               END ;
                                   ELSE
471  6                                DO ;
472  7                                   CALL HOME$TORCH ;
473  7                                   IF NOT DISPLACEMENT$ERROR
                                         THEN
474  7                                       DO ;
475  8                                         CALL CLEAR$PROMPT$FIELD ;
476  8                                         R$0$ENABLED,
                                               R$1$ENABLED,
                                               R$2$ENABLED,
                                               R$3$ENABLED,
                                               CONTROLLER$ACTIVE,
                                               ON$SEAM            = FALSE ;
477  8                                         CALL SET$CONTROLLER$INDICATORS ;
478  8                                         PATH$SET = TRUE ;
479  8                                         RETURN ;
480  8                                       END ;
481  7                                   CALL POSITION$CURSOR (PROMPT$FIELD) ;
482  7                                   CALL WRITE$MESSAGE (.(DELETE$LINE,
                                                        'MOVE TO REST POSITION',
                                                         BELL, EOT )) ;
483  7                                   CALL DISPLAY$X ;
484  7                                   CALL DISPLAY$Y ;
485  7                                   CALL DISPLAY$Z ;
486  7                                   CALL DISPLAY$R(0) ;
487  7                                   CALL DISPLAY$R(1) ;
488  7                                   CALL DISPLAY$R(2) ;
489  7                                   CALL DISPLAY$R(3) ;
490  7                                   LINEAR$ENABLED,
                                         ROTATION$ENABLED,
                                         R$0$ENABLED,
                                         R$1$ENABLED,
                                         R$2$ENABLED,
```

```
                                                R$3$ENABLED    = TRUE ;
491   7                                         CALL SET$CONTROLLER$INDICATORS ;
492   7                                     END ;
493   6                                 END ;
                                        ELSE
494   5                                     DO ;
495   6                                     WELD$SEAM    = FALSE ;
496   6                                     IF (IDENTIFIER:= DISPLACEMENT$TABLE(POINT$COUNT).IDENTIFIER)
                                            THEN
497   6                                         DISPLACEMENT$TABLE(POINT$COUNT).IDENTIFIER =
                                                                    IDENTIFIER AND 1111$1110B ;
                                            ELSE
498   6                                         DISPLACEMENT$TABLE(POINT$COUNT).IDENTIFIER
      =
                                                                    IDENTIFIER OR 0000$0010B ;
499   6                                     SEAM$COUNT   = SEAM$COUNT + 1 ;
500   6                                     ON$SEAM      = FALSE ;
501   6                                     LINEAR$ENABLED,
                                            ROTATION$ENABLED,
                                            R$0$ENABLED,
                                            R$1$ENABLED,
                                            R$2$ENABLED,
                                            R$3$ENABLED  = TRUE ;
502   6                                     CALL SET$CONTROLLER$INDICATORS ;
503   6                                 END ;
504   5                             END ;
505   4                         END ;
506   3                 END ;
507   2             CALL POSITION$CURSOR (ERROR$FIELD) ;
508   2             CALL WRITE$MESSAGE (.(DELETE$LINE,
                                        'TOO MANY POINTS',
                                         BELL, EOT         )) ;
509   2         END PATH ;
510   1     END PATH$PROCEDURE ;
```

MODULE INFORMATION:

```
    CODE AREA SIZE     = 09B3H   2483D
    VARIABLE AREA SIZE = 1BC3H   7107D
    MAXIMUM STACK SIZE = 000AH     10D
    1036 LINES READ
    0 PROGRAM ERROR(S)
```

END OF PL/M-80 COMPILATION

```
ISIS-II PL/M-80 V3.1 COMPILATION OF MODULE SETORIGINPROCEDURE
OBJECT MODULE PLACED IN :F1:WELD3.OBJ
COMPILER INVOKED BY:  PLM80 :F1:WELD3.PLM DEBUG WORKFILES(:F1:,:F1:) PRINT(:F1:WELD3.LST)



            $TITLE ('SET$ORIGIN$PROCEDURE   VER 2.00    DATE 20/03/81')
  1         SET$ORIGIN$PROCEDURE: DO ;

            /* A PLM80 ROUTINE TO ALLOW THE USER TO RE-DEFINE THE X, Y, Z ORIGIN.
               THE USER MOVES THE TORCH TO THE NEW ORIGIN AND THE  ROUTINE ASSIGNS
               THE CO-ORDINATES (0, 0, 0) TO THAT POSITION.
               RE-ASSIGNING THE ORIGIN INVALIDATES ANY PRE-DEFINED MOVEMENT LIMITS
               AND THUS THE ROUTINE FORCES THE USER TO RE-ENTER ALL LIMITS.
            */
            /* EXTERNALS */
            $INCLUDE (:F1:WELD.EQU)
        =   $NOLIST
        =
            $INCLUDE (:F1:WELD1.EDS)
        =   $NOLIST
            $INCLUDE (:F1:MATH.EDS)
        =   $NOLIST
            $INCLUDE (:F1:WELD5.EDS)
        =   $NOLIST
            $INCLUDE (:F1:WELD2.EDS)
        =   $NOLIST

            /* LOCAL DATA */
202   1         DECLARE ORIGIN$SET BYTE ;
```

```
            $EJECT
 203   1        SET$ORIGIN: PROCEDURE PUBLIC ;
 204   2            ORIGIN$SET = FALSE ;
 205   2            DO WHILE NOT (ORIGIN$SET) ;
 206   3                CALL POSITION$CURSOR (PROMPT$FIELD) ;
 207   3                CALL WRITE$MESSAGE (.(DELETE$LINE,
                                        'MOVE TORCH TO ORIGIN', EOT)) ;
 208   3                CALL POSITION ;
 209   3                CALL POSITION$CURSOR (PROMPT$FIELD) ;
 210   3                CALL WRITE$MESSAGE   (.(DELETE$LINE,
                                          'IS TORCH AT ORIGIN ?', EOT)) ;
 211   3                IF CONSOLE$INPUT = 'Y'
                        THEN
 212   3                    DO ;
 213   4                        CALL STORE$NUMBER (.X$ORDINATE, .ZERO) ;
 214   4                        CALL STORE$NUMBER (.Y$ORDINATE, .ZERO) ;
 215   4                        CALL STORE$NUMBER (.Z$ORDINATE, .ZERO) ;
 216   4                        CALL STORE$NUMBER (.R$ORDINATES(0).ORDINATE, .ZERO) ;
 217   4                        CALL STORE$NUMBER (.R$ORDINATES(1).ORDINATE, .ZERO) ;
 218   4                        CALL STORE$NUMBER (.R$ORDINATES(2).ORDINATE, .ZERO) ;
 219   4                        CALL STORE$NUMBER (.R$ORDINATES(3).ORDINATE, .ZERO) ;
 220   4                        ORIGIN$SET = TRUE ;
 221   4                        CALL DISPLAY$X ;
 222   4                        CALL DISPLAY$Y ;
 223   4                        CALL DISPLAY$Z ;
 224   4                        CALL DISPLAY$R(0) ;
 225   4                        CALL DISPLAY$R(1) ;
 226   4                        CALL DISPLAY$R(2) ;
 227   4                        CALL DISPLAY$R(3) ;
 228   4                    END ;
 229   3            END ;
 230   2            CALL CLEAR$PROMPT$FIELD ;
 231   2        END SET$ORIGIN ;
 232   1    END SET$ORIGIN$PROCEDURE ;



MODULE INFORMATION:

    CODE AREA SIZE     = 00C5H    197D
    VARIABLE AREA SIZE = 0001H      1D
    MAXIMUM STACK SIZE = 0002H      2D
    605 LINES READ
    0 PROGRAM ERROR(S)

END OF PL/M-80 COMPILATION
```

```
ISIS-II PL/M-80 V3.1 COMPILATION OF MODULE POWERSOURCECONTROLPROCEDURES
OBJECT MODULE PLACED IN :F1:WELD22.OBJ
COMPILER INVOKED BY:  PLM80 :F1:WELD22.PLM DEBUG WORKFILES(:F1:,:F1:) PRINT(:F1:WELD22.LST)



            $TITLE ('POWER$SOURCE$CONTROL$PROCEDURES    VER 3.00    DATE 08/07/81')
  1         POWER$SOURCE$CONTROL$PROCEDURES: DO ;

            /*PLM80 PROCEDURES FOR CONTROLING THE POWER SOURCE FUNCTIONS.
            */
            $INCLUDE (:F1:WELD.EQU)
        =   $NOLIST
        =


  3   1         DECLARE (START$ENABLE,
                         STOP$ENABLE,
                         GAS$ENABLE,
                         WIRE$ENABLE      ) BYTE PUBLIC ;


  4   1         SET$POWER$SOURCE$CONTROL: PROCEDURE PUBLIC ;
  5   2             DECLARE STATUS BYTE ;
  6   2             STATUS = 0FFH ;
  7   2             IF START$ENABLE
                    THEN
  8   2                 STATUS = STATUS AND BIT$0 ;
  9   2             IF STOP$ENABLE
                    THEN
 10   2                 STATUS = STATUS AND BIT$1 ;
 11   2             IF GAS$ENABLE
                    THEN
 12   2                 STATUS = STATUS AND BIT$2 ;
 13   2             IF WIRE$ENABLE
                    THEN
 14   2                 STATUS = STATUS AND BIT$3 ;
 15   2             OUTPUT (POWER$SOURCE$CONTROL) = NOT (STATUS) ;
 16   2         END SET$POWER$SOURCE$CONTROL ;

 17   1         RESET$POWER$SOURCE$CONTROL: PROCEDURE PUBLIC ;
 18   2             START$ENABLE,
                    STOP$ENABLE,
                    GAS$ENABLE,
                    WIRE$ENABLE     = FALSE ;
 19   2             CALL SET$POWER$SOURCE$CONTROL ;
 20   2         END RESET$POWER$SOURCE$CONTROL ;

 21   1     END POWER$SOURCE$CONTROL$PROCEDURES ;



MODULE INFORMATION:

    CODE AREA SIZE     = 005AH     90D
    VARIABLE AREA SIZE = 0005H      5D
    MAXIMUM STACK SIZE = 0002H      2D
    252 LINES READ
```

0 PROGRAM ERROR(S)

```
ISIS-II PL/M-80 V3.1 COMPILATION OF MODULE SETLIMITSPROCEDURE
OBJECT MODULE PLACED IN :F1:WELD5.OBJ
COMPILER INVOKED BY:  PLM80 :F1:WELD5.PLM DEBUG WORKFILES(:F1:,:F1:) PRINT(:F1:WELD5.LST)


           $TITLE('SET$LIMITS$PROCEDURE  VER 2.00  DATE 22/03/81')
  1        SET$LIMITS$PROCEDURE: DO ;

           /* A PLM80 ROUTINE TO ALLOW THE USER TO SET THE SYSTEM MOVEMENT LIMITS :-
                                 XUPPER
                                 XLOWER
                                 YUPPER
                                 YLOWER
                                 ZUPPER
                                 ZLOWER
              THE USER FIRSTLY ENTERS THE NAME OF THE LIMIT TO BE SET AND THEN MOVES THE
              TORCH TO THE POSITION CORRESPONDING TO THAT LIMIT.
             THE PROCESS OF ENTERING THE LIMIT NAME AND MOVING THE TORCH CAN BE REPEATED
              UNTIL ALL THE REQUIRED CHANGES HAVE BEEN MADE.

              THE ROUTINE IS EXITED BY PRESSING THE 'ESCAPE' INSTEAD OF ENTERING A NAME.
           */
```

```
            $EJECT

            /* EXTERNALS */
            $INCLUDE (:F1:WELD.EQU)
        =   $NOLIST
        =
            $INCLUDE (:F1:WELD1.EDS)
        =   $NOLIST
            $INCLUDE (:F1:MATH.EDS)
        =   $NOLIST
            $INCLUDE (:F1:WELD2.EDS)
        =   $NOLIST

            /* PUBLIC DATA */
187   1         DECLARE (X$UPPER$LIMIT,
                       X$LOWER$LIMIT,
                       Y$UPPER$LIMIT,
                       Y$LOWER$LIMIT,
                       Z$UPPER$LIMIT,
                       Z$LOWER$LIMIT  )          (4) BYTE PUBLIC ;

            /* LOCAL DATA */
188   1         DECLARE BUFFER BASED BUFFER$POINTER BYTE ;
189   1         DECLARE (WORD$START, WORD$END ) ADDRESS ;
```

```
                $EJECT
190   1         SET$X$UPPER$LIMIT: PROCEDURE PUBLIC ;
191   2             CALL POSITION$CURSOR (PROMPT$FIELD) ;
192   2             CALL WRITE$MESSAGE (.(DELETE$LINE,
                                        'MOVE TORCH TO X UPPER LIMIT', EOT)) ;
193   2             X$UPPER$LIMIT$SET = FALSE ;
194   2             CALL POSITION ;
195   2             CALL STORE$NUMBER (.X$UPPER$LIMIT, .X$ORDINATE) ;
196   2             X$UPPER$LIMIT$SET = TRUE ;
197   2         END SET$X$UPPER$LIMIT ;

198   1         SET$X$LOWER$LIMIT: PROCEDURE PUBLIC ;
199   2             CALL POSITION$CURSOR (PROMPT$FIELD) ;
200   2             CALL WRITE$MESSAGE (.(DELETE$LINE,
                                        'MOVE TORCH TO X LOWER LIMIT', EOT)) ;
201   2             X$LOWER$LIMIT$SET = FALSE ;
202   2             CALL POSITION ;
203   2             CALL STORE$NUMBER (.X$LOWER$LIMIT, .X$ORDINATE ) ;
204   2             X$LOWER$LIMIT$SET = TRUE ;
205   2         END SET$X$LOWER$LIMIT ;

206   1         SET$Y$UPPER$LIMIT: PROCEDURE PUBLIC ;
207   2             CALL POSITION$CURSOR (PROMPT$FIELD) ;
208   2             CALL WRITE$MESSAGE (.(DELETE$LINE,
                                        'MOVE TORCH TO Y UPPER LIMIT', EOT)) ;
209   2             Y$UPPER$LIMIT$SET = FALSE ;
210   2             CALL POSITION ;
211   2             CALL STORE$NUMBER (.Y$UPPER$LIMIT, .Y$ORDINATE) ;
212   2             Y$UPPER$LIMIT$SET = TRUE ;
213   2         END SET$Y$UPPER$LIMIT ;

214   1         SET$Y$LOWER$LIMIT: PROCEDURE PUBLIC ;
215   2             CALL POSITION$CURSOR (PROMPT$FIELD) ;
216   2             CALL WRITE$MESSAGE (.(DELETE$LINE,
                                        'MOVE TORCH TO Y LOWER LIMIT', EOT)) ;
217   2             Y$LOWER$LIMIT$SET = FALSE ;
218   2             CALL POSITION ;
219   2             CALL STORE$NUMBER (.Y$LOWER$LIMIT, .Y$ORDINATE) ;
220   2             Y$LOWER$LIMIT$SET = TRUE ;
221   2         END SET$Y$LOWER$LIMIT ;

222   1         SET$Z$UPPER$LIMIT: PROCEDURE PUBLIC ;
223   2             CALL POSITION$CURSOR (PROMPT$FIELD) ;
224   2             CALL WRITE$MESSAGE (.(DELETE$LINE,
                                        'MOVE TORCH TO Z UPPER LIMIT', EOT)) ;
225   2             Z$UPPER$LIMIT$SET = FALSE ;
226   2             CALL POSITION ;
227   2             CALL STORE$NUMBER (.Z$UPPER$LIMIT, .Z$ORDINATE) ;
228   2             Z$UPPER$LIMIT$SET = TRUE ;
229   2         END SET$Z$UPPER$LIMIT ;

230   1         SET$Z$LOWER$LIMIT: PROCEDURE PUBLIC ;
231   2             CALL POSITION$CURSOR (PROMPT$FIELD) ;
232   2             CALL WRITE$MESSAGE (.(DELETE$LINE,
                                        'MOVE TORCH TO Z LOWER LIMIT', EOT)) ;
233   2             Z$LOWER$LIMIT$SET = FALSE ;
234   2             CALL POSITION ;
```

```
235   2             CALL STORE$NUMBER (.Z$LOWER$LIMIT, .Z$ORDINATE) ;
236   2             Z$LOWER$LIMIT$SET = TRUE ;
237   2         END SET$Z$LOWER$LIMIT ;
```

```
                $EJECT
 238   1          SET$LIMITS: PROCEDURE PUBLIC ;
 239   2              CALL POSITION$CURSOR (PROG$FIELD) ;
 240   2              CALL WRITE$MESSAGE (.(DELETE$LINE,
                                       'SET LIMITS', EOT)) ;
 241   2          A20: CALL POSITION$CURSOR (PROMPT$FIELD) ;
 242   2              CALL WRITE$MESSAGE (.(DELETE$LINE,
                                       'WHICH LIMIT ?', EOT)) ;
 243   2              CALL READ$LINE ;
 244   2              BUFFER$POINTER = .EDIT$BUFFER ;
 245   2              WORD$START = NEXT$CHARACTER ;
 246   2              IF BUFFER = CARRIGE$RETURN
                      THEN
 247   2                  GOTO A20 ;
 248   2              IF BUFFER = ESC
                      THEN
 249   2                  DO ;
 250   3                      CALL POSITION$CURSOR (LIMITS$WARNING$FIELD) ;
 251   3                      IF NOT (X$UPPER$LIMIT$SET AND X$LOWER$LIMIT$SET
                                AND  Y$UPPER$LIMIT$SET AND Y$LOWER$LIMIT$SET
                                AND  Z$UPPER$LIMIT$SET AND Z$LOWER$LIMIT$SET )
                              THEN
 252   3                          CALL WRITE$MESSAGE (.LIMITS$WARNING$MESSAGE) ;
                              ELSE
 253   3                          CALL CONSOLE$OUTPUT (DELETE$LINE) ;
 254   3                      CALL CLEAR$PROMPT$FIELD ;
 255   3                      RETURN ;
 256   3                  END ;
 257   2              WORD$END = NEXT$SPACE ;
 258   2              IF COMPARE$WORDS (WORD$START, WORD$END, .('XUPPER'), 6)
                      THEN
 259   2                  CALL SET$X$UPPER$LIMIT ;
 260   2              IF COMPARE$WORDS (WORD$START, WORD$END, .('XLOWER'), 6)
                      THEN
 261   2                  CALL SET$X$LOWER$LIMIT ;
 262   2              IF COMPARE$WORDS (WORD$START, WORD$END, .('YUPPER'), 6)
                      THEN
 263   2                  CALL SET$Y$UPPER$LIMIT ;
 264   2              IF COMPARE$WORDS (WORD$START, WORD$END, .('YLOWER'), 6)
                      THEN
 265   2                  CALL SET$Y$LOWER$LIMIT ;
 266   2              IF COMPARE$WORDS (WORD$START, WORD$END, .('ZUPPER'), 6)
                      THEN
 267   2                  CALL SET$Z$UPPER$LIMIT ;
 268   2              IF COMPARE$WORDS (WORD$START, WORD$END, .('ZLOWER'), 6)
                      THEN
 269   2                  CALL SET$Z$LOWER$LIMIT ;
 270   2              GOTO A20 ;
 271   2          END SET$LIMITS ;

 272   1      END SET$LIMITS$PROCEDURE ;



MODULE INFORMATION:

    CODE AREA SIZE     = 02D1H    721D
```

```
    VARIABLE AREA SIZE = 001CH     28D
    MAXIMUM STACK SIZE = 0006H      6D
    670 LINES READ
    0 PROGRAM ERROR(S)

END OF PL/M-80 COMPILATION
```

```
ISIS-II PL/M-80 V3.1 COMPILATION OF MODULE TAPEIO
OBJECT MODULE PLACED IN :F1:WELD25.OBJ
COMPILER INVOKED BY:  PLM80 :F1:WELD25.PLM DEBUG WORKFILES(:F1:,:F1:) PRINT(:F1:WELD25.LST)



                $TITLE ('TAPE$IO  VER 3.00  DATE 16/04/82')
  1             TAPE$IO: DO ;

                /* SYSTEM EQUATES */
                $INCLUDE (:F1:WELD.EQU)
            =   $NOLIST
            =

                /* EXTERNAL PROCEDURES */
  3   1             CONSOLE$OUTPUT: PROCEDURE (CHARACTER) EXTERNAL ;
  4   2                 DECLARE CHARACTER BYTE ;
  5   2             END CONSOLE$OUTPUT ;

  6   1             CONSOLE$INPUT: PROCEDURE BYTE EXTERNAL ;
  7   2             END CONSOLE$INPUT ;

  8   1             POSITION$CURSOR: PROCEDURE (COLUMN$NUMBER, LINE$NUMBER) EXTERNAL ;
  9   2                 DECLARE (COLUMN$NUMBER, LINE$NUMBER) BYTE ;
 10   2             END POSITION$CURSOR ;

 11   1             WRITE$MESSAGE: PROCEDURE (STRING$ADDRESS) EXTERNAL ;
 12   2                 DECLARE STRING$ADDRESS ADDRESS ;
 13   2             END WRITE$MESSAGE ;

 14   1             CLEAR$WARNING$FIELD: PROCEDURE EXTERNAL ;
 15   2             END CLEAR$WARNING$FIELD ;

 16   1             SET$INTERRUPT$MASK: PROCEDURE EXTERNAL ;
 17   2             END SET$INTERRUPT$MASK ;

 18   1             DISPLAY$PATH$NAME: PROCEDURE EXTERNAL ;
 19   2             END DISPLAY$PATH$NAME ;

                /* EXTERNAL DATA */
 20   1             DECLARE UART$STATUS           BYTE EXTERNAL ;
 21   1             DECLARE UART$DATA             BYTE EXTERNAL ;
 22   1             DECLARE DELAY$FLAG            BYTE EXTERNAL ;
 23   1             DECLARE DELAY$COUNT        ADDRESS EXTERNAL ;
 24   1             DECLARE RST$5$5$MASK          BYTE EXTERNAL ;
 25   1             DECLARE LIMITS$WARNING$MESSAGE BYTE EXTERNAL ;
 26   1             DECLARE PATH$SET BYTE EXTERNAL ;
 27   1             DECLARE (X$UPPER$LIMIT$SET, X$LOWER$LIMIT$SET,
                              Y$UPPER$LIMIT$SET, Y$LOWER$LIMIT$SET,
                              Z$UPPER$LIMIT$SET, Z$LOWER$LIMIT$SET ) BYTE EXTERNAL ;
 28   1             DECLARE (X$UPPER$LIMIT, X$LOWER$LIMIT,
                              Y$UPPER$LIMIT, Y$LOWER$LIMIT,
                              Z$UPPER$LIMIT, Z$LOWER$LIMIT ) (4) BYTE EXTERNAL ;
 29   1             DECLARE POINT$COUNT ADDRESS EXTERNAL ;
 30   1             DECLARE DISPLACEMENT$TABLE (1000) STRUCTURE (IDENTIFIER       BYTE,
                                                                 DISPLACEMENT$1$SINGLE (2) BYTE,
                                                                 DISPLACEMENT$2$SINGLE (2) BYTE,
```

```
                                                    DISPLACEMENT$3$SINGLE (2) BYTE) EXTERNAL ;
 31  1          DECLARE (REST$POSITION$X,
                         REST$POSITION$Y,
                         REST$POSITION$Z,
                         REST$POSITION$R$0,
                         REST$POSITION$R$1,
                         REST$POSITION$R$2,
                         REST$POSITION$R$3 ) (4) BYTE EXTERNAL ;
 32  1          DECLARE SEQUENCE$TABLE      (60) BYTE EXTERNAL ;
 33  1          DECLARE PARAMETER$GROUP$TABLE (256) STRUCTURE (PULSE$CURRENT$FLOAT (4) BYTE,
                                                               BACKGROUND$CURRENT$FLOAT (4) BYTE,
                                                               PULSE$ON$TIME$FLOAT (4) BYTE,
                                                               PULSE$OFF$TIME$FLOAT (4) BYTE,
                                                               WELDING$DIVISOR$FLOAT (4) BYTE,
                                                               PRE$TRAVERSE$TIME$FLOAT (4) BYTE,
                                                               FALL$TIME$FLOAT (4) BYTE,
                                                               R$0$DIVISOR$SINGLE (2) BYTE,
                                                               R$1$DIVISOR$SINGLE (2) BYTE,
                                                               R$2$DIVISOR$SINGLE (2) BYTE,
                                                               R$3$DIVISOR$SINGLE (2) BYTE,
                                                               TOUCH$START            BYTE,
                                                               WIRE$FEED              BYTE) EXTERNAL ;
 34  1          DECLARE PARAMETERS$SET$FLAG (256) BYTE EXTERNAL ;
 35  1          DECLARE PATH$NAME           (16) BYTE EXTERNAL ;
 36  1          DECLARE PATH$NAME$SET            BYTE EXTERNAL ;
 37  1          DECLARE SEAM$COUNT               BYTE EXTERNAL ;
```

```
            $EJECT
            /* LOCAL DATA */
 38   1        DECLARE (HEX$NIBBLE,
                        BYTE$VALUE,
                        TIME$COUNT,
                        STATUS,
                        EOF$BYTE     ) BYTE ;
 39   1        DECLARE (GROUP$COUNT,
                        I            ) ADDRESS ;
 40   1        DECLARE HEX$DATA$ERROR                LITERALLY '01H',
                      TIME$OUT$ERROR                LITERALLY '02H',
                      LENGTH$ERROR                  LITERALLY '03H',
                      ID$ERROR                      LITERALLY '04H',
                      CHECK$SUM$ERROR               LITERALLY '05H',
                      PATH$READ$ERROR               LITERALLY '06H',
                      SEQUENCE$READ$ERROR           LITERALLY '07H',
                      PARAMETER$READ$ERROR          LITERALLY '08H',
                      LIMIT$READ$ERROR              LITERALLY '09H',
                      EOF$ERROR                     LITERALLY '0AH',
                      PATH$STATUS$ID                LITERALLY '10H',
                      PATH$NAME$ID                  LITERALLY '11H',
                      POINT$COUNT$ID                LITERALLY '12H',
                      DISPLACEMENT$ID               LITERALLY '13H',
                      REST$POSITION$ID              LITERALLY '14H',
                      SEQUENCE$ID                   LITERALLY '15H',
                      GROUP$COUNT$ID                LITERALLY '16H',
                      PARAMETER$ID                  LITERALLY '17H',
                      LIMIT$STATUS$ID               LITERALLY '18H',
                      LIMIT$ID                      LITERALLY '19H',
                      EOF$ID                        LITERALLY '1AH',
                      SEAM$COUNT$ID                 LITERALLY '1BH' ;
```

```
                $EJECT

 41  1          TX$EMPTY$DELAY: PROCEDURE ;

 42  2              DELAY$FLAG = FALSE ;
 43  2              DELAY$COUNT = 10 ;
 44  2              RST$5$5$MASK = FALSE ;
 45  2              CALL SET$INTERRUPT$MASK ;
 46  2              DELAY$FLAG = TRUE ;
 47  2              ENABLE ;
 48  2              DO WHILE DELAY$FLAG ;           /* DELAY FOR 70 mS */
 49  3              END ;
 50  2          END TX$EMPTY$DELAY ;

 51  1          CONVERT$BINARY$TO$ASCII: PROCEDURE (NIBBLE) BYTE ;
 52  2              DECLARE NIBBLE BYTE ;

 53  2              IF NIBBLE < 0AH
                    THEN
 54  2                  RETURN (NIBBLE + 30H) ;
                    ELSE
 55  2                  RETURN (NIBBLE + 37H) ;
 56  2          END CONVERT$BINARY$TO$ASCII ;

 57  1          PUNCH$BYTE$HEX: PROCEDURE (DATA$BYTE) ;
 58  2              DECLARE DATA$BYTE BYTE ;

 59  2              CALL CONSOLE$OUTPUT ( CONVERT$BINARY$TO$ASCII(ROL(DATA$BYTE AND 11110000B,4)) ) ;
 60  2              CALL CONSOLE$OUTPUT ( CONVERT$BINARY$TO$ASCII(DATABYTE AND 00001111B) ) ;
 61  2          END PUNCH$BYTE$HEX ;

 62  1          PUNCH$RECORD: PROCEDURE (COUNT, DATA$ADDRESS, IDENTIFIER) ;
 63  2              DECLARE (I, COUNT, IDENTIFIER, SUM) BYTE ;
 64  2              DECLARE DATA$ADDRESS ADDRESS ;
 65  2              DECLARE DATA$BYTE BASED DATA$ADDRESS (256) BYTE ;

 66  2              CALL CONSOLE$OUTPUT (CARRIGE$RETURN) ;
 67  2              CALL CONSOLE$OUTPUT (LINE$FEED) ;
 68  2              CALL CONSOLE$OUTPUT (':') ;
 69  2              CALL PUNCH$BYTE$HEX (COUNT) ;
 70  2              SUM = COUNT ;
 71  2              CALL PUNCH$BYTE$HEX (IDENTIFIER) ;
 72  2              SUM = SUM + IDENTIFIER ;
 73  2              DO I = 1 TO COUNT ;
 74  3                  CALL PUNCH$BYTE$HEX (DATA$BYTE(I-1)) ;
 75  3                  SUM = SUM + DATA$BYTE(I-1) ;
 76  3              END ;
 77  2              CALL PUNCH$BYTE$HEX (NOT(SUM-1)) ;
 78  2          END PUNCH$RECORD ;
```

```
                $EJECT
 79   1          CONVERT$ASCII$TO$HEX: PROCEDURE (ASCII$BYTE) BYTE ;
 80   2              DECLARE ASCII$BYTE BYTE ;

 81   2              IF (ASCII$BYTE > 2FH) AND (ASCII$BYTE < 3AH)
                     THEN
 82   2                  HEX$NIBBLE = ASCII$BYTE - 30H ;
                     ELSE
 83   2                  DO ;
 84   3                      IF (ASCII$BYTE > 40H) AND (ASCII$BYTE < 47H)
                             THEN
 85   3                          HEX$NIBBLE = ASCII$BYTE - 37H ;
                             ELSE
 86   3                          RETURN HEX$DATA$ERROR ;
 87   3                  END ;
 88   2              RETURN 00 ;
 89   2          END CONVERT$ASCII$TO$HEX ;

 90   1          READ$BYTE: PROCEDURE BYTE ;

 91   2              IF CONVERT$ASCII$TO$HEX (CONSOLE$INPUT) <> 0
                     THEN
 92   2                  RETURN HEX$DATA$ERROR ;
 93   2              BYTE$VALUE = ROL (HEX$NIBBLE, 4) ;
 94   2              IF CONVERT$ASCII$TO$HEX (CONSOLE$INPUT) <> 0
                     THEN
 95   2                  RETURN HEX$DATA$ERROR ;
 96   2              BYTE$VALUE = BYTE$VALUE OR HEX$NIBBLE ;
 97   2              RETURN 00 ;
 98   2          END READ$BYTE ;
```

```
$EJECT

 99  1    READ$RECORD: PROCEDURE (BUFFER$POINTER, LENGTH, ID) BYTE ;
100  2        DECLARE BUFFER$POINTER ADDRESS ;
101  2        DECLARE BUFFER BASED BUFFER$POINTER (256) BYTE ;
102  2        DECLARE (LENGTH,
                      ID,
                      I,
                      COUNT,
                      SUM    ) BYTE ;

103  2        TIME$COUNT = 0FFFFH ;
104  2        DO WHILE CONSOLE$INPUT <> ':' ;
105  3            TIME$COUNT = TIME$COUNT - 1 ;
106  3            IF TIME$COUNT = 0
                  THEN
107  3                RETURN TIME$OUT$ERROR ;
108  3            END ;
109  2            IF (STATUS:=READ$BYTE) <> 0
                  THEN
110  2                RETURN STATUS ;
111  2            COUNT = BYTE$VALUE ;
112  2            IF COUNT <> LENGTH
                  THEN
113  2                RETURN LENGTH$ERROR ;
114  2            SUM = COUNT ;
115  2            IF (STATUS:=READ$BYTE) <> 0
                  THEN
116  2                RETURN STATUS ;
117  2            IF BYTE$VALUE <> ID
                  THEN
118  2                RETURN ID$ERROR ;
119  2            SUM = SUM + BYTE$VALUE ;
120  2            DO I = 1 TO COUNT ;
121  3                IF (STATUS:=READ$BYTE) <> 0
                      THEN
122  3                    RETURN STATUS ;
123  3                BUFFER(I-1) = BYTE$VALUE ;
124  3                SUM = SUM + BYTE$VALUE ;
125  3            END ;
126  2            IF (STATUS:=READ$BYTE) <> 0
                  THEN
127  2                RETURN STATUS ;
128  2            SUM = SUM + BYTE$VALUE ;
129  2            IF SUM <> 0
                  THEN
130  2                RETURN CHECK$SUM$ERROR ;
131  2            RETURN 00 ;
132  2        END READ$RECORD ;
```

```
                $EJECT
 133   1          TAPE$READ: PROCEDURE BYTE ;

 134   2              DO I = 0 TO 15 ;
 135   3                  PATH$NAME(I) = ' ' ;
 136   3              END ;
 137   2              PATH$NAME$SET = TRUE ;
 138   2              IF READ$RECORD (.PATH$SET, 1, PATH$STATUS$ID) <> 0
                       THEN
 139   2                  DO ;
 140   3                      PATH$SET = FALSE ;
 141   3                      RETURN PATH$READ$ERROR ;
 142   3                  END ;
 143   2              IF PATH$SET
                       THEN
 144   2                  DO ;
 145   3                      PATH$SET = FALSE ;
 146   3                      IF READ$RECORD (.PATH$NAME, 16, PATH$NAME$ID) <> 0
                               THEN
 147   3                          RETURN PATH$READ$ERROR ;
 148   3                      IF READ$RECORD (.POINT$COUNT, 2, POINT$COUNT$ID) <> 0
                               THEN
 149   3                          RETURN PATH$READ$ERROR ;
 150   3                      DO I = 0 TO POINT$COUNT-1 ;
 151   4                          IF READ$RECORD (.DISPLACEMENT$TABLE(I), 7, DISPLACEMENT$ID) <> 0
                                   THEN
 152   4                              RETURN PATH$READ$ERROR ;
 153   4                      END ;
 154   3                      IF READ$RECORD (.REST$POSITION$X, 28, REST$POSITION$ID) <> 0
                               THEN
 155   3                          RETURN PATH$READ$ERROR ;
 156   3                      IF READ$RECORD (.SEAM$COUNT, 1, SEAM$COUNT$ID) <> 0
                               THEN
 157   3                          RETURN PATH$READ$ERROR ;
 158   3                      PATH$SET = TRUE ;
 159   3                  END ;
 160   2              IF READ$RECORD (.SEQUENCE$TABLE, 60, SEQUENCE$ID) <> 0
                       THEN
 161   2                  RETURN SEQUENCE$READ$ERROR ;
 162   2              IF READ$RECORD (.GROUP$COUNT, 2, GROUP$COUNT$ID) <> 0
                       THEN
 163   2                  RETURN PARAMETER$READ$ERROR ;
 164   2              DO I = 1 TO GROUP$COUNT ;
 165   3                  PARAMETERS$SET$FLAG(I-1) = FALSE ;
 166   3                  IF READ$RECORD (.PARAMETER$GROUP$TABLE(I-1), 38, PARAMETER$ID) <> 0
                           THEN
 167   3                      RETURN PARAMETER$READ$ERROR ;
 168   3                  PARAMETERS$SET$FLAG(I-1) = TRUE ;
 169   3              END ;
 170   2              IF READ$RECORD (.XUPPER$LIMIT$SET, 1, LIMIT$STATUS$ID) <> 0
                       THEN
 171   2                  DO ;
 172   3                      X$UPPER$LIMIT$SET = FALSE ;
 173   3                      RETURN LIMIT$READ$ERROR ;
 174   3                  END ;
 175   2              IF X$UPPER$LIMIT$SET
                       THEN
```

```
176  2              DO ;
177  3                  X$UPPER$LIMIT$SET = FALSE ;
178  3                  IF READ$RECORD (.X$UPPER$LIMIT, 4, LIMIT$ID) <> 0
                        THEN
179  3                      RETURN LIMIT$READ$ERROR ;
180  3                  X$UPPER$LIMIT$SET = TRUE ;
181  3              END ;
182  2            IF READ$RECORD (.X$LOWER$LIMIT$SET, 1, LIMIT$STATUS$ID) <> 0
                  THEN
183  2              DO ;
184  3                  X$LOWER$LIMIT$SET = FALSE ;
185  3                 RETURN LIMIT$READ$ERROR ;
186  3              END ;
187  2            IF X$LOWER$LIMIT$SET
                  THEN
188  2              DO ;
189  3                  X$LOWER$LIMIT$SET = FALSE ;
190  3                  IF READ$RECORD (.X$LOWER$LIMIT, 4, LIMIT$ID) <> 0
                        THEN
191  3                      RETURN LIMIT$READ$ERROR ;
192  3                  X$LOWER$LIMIT$SET = TRUE ;
193  3              END ;
194  2            IF READ$RECORD (.Y$UPPER$LIMIT$SET, 1, LIMIT$STATUS$ID) <> 0
                  THEN
195  2              DO ;
196  3                  Y$UPPER$LIMIT$SET = FALSE ;
197  3                  RETURN LIMIT$READ$ERROR ;
198  3              END ;
199  2            IF Y$UPPER$LIMIT$SET
                  THEN
200  2              DO ;
201  3                  Y$UPPER$LIMIT$SET = FALSE ;
202  3                  IF READ$RECORD (.Y$UPPER$LIMIT, 4, LIMIT$ID) <> 0
                        THEN
203  3                      RETURN LIMIT$READ$ERROR ;
204  3                  Y$UPPER$LIMIT$SET = TRUE ;
205  3              END ;
206  2            IF READ$RECORD (.Y$LOWER$LIMIT$SET, 1, LIMIT$STATUS$ID) <> 0
                  THEN
207  2              DO ;
208  3                  Y$LOWER$LIMIT$SET = FALSE ;
209  3                  RETURN LIMIT$READ$ERROR ;
210  3              END ;
211  2            IF Y$LOWER$LIMIT$SET
                  THEN
212  2              DO ;
213  3                  Y$LOWER$LIMIT$SET = FALSE ;
214  3                  IF READ$RECORD (.Y$LOWER$LIMIT, 4, LIMIT$ID) <> 0
                        THEN
215  3                     RETURN LIMIT$READ$ERROR ;
216  3                  Y$LOWER$LIMIT$SET = TRUE ;
217  3              END ;
218  2            IF READ$RECORD (.Z$UPPER$LIMIT$SET, 1, LIMIT$STATUS$ID) <> 0
                  THEN
219  2              DO ;
220  3                  Z$UPPER$LIMIT$SET = FALSE ;
221  3                  RETURN LIMIT$READ$ERROR ;
```

```
222  3              END ;
223  2           IF Z$UPPER$LIMIT$SET
                 THEN
224  2              DO ;
225  3                 Z$UPPER$LIMIT$SET = FALSE ;
226  3                 IF READ$RECORD (.Z$UPPER$LIMIT, 4, LIMIT$ID) <> 0
                       THEN
227  3                    RETURN LIMIT$READ$ERROR ;
228  3                 Z$UPPER$LIMIT$SET = TRUE ;
229  3              END ;
230  2           IF READ$RECORD (.Z$LOWER$LIMIT$SET, 1, LIMIT$STATUS$ID) <> 0
                 THEN
231  2              DO ;
232  3                 Z$LOWER$LIMIT$SET = FALSE ;
233  3                 RETURN LIMIT$READ$ERROR ;
234  3              END ;
235  2           IF Z$LOWER$LIMIT$SET
                 THEN
236  2              DO ;
237  3                 Z$LOWER$LIMIT$SET = FALSE ;
238  3                 IF READ$RECORD (.Z$LOWER$LIMIT, 4, LIMIT$ID) <> 0
                       THEN
239  3                    RETURN LIMIT$READ$ERROR ;
240  3                 Z$LOWER$LIMIT$SET = TRUE ;
241  3              END ;
242  2           IF READ$RECORD (.EOF$BYTE, 1, EOF$ID) <> 0
                 THEN
243  2              RETURN EOF$ERROR ;
244  2           IF EOF$BYTE <> 0
                 THEN
245  2              RETURN EOF$ERROR ;
246  2           RETURN 00 ;
247  2        END TAPE$READ ;
```

```
                $EJECT
 248   1          DUMP: PROCEDURE PUBLIC ;

 249   2              DO I = 0 TO 255 ;
 250   3                  CALL TIME (255) ;
 251   3              END ;                          /* DELAY WHILST TAPE UNIT IS SWITCHED IN */
 252   2              UART$CONTROL = SETTING (DIVIDE$64 AND
                                               ALT$0     AND
                                               RTS$LO    AND
                                               DISABLE$INT   ) ;
 253   2              IF PATH$SET
                       THEN
 254   2                  DO ;
 255   3                      CALL PUNCH$RECORD (1, .(TRUE), PATH$STATUS$ID) ;
 256   3                      CALL PUNCH$RECORD (16, .PATH$NAME, PATH$NAME$ID) ;
 257   3                      CALL PUNCH$RECORD (2, .POINT$COUNT, POINT$COUNT$ID) ;
 258   3                      DO I = 0 TO POINT$COUNT-1 ;
 259   4                          CALL PUNCH$RECORD (7, .DISPLACEMENT$TABLE(I), DISPLACEMENT$ID) ;
 260   4                      END ;
 261   3                      CALL PUNCH$RECORD (28, .REST$POSITION$X, REST$POSITION$ID) ;
 262   3                      CALL PUNCH$RECORD (1,  .SEAM$COUNT,      SEAM$COUNT$ID) ;
 263   3                  END ;
                       ELSE
 264   2                  CALL PUNCH$RECORD (1, .(FALSE), PATH$STATUS$ID) ;
 265   2              CALL PUNCH$RECORD (60, .SEQUENCE$TABLE, SEQUENCE$ID) ;
 266   2              GROUP$COUNT = 0 ;
 267   2              DO WHILE (PARAMETERS$SET$FLAG(GROUP$COUNT) AND (GROUP$COUNT < 256)) ;
 268   3                  GROUP$COUNT = GROUP$COUNT + 1 ;
 269   3              END ;
 270   2              CALL PUNCH$RECORD (2, .GROUP$COUNT, GROUP$COUNT$ID) ;
 271   2              DO I = 1 TO GROUP$COUNT ;
 272   3                  CALL PUNCH$RECORD (38, .PARAMETER$GROUP$TABLE(I-1), PARAMETER$ID) ;
 273   3              END ;
 274   2              IF X$UPPER$LIMIT$SET
                       THEN
 275   2                  DO ;
 276   3                      CALL PUNCH$RECORD (1, .(TRUE), LIMIT$STATUS$ID) ;
 277   3                      CALL PUNCH$RECORD (4, .X$UPPER$LIMIT, LIMIT$ID) ;
 278   3                  END ;
                       ELSE
 279   2                  CALL PUNCH$RECORD (1, .(FALSE), LIMIT$STATUS$ID) ;
 280   2              IF X$LOWER$LIMIT$SET
                       THEN
 281   2                  DO ;
 282   3                      CALL PUNCH$RECORD (1, .(TRUE), LIMIT$STATUS$ID) ;
 283   3                      CALL PUNCH$RECORD (4, .X$LOWER$LIMIT, LIMIT$ID) ;
 284   3                  END ;
                       ELSE
 285   2                  CALL PUNCH$RECORD (1, .(FALSE), LIMIT$STATUS$ID) ;
 286   2              IF Y$UPPER$LIMIT$SET
                       THEN
 287   2                  DO ;
 288   3                      CALL PUNCH$RECORD (1, .(TRUE), LIMIT$STATUS$ID) ;
 289   3                      CALL PUNCH$RECORD (4, .Y$UPPER$LIMIT, LIMIT$ID) ;
 290   3                  END ;
                       ELSE
 291   2                  CALL PUNCH$RECORD (1, .(FALSE), LIMIT$STATUS$ID) ;
```

```
292  2          IF Y$LOWER$LIMIT$SET
                THEN
293  2            DO ;
294  3              CALL PUNCH$RECORD (1, .(TRUE), LIMIT$STATUS$ID) ;
295  3              CALL PUNCH$RECORD (4, .Y$LOWER$LIMIT, LIMIT$ID) ;
296  3            END ;
                ELSE
297  2            CALL PUNCH$RECORD (1, .(FALSE), LIMIT$STATUS$ID) ;

298  2          IF Z$UPPER$LIMIT$SET
                THEN
299  2            DO ;
300  3              CALL PUNCH$RECORD (1, .(TRUE), LIMIT$STATUS$ID) ;
301  3              CALL PUNCH$RECORD (4, .Z$UPPER$LIMIT, LIMIT$ID) ;
302  3            END ;
                ELSE
303  2            CALL PUNCH$RECORD (1, .(FALSE), LIMIT$STATUS$ID) ;
304  2          IF Z$LOWER$LIMIT$SET
                THEN
305  2            DO ;
306  3              CALL PUNCH$RECORD (1, .(TRUE), LIMIT$STATUS$ID) ;
307  3              CALL PUNCH$RECORD (4, .Z$LOWER$LIMIT, LIMIT$ID) ;
308  3            END ;
                ELSE
309  2            CALL PUNCH$RECORD (1, .(FALSE), LIMIT$STATUS$ID) ;
310  2          CALL PUNCH$RECORD (1, .(0), EOF$ID) ;
311  2          CALL TX$EMPTY$DELAY ;                  /$DELAY WHILST TX REG EMPTIES$/
312  2          UART$CONTROL = SETTING (DIVIDE$16 AND
                                        ALT$0     AND
                                        RTS$LO    AND
                                        DISABLE$INT) ;
313  2        END DUMP ;
```

```
            $EJECT
 314   1        LOAD: PROCEDURE PUBLIC ;

 315   2            CALL POSITION$CURSOR (WARNING$FIELD) ;
 316   2            CALL WRITE$MESSAGE (.(DELETE$LINE,
                                         BLINK$ON,
                                         'LOADING TAPE',
                                         BLINK$OFF,
                                         BELL, EOT)) ;
 317   2            CALL TX$EMPTY$DELAY ;                  /$DELAY WHILST TX REG EMPTIES$/
 318   2            UART$CONTROL = SETTING (DIVIDE$64 AND
                                            ALT$0     AND
                                            RTS$LO    AND
                                            DISABLE$INT) ;
 319   2            IF TAPE$READ <> 0
                     THEN
 320   2                DO ;
 321   3                    UART$CONTROL = SETTING (DIVIDE$16 AND
                                                    ALT$0     AND
                                                    RTS$LO    AND
                                                    DISABLE$INT) ;
 322   3                   CALL POSITION$CURSOR (ERROR$FIELD) ;
 323   3                   CALL WRITE$MESSAGE (.(DELETE$LINE,
                                                 'TAPE READ ERROR',
                                                  BELL, EOT )) ;
 324   3                END ;
                     ELSE
 325   2                UART$CONTROL = SETTING (DIVIDE$16 AND
                                                ALT$0     AND
                                                RTS$LO    AND
                                                DISABLE$INT ) ;
 326   2            CALL CLEAR$WARNING$FIELD ;
 327   2            CALL POSITION$CURSOR (LIMITS$WARNING$FIELD) ;
 328   2            IF (X$UPPER$LIMIT$SET AND X$LOWER$LIMIT$SET AND
                       Y$UPPER$LIMIT$SET AND Y$LOWER$LIMIT$SET AND
                       Z$UPPER$LIMIT$SET AND Z$LOWER$LIMIT$SET    )
                     THEN
 329   2                CALL WRITE$MESSAGE (.(DELETE$LINE, EOT)) ;
                     ELSE
 330   2                CALL WRITE$MESSAGE (.LIMITS$WARNING$MESSAGE) ;
 331   2            IF PATH$SET
                     THEN
 332   2                CALL DISPLAY$PATH$NAME ;
 333   2        END LOAD ;
 334   1    END TAPE$IO ;



MODULE INFORMATION:

     CODE AREA SIZE     = 07E3H   2019D
     VARIABLE AREA SIZE = 0019H     25D
     MAXIMUM STACK SIZE = 000AH     10D
     731 LINES READ
     0 PROGRAM ERROR(S)

END OF PL/M-80 COMPILATION
```

```
ISIS-II PL/M-80 V3.1 COMPILATION OF MODULE RESETSYSTEMPROCEDURE
OBJECT MODULE PLACED IN :F1:WELD21.OBJ
COMPILER INVOKED BY:  PLM80 :F1:WELD21.PLM DEBUG WORKFILES(:F1:,:F1:) PRINT(:F1:WELD21.LST)


            $TITLE ('RESET$SYSTEM$PROCEDURE      VER 1.00     DATE 12/11/81')
  1         RESET$SYSTEM$PROCEDURE: DO ;

            /* A PLM80 PROCEDURE TO RESET THE SYSTEM HARDWARE.
               THE POSITIONING HARDWARE IS INITIALISED.
               THE SYSTEM STATUS FLAGS ARE INITIALISED.
               THE POWER SOURCE CONTROL PORT IS INITIALISED.
               THE CURRENT CONTROL PORT OUTPUT IS RESET TO ZERO.
               THE SYSTEM INTERRUPT MASKS ARE SET ACTIVE.
               THE UART IS INITIALISED.
            */

            /* EXTERNALS */
            $INCLUDE (:F1:WELD.EQU)
        =   $NOLIST
        =
            $INCLUDE (:F1:MATH.EQU)
        =   $NOLIST

  4   1         DECLARE UART$STATUS BYTE EXTERNAL ;
  5   1         DECLARE UART$DATA   BYTE EXTERNAL ;

  6   1         WRITE$MESSAGE: PROCEDURE (MESSAGE$ADDRESS) EXTERNAL ;
  7   2             DECLARE MESSAGE$ADDRESS ADDRESS ;
  8   2         END WRITE$MESSAGE ;

  9   1         SET$CONTROLLER$INDICATORS: PROCEDURE EXTERNAL ;
 10   2         END SET$CONTROLLER$INDICATORS ;

 11   1         RESET$POWER$SOURCE$CONTROL: PROCEDURE EXTERNAL ;
 12   2         END RESET$POWER$SOURCE$CONTROL ;

 13   1         SET$DIRECTION$CONTROL: PROCEDURE EXTERNAL ;
 14   2         END SET$DIRECTION$CONTROL ;

 15   1         SET$INTERRUPT$MASK: PROCEDURE EXTERNAL ;
 16   2         END SET$INTERRUPT$MASK ;

 17   1         FIX$SINGLE$ROUND: PROCEDURE (DATA$ADDRESS) EXTERNAL ;
 18   2             DECLARE DATA$ADDRESS ADDRESS ;
 19   2         END FIX$SINGLE$ROUND ;

 20   1         STORE$RESULT$SINGLE: PROCEDURE (STORE$ADDRESS) EXTERNAL ;
 21   2             DECLARE STORE$ADDRESS ADDRESS ;
 22   2         END STORE$RESULT$SINGLE ;

 23   1         DECLARE RESULT$SINGLE         (4) BYTE EXTERNAL ;
 24   1         DECLARE PARAMETERS$SET$FLAG (256) BYTE EXTERNAL ;
 25   1         DECLARE (R$0$ENABLED,
                        R$1$ENABLED,
                        R$2$ENABLED,
```

```
        R$3$ENABLED,
        CONTROLLER$ACTIVE,
        ON$SEAM,
        PATH$SET,
        MOTORS$RUNNING          )   BYTE EXTERNAL ;
```

```
                $EJECT
                /* PUBLIC DATA */
                   /* SYSTEM STATUS FLAGS */
 26   1            DECLARE  X$UPPER$LIMIT$SET              BYTE PUBLIC ;
 27   1            DECLARE  X$LOWER$LIMIT$SET              BYTE PUBLIC ;
 28   1            DECLARE  Y$UPPER$LIMIT$SET              BYTE PUBLIC ;
 29   1            DECLARE  Y$LOWER$LIMIT$SET              BYTE PUBLIC ;
 30   1            DECLARE  Z$UPPER$LIMIT$SET              BYTE PUBLIC ;
 31   1            DECLARE  Z$LOWER$LIMIT$SET              BYTE PUBLIC ;
 32   1            DECLARE  CURRENT$INCREMENT$SET          BYTE PUBLIC ;
 33   1            DECLARE  PATH$NAME$SET                  BYTE PUBLIC ;
 34   1            DECLARE  REST$POSITION$SET              BYTE PUBLIC ;

                   /* INTERRUPT MASK FLAGS */
 35   1            DECLARE  RST$5$5$MASK BYTE PUBLIC ;
 36   1            DECLARE  RST$6$5$MASK BYTE PUBLIC ;
 37   1            DECLARE  RST$7$5$MASK BYTE PUBLIC ;

                   /* TIMING DATA */
 38   1            DECLARE  MAIN$DIVISOR$FLOAT             (4) BYTE PUBLIC
                            DATA (05H, 0A0H, 00H, 00H) ;
 39   1            DECLARE RTC$DIVISOR$FLOAT               (4) BYTE PUBLIC
                            DATA (TEN$THOUSAND$REAL) ;
 40   1            DECLARE RTC$DIVISOR$SINGLE              (2) BYTE PUBLIC ;

                   /* REAL TIME INTERRUPT FLAGS */
 41   1            DECLARE  PRE$TRAVERSE$FLAG BYTE PUBLIC ;
 42   1            DECLARE  PULSE$ON$FLAG     BYTE PUBLIC ;
 43   1            DECLARE  PULSE$OFF$FLAG    BYTE PUBLIC ;
 44   1            DECLARE  FALL$FLAG         BYTE PUBLIC ;
 45   1            DECLARE  DELAY$FLAG        BYTE PUBLIC ;
 46   1            DECLARE  FALL$DELAY$FLAG   BYTE PUBLIC ;

                   /* DIRECTION CONTROL FLAGS */
 47   1            DECLARE (R$ENABLED,
                            Z$ENABLED,
                            Y$ENABLED,
                            X$ENABLED,
                            R$POSITIVE,
                            Z$POSITIVE,
                            Y$POSITIVE,
                            X$POSITIVE        ) BYTE PUBLIC ;
```

```
          $EJECT
48  1       DECLARE SIGN$ON$MESSAGE (*) BYTE PUBLIC
                  DATA ( PRINT$PORT$OFF,
                         CLEAR$SCREEN,
                         CURSOR$HOME,
                'TIG WELD VER 3.0  LIVERPOOL UNIVERSITY / WELDING INSTITUTE',
                CR$LF,
                CR$LF,
                '     X =                    Y =                    Z =',
                CR$LF,
                '      R0=              R1=              R2=              R3=',
                CR$LF,
                CR$LF,
                '     PATH NAME :-',
                CR$LF,
                '     GROUP :-                   IGNITION:-',
                CR$LF,
                '     CURRENT RANGE =                    CURRENT INCREMENT =',
                CR$LF,
                CR$LF,
                '     PULSE CURRENT =                    BACKGROUND CURRENT =',
                CR$LF,
                '     PULSE ON TIME =                    PULSE OFF TIME =',
                CR$LF,
                CR$LF,
                '     WELDING SPEED =                WIRE:-',
                CR$LF,
                CR$LF,
                '     PRE-TRAVERSE TIME =                FALL TIME =',
                CR$LF,
                CR$LF,
                CR$LF,
                '     PROGRAM',
                CR$LF,
                EOT                                                    ) ;
```

```
          $EJECT
          /* LOCAL DATA */

 49   1       DECLARE I ADDRESS ;
```

```
$EJECT

 50  1      RESET$SYSTEM: PROCEDURE PUBLIC ;
 51  2          X$ENABLED, Y$ENABLED, Z$ENABLED, R$ENABLED     = FALSE ;
 52  2          X$POSITIVE, Y$POSITIVE, Z$POSITIVE, R$POSITIVE = TRUE ;
 53  2          CALL SET$DIRECTION$CONTROL ;
 54  2          CALL RESET$POWER$SOURCE$CONTROL ;
 55  2          OUTPUT (CURRENT$CONTROL) = 0 ;
 56  2          OUTPUT (ROTATION) = 0 ;
 57  2          PRE$TRAVERSE$FLAG,
                PULSE$ON$FLAG,
                PULSE$OFF$FLAG,
                FALL$FLAG,
                DELAY$FLAG              = FALSE ;
 58  2          RST$5$5$MASK, RST$6$5$MASK, RST$7$5$MASK = TRUE ;
 59  2          CALL SET$INTERRUPT$MASK ;
 60  2          X$UPPER$LIMIT$SET,
                X$LOWER$LIMIT$SET,
                Y$UPPER$LIMIT$SET,
                Y$LOWER$LIMIT$SET,
                Z$UPPER$LIMIT$SET,
                Z$LOWER$LIMIT$SET       = FALSE ;
 61  2          CURRENT$INCREMENT$SET,
                PATH$SET,
                PATH$NAME$SET,
                REST$POSITION$SET       = FALSE ;
 62  2          DO I = 0 TO LAST(PARAMETERS$SET$FLAG) ;
 63  3              PARAMETERS$SET$FLAG(I) = FALSE ;
 64  3          END ;
 65  2          OUTPUT (MAIN$DIVIDER$MODE) = SETTING ( SELECT$0
                                                       AND MODE$2
                                                       AND READ$LOAD$LSB$MSB
                                                       AND BINARY           ) ;
 66  2          OUTPUT (MAIN$DIVIDER$MODE) = SETTING ( SELECT$1
                                                       AND MODE$0
                                                       AND READ$LOAD$LSB$MSB
                                                       AND BINARY           ) ;
 67  2          OUTPUT (MAIN$DIVIDER$MODE) = SETTING ( SELECT$2
                                                       AND MODE$0
                                                       AND READ$LOAD$LSB$MSB
                                                       AND BINARY           ) ;
 68  2          CALL FIX$SINGLE$ROUND (.MAIN$DIVISOR$FLOAT) ;
 69  2          OUTPUT (MAIN$DIVIDER) = RESULT$SINGLE(1) ;
 70  2          OUTPUT (MAIN$DIVIDER) = RESULT$SINGLE(0) ;
 71  2          CALL FIX$SINGLE$ROUND (.RTC$DIVISOR$FLOAT) ;
 72  2          CALL STORE$RESULT$SINGLE (.RTC$DIVISOR$SINGLE) ;
 73  2          OUTPUT (REAL$TIME$CLOCK) = RESULT$SINGLE(1) ;
 74  2          OUTPUT (REAL$TIME$CLOCK) = RESULT$SINGLE(0) ;
 75  2          OUTPUT (COUNTER$MODE) = SETTING ( SELECT$0
                                                  AND MODE$0
                                                  AND READ$LOAD$LSB$MSB
                                                  AND BINARY           ) ;
 76  2          OUTPUT (COUNTER$MODE) = SETTING ( SELECT$1
                                                  AND MODE$0
                                                  AND READ$LOAD$LSB$MSB
                                                  AND BINARY           ) ;
```

```
 77  2          OUTPUT (COUNTER$MODE) = SETTING ( SELECT$2
                                                  AND MODE$0
                                                  AND READ$LOAD$LSB$MSB
                                                  AND BINARY           ) ;

 78  2          OUTPUT (X$Y$Z$DIVIDER$MODE) = SETTING ( SELECT$0
                                                        AND MODE$2
                                                        AND READ$LOAD$LSB$MSB
                                                        AND BINARY           ) ;
 79  2          OUTPUT (X$Y$Z$DIVIDER$MODE) = SETTING ( SELECT$1
                                                        AND MODE$2
                                                        AND READ$LOAD$LSB$MSB
                                                        AND BINARY           ) ;
 80  2          OUTPUT (X$Y$Z$DIVIDER$MODE) = SETTING ( SELECT$2
                                                        AND MODE$2
                                                        AND READ$LOAD$LSB$MSB
                                                        AND BINARY           ) ;
 81  2          OUTPUT (CONTROL$8155)       = 0000$1101B ;
 82  2          R$0$ENABLED,
                R$1$ENABLED,
                R$2$ENABLED,
                R$3$ENABLED,
                CONTROLLER$ACTIVE,
                ON$SEAM,
                MOTORS$RUNNING              = FALSE ;
 83  2          CALL SET$CONTROLLER$INDICATORS ;
 84  2          UART$CONTROL = SETTING (RESET$UART
                                       AND ALT$0
                                       AND RTS$LO
                                       AND DISABLE$INT ) ;
 85  2          UART$CONTROL = SETTING (DIVIDE$16
                                       AND ALT$0
                                       AND RTS$LO
                                       AND DISABLE$INT ) ;
 86  2          MOTORS$RUNNING = FALSE ;
 87  2          CALL WRITE$MESSAGE (.SIGN$ON$MESSAGE) ;
 88  2        END RESET$SYSTEM ;
 89  1      END RESET$SYSTEM$PROCEDURE ;


MODULE INFORMATION:

    CODE AREA SIZE     = 0359H    857D
    VARIABLE AREA SIZE = 001EH     30D
    MAXIMUM STACK SIZE = 0002H      2D
    477 LINES READ
    0 PROGRAM ERROR(S)

END OF PL/M-80 COMPILATION
```

```
ISIS-II PL/M-80 V3.1 COMPILATION OF MODULE MOTORCALCPROCEDURE
OBJECT MODULE PLACED IN :F1:WELD24.OBJ
COMPILER INVOKED BY:  PLM80 :F1:WELD24.PLM DEBUG WORKFILES(:F1:,:F1:) PRINT(:F1:WELD24.LST)


                $TITLE ('MOTOR$CALC$PROCEDURE      VER 3.00     26/01/82')
  1             MOTOR$CALC$PROCEDURE: DO ;

                /* A PLM80 PROCEDURE TO CALCULATE THE X, Y & Z MOTOR INTERFACE COUNTER
                   AND DIVISOR VALUES FROM THE RELEVANT DISPLACEMENT AND SPEED VALUES. */


                $INCLUDE (:F1:WELD.EQU)
          =     $NOLIST
          =
                $INCLUDE (:F1:MATH.EQU)
          =     $NOLIST

                /* EXTERNAL ROUTINES */

  4   1             REAL$ADD: PROCEDURE (B$ADDRESS, A$ADDRESS) EXTERNAL ;
  5   2                 DECLARE (A$ADDRESS, B$ADDRESS) ADDRESS ;
  6   2             END REAL$ADD ;

  7   1             REAL$MULTIPLY: PROCEDURE (B$ADDRESS, A$ADDRESS) EXTERNAL ;
  8   2                 DECLARE (A$ADDRESS, B$ADDRESS) ADDRESS ;
  9   2             END REAL$MULTIPLY ;

 10   1             REAL$DIVIDE: PROCEDURE (B$ADDRESS, A$ADDRESS) EXTERNAL ;
 11   2                 DECLARE (A$ADDRESS, B$ADDRESS) ADDRESS ;
 12   2             END REAL$DIVIDE ;

 13   1             REAL$NEGATE: PROCEDURE (A$ADDRESS) EXTERNAL ;
 14   2                 DECLARE A$ADDRESS ADDRESS ;
 15   2             END REAL$NEGATE ;

 16   1             SUBTRACT$SINGLE: PROCEDURE (B$ADDRESS, A$ADDRESS) EXTERNAL ;
 17   2                 DECLARE (A$ADDRESS, B$ADDRESS) ADDRESS ;
 18   2             END SUBTRACT$SINGLE ;

 19   1             ADD$SINGLE$TO$DOUBLE: PROCEDURE (DOUBLE$ADDRESS, SINGLE$ADDRESS) EXTERNAL ;
 20   2                 DECLARE (DOUBLE$ADDRESS, SINGLE$ADDRESS) ADDRESS ;
 21   2             END ADD$SINGLE$TO$DOUBLE ;

 22   1             FLOAT$SINGLE: PROCEDURE (A$ADDRESS) EXTERNAL ;
 23   2                 DECLARE A$ADDRESS ADDRESS ;
 24   2             END FLOAT$SINGLE ;

 25   1             FIX$16$BIT: PROCEDURE (A$ADDRESS) EXTERNAL ;
 26   2                 DECLARE A$ADDRESS ADDRESS ;
 27   2             END FIX$16$BIT ;

 28   1             SQUARE$ROOT: PROCEDURE (A$ADDRESS) EXTERNAL ;
 29   2                 DECLARE A$ADDRESS ADDRESS ;
 30   2             END SQUARE$ROOT ;
```

```
 31  1          STORE$RESULT: PROCEDURE (STORE$ADDRESS) EXTERNAL ;
 32  2             DECLARE STORE$ADDRESS ADDRESS ;
 33  2          END STORE$RESULT ;

 34  1          STORE$RESULT$SINGLE: PROCEDURE (STORE$ADDRESS) EXTERNAL ;
 35  2             DECLARE STORE$ADDRESS ADDRESS ;
 36  2          END STORE$RESULT$SINGLE ;

          /* EXTERNAL DATA */

 37  1          DECLARE MATH$STATUS BYTE EXTERNAL ;
 38  1          DECLARE (FOUR,
                      FIVE,
                      FOUR$FLOAT,
                      FIVE$FLOAT,
                      ONE$FLOAT,
                      MINUS$ONE$FLOAT,
                      FIVE$HUNDRED$FLOAT,
                      FIFTY$FLOAT,
                      ONE$MILLION$FLOAT,
                      ZERO$FLOAT,
                      ONE$HUNDRED$FLOAT,
                      ONE$THOUSAND$FLOAT,
                      FOUR$HUNDRED$FLOAT,
                      RESULT          ) (4) BYTE EXTERNAL ;
 39  1          DECLARE (ONE$SINGLE,
                      MINUS$ONE$SINGLE,
                      ZERO$SINGLE,
                      RESULT$SINGLE  ) (2) BYTE EXTERNAL ;
 40  1          DECLARE (X$ORDINATE,
                      Y$ORDINATE,
                      Z$ORDINATE    )  (4) BYTE EXTERNAL ;
 41  1          DECLARE  MATH$IN$USE             BYTE PUBLIC ;
 42  1          DECLARE (X$DIVISOR$SINGLE,
                      X$COUNT$SINGLE,
                      Y$DIVISOR$SINGLE,
                      Y$COUNT$SINGLE,
                      Z$DIVISOR$SINGLE,
                      Z$COUNT$SINGLE    )      (2) BYTE PUBLIC ;
 43  1          DECLARE (X$ZERO, Y$ZERO, Z$ZERO,
                      X$DIRECTION$POSITIVE,
                      Y$DIRECTION$POSITIVE,
                      Z$DIRECTION$POSITIVE    ) BYTE PUBLIC ;

          /* LOCAL DATA */
 44  1              DECLARE (X$DISPLACEMENT$FLOAT,
                          Y$DISPLACEMENT$FLOAT,
                          Z$DISPLACEMENT$FLOAT,
                          X$VECTOR$FLOAT,
                          Y$VECTOR$FLOAT,
                          Z$VECTOR$FLOAT,
                          X$SQUARED$FLOAT,
                          Y$SQUARED$FLOAT,
                          DIVISOR$CONSTANT$FLOAT,
                          DISPLACEMENT$FLOAT              ) (4) BYTE ;
 45  1          MOTOR$CALCULATIONS: PROCEDURE (TABLE$ADDRESS, DIVISOR$FLOAT$ADDRESS, UPDATE) PUBLIC ;
 46  2             DECLARE (TABLE$ADDRESS, DIVISOR$FLOAT$ADDRESS) ADDRESS ;
```

```
 47  2          DECLARE UPDATE BYTE ;
 48  2          DECLARE TABLE$DATA BASED TABLE$ADDRESS (6) BYTE ;
 49  2          MATH$IN$USE = TRUE ;
 50  2          X$DIRECTION$POSITIVE,
                Y$DIRECTION$POSITIVE,
                Z$DIRECTION$POSITIVE       = TRUE ;
 51  2          X$ZERO,
                Y$ZERO,
                Z$ZERO   = FALSE ;
 52  2          IF UPDATE
                THEN
 53  2              DO ;
 54  3                  CALL ADD$SINGLE$TO$DOUBLE (.X$ORDINATE, .TABLE$DATA(0)) ;
 55  3                  CALL STORE$RESULT (.X$ORDINATE) ;
 56  3              END ;
 57  2          CALL FLOAT$SINGLE (.TABLE$DATA(0)) ;
 58  2          IF MATH$NEGATIVE
                THEN
 59  2              DO ;
 60  3                  CALL REAL$NEGATE (.RESULT) ;
 61  3                  X$DIRECTION$POSITIVE = FALSE ;
 62  3              END ;
 63  2          IF MATH$ZERO
                THEN
 64  2              X$ZERO = TRUE ;
 65  2          CALL STORE$RESULT (.X$DISPLACEMENT$FLOAT) ;
 66  2          CALL REAL$MULTIPLY (.RESULT, .RESULT) ;
 67  2          CALL STORE$RESULT (.X$SQUARED$FLOAT) ;
 68  2          IF UPDATE
                THEN
 69  2              DO ;
 70  3                  CALL ADD$SINGLE$TO$DOUBLE (.Y$ORDINATE, .TABLE$DATA(2)) ;
 71  3                  CALL STORE$RESULT (.Y$ORDINATE) ;
 72  3              END ;
 73  2          CALL FLOAT$SINGLE (.TABLE$DATA(2)) ;
 74  2          IF MATH$NEGATIVE
                THEN
 75  2              DO ;
 76  3                  CALL REAL$NEGATE (.RESULT) ;
 77  3                  Y$DIRECTION$POSITIVE = FALSE ;
 78  3              END ;
 79  2          IF MATH$ZERO
                THEN
 80  2              Y$ZERO = TRUE ;
 81  2          CALL STORE$RESULT (.Y$DISPLACEMENT$FLOAT) ;
 82  2          CALL REAL$MULTIPLY (.RESULT, .FIVE$FLOAT) ;
 83  2          CALL REAL$DIVIDE   (.RESULT, .FOUR$FLOAT) ;
 84  2          CALL REAL$MULTIPLY (.RESULT, .RESULT) ;
 85  2          CALL STORE$RESULT (.Y$SQUARED$FLOAT) ;
 86  2          IF UPDATE
                THEN
 87  2              DO ;
 88  3                  CALL ADD$SINGLE$TO$DOUBLE (.Z$ORDINATE, .TABLE$DATA(4)) ;
 89  3                  CALL STORE$RESULT (.Z$ORDINATE) ;
 90  3              END ;
 91  2          CALL FLOAT$SINGLE (.TABLE$DATA(4)) ;
 92  2          IF MATH$NEGATIVE
```

```
                    THEN
 93  2                  DO ;
 94  3                      CALL REAL$NEGATE (.RESULT) ;
 95  3                      Z$DIRECTION$POSITIVE = FALSE ;
 96  3                  END ;
 97  2                IF MATH$ZERO
                      THEN
 98  2                  Z$ZERO = TRUE ;
 99  2                CALL STORE$RESULT (.Z$DISPLACEMENT$FLOAT) ;
100  2                CALL REAL$MULTIPLY (.RESULT, .RESULT) ;
101  2                CALL REAL$ADD (.RESULT, .Y$SQUARED$FLOAT) ;
102  2                CALL REAL$ADD (.RESULT, .X$SQUARED$FLOAT) ;
103  2                CALL SQUARE$ROOT (.RESULT) ;
104  2                CALL REAL$MULTIPLY (.RESULT, DIVISOR$FLOAT$ADDRESS) ;
105  2                CALL STORE$RESULT (.DIVISOR$CONSTANT$FLOAT) ;
106  2                IF NOT X$ZERO
                      THEN
107  2                  DO ;
108  3                      CALL REAL$DIVIDE (.DIVISOR$CONSTANT$FLOAT, .X$DISPLACEMENT$FLOAT) ;
109  3                      CALL FIX$16$BIT (.RESULT) ;
110  3                      CALL STORE$RESULT$SINGLE (.X$DIVISOR$SINGLE) ;
111  3                  END ;
112  2                IF NOT Y$ZERO
                      THEN
113  2                  DO ;
114  3                      CALL REAL$DIVIDE (.DIVISOR$CONSTANT$FLOAT, .Y$DISPLACEMENT$FLOAT) ;
115  3                      CALL FIX$16$BIT (.RESULT) ;
116  3                      CALL STORE$RESULT$SINGLE (.Y$DIVISOR$SINGLE) ;
117  3                  END ;
118  2                IF NOT Z$ZERO
                      THEN
119  2                  DO ;
120  3                      CALL REAL$DIVIDE (.DIVISOR$CONSTANT$FLOAT, .Z$DISPLACEMENT$FLOAT) ;
121  3                      CALL FIX$16$BIT (.RESULT) ;
122  3                      CALL STORE$RESULT$SINGLE (.Z$DIVISOR$SINGLE) ;
123  3                  END ;
124  2                IF X$DIRECTION$POSITIVE
                      THEN
125  2                  CALL SUBTRACT$SINGLE (.TABLE$DATA(0), .ONE$SINGLE) ;
                      ELSE
126  2                  CALL SUBTRACT$SINGLE (.MINUS$ONE$SINGLE, .TABLE$DATA(0)) ;
127  2                CALL STORE$RESULT$SINGLE (.X$COUNT$SINGLE) ;
128  2                IF Y$DIRECTION$POSITIVE
                      THEN
129  2                  CALL SUBTRACT$SINGLE (.TABLE$DATA(2), .ONE$SINGLE) ;
                      ELSE
130  2                  CALL SUBTRACT$SINGLE (.MINUS$ONE$SINGLE, .TABLE$DATA(2)) ;
131  2                CALL STORE$RESULT$SINGLE (.Y$COUNT$SINGLE) ;
132  2                IF Z$DIRECTION$POSITIVE
                      THEN
133  2                  CALL SUBTRACT$SINGLE (.TABLE$DATA(4), .ONE$SINGLE) ;
                      ELSE
134  2                  CALL SUBTRACT$SINGLE (.MINUS$ONE$SINGLE, .TABLE$DATA(4)) ;
135  2                CALL STORE$RESULT$SINGLE (.Z$COUNT$SINGLE) ;
136  2                MATH$IN$USE = FALSE ;
137  2            END MOTOR$CALCULATIONS ;
```

```
 138  1      END MOTOR$CALC$PROCEDURE ;


MODULE INFORMATION:

     CODE AREA SIZE     = 0244H    580D
     VARIABLE AREA SIZE = 0040H     64D
     MAXIMUM STACK SIZE = 0002H      2D
     466 LINES READ
     0 PROGRAM ERROR(S)

END OF PL/M-80 COMPILATION
```

```
ISIS-II PL/M-80 V3.1 COMPILATION OF MODULE ARITHMETICPROCEDURES
OBJECT MODULE PLACED IN :F1:MATH.OBJ
COMPILER INVOKED BY:  PLM80 :F1:MATH.PLM DEBUG



           $TITLE ('ARITHMETIC$PROCEDURES  VER 2.00  DATE 03/04/81')
  1        ARITHMETIC$PROCEDURES: DO ;









           /*****************************************************************************
              AAA  RRRR  IIIII TTTTT H   H  M M  EEEEE TTTTT IIIII  CCCC
             A   A R   R   I     T   H   H M M M E       T     I   C
             AAAAA RRRR    I     T   HHHHH M M M EEE     T     I   C
             A   A R  R    I     T   H   H M   M E       T     I   C
             A   A R   R IIIII   T   H   H M   M EEEEE   T   IIIII  CCCC

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

                MODULE OF PLM80 ARITHMETIC PROCEDURES
           */

           /* EXTERNALS */
           $INCLUDE (:F1:WELD.EQU)
       =   $NOLIST
       =
           $INCLUDE (:F1:MATH.EQU)
       =   $NOLIST

           /* PUBLIC DATA */
  4   1        DECLARE MATH$STATUS       BYTE PUBLIC ;
  5   1        DECLARE RESULT        (4) BYTE PUBLIC ;
  6   1        DECLARE RESULT$SINGLE (2) BYTE PUBLIC ;

  7   1        DECLARE ONE$MILLION$FLOAT (4)    BYTE PUBLIC
                       DATA (14H, 0F4H, 24H,00H) ;
  8   1        DECLARE ONE          (4)         BYTE PUBLIC
                       DATA (00, 00, 00, 01) ;
  9   1        DECLARE ONE$SINGLE   (2)         BYTE PUBLIC AT (.ONE + 2) ;
 10   1        DECLARE MINUS$ONE    (4)         BYTE PUBLIC
                       DATA (0FFH, 0FFH, 0FFH, 0FFH) ;
 11   1        DECLARE MINUS$ONE$SINGLE   (2)   BYTE PUBLIC AT (.MINUS$ONE + 2) ;
 12   1        DECLARE FOUR         (4)         BYTE PUBLIC
                       DATA (00, 00, 00, 04) ;
 13   1        DECLARE TWO          (4)         BYTE PUBLIC
                       DATA (00, 00, 00, 02) ;
 14   1        DECLARE TWO$SINGLE   (2)         BYTE PUBLIC
                       AT (.TWO + 2) ;
 15   1        DECLARE TWO$FLOAT    (4)         BYTE PUBLIC
                       DATA (02, 80H, 00, 00) ;
 16   1        DECLARE THREE$SIX$ZERO$FLOAT (4) BYTE PUBLIC
                       DATA (09, 0B4H, 00, 00) ;
```

```
 17  1          DECLARE FIVE$HUNDRED$FLOAT  (4) BYTE PUBLIC
                        DATA (09H, 0FAH, 00H, 00H) ;
 18  1          DECLARE FOUR$HUNDRED$FLOAT  (4) BYTE PUBLIC
                        DATA (09H, 0C1H, 00H, 00H) ;
 19  1          DECLARE FIFTY$FLOAT         (4) BYTE PUBLIC
                        DATA (06H, 0CBH, 00H, 00H) ;
 20  1          DECLARE ONE$HUNDRED$FLOAT   (4) BYTE PUBLIC
                        DATA (07H, 0C8H, 00H, 00H) ;
 21  1          DECLARE TWO$THOUSAND$SINGLE (2) BYTE PUBLIC
                        DATA (07H, 0D0H) ;
 22  1          DECLARE FOUR$SINGLE  (2)        BYTE PUBLIC AT (.FOUR + 2) ;
 23  1          DECLARE FIVE         (4)        BYTE PUBLIC
                        DATA (00, 00, 00, 05) ;
 24  1          DECLARE FIVE$SINGLE  (2)        BYTE PUBLIC AT (.FIVE + 2) ;
 25  1          DECLARE TEN          (4)        BYTE PUBLIC
                        DATA (00, 00, 00, 10) ;
 26  1          DECLARE TEN$SINGLE   (2)        BYTE PUBLIC AT (.TEN + 2) ;
 27  1          DECLARE TEN$FLOAT     (4)       BYTE PUBLIC
                        DATA (04H, 0A0H, 00H, 00H) ;
 28  1          DECLARE POINT$FIVE$FLOAT   (4)  BYTE PUBLIC
                        DATA (00H, 80H, 00H, 00H) ;
 29  1          DECLARE ONE$FLOAT     (4)       BYTE PUBLIC
                        DATA (01H, 80H, 00H, 00H) ;
 30  1          DECLARE MINUS$ONE$FLOAT    (4)  BYTE PUBLIC
                        DATA (81H, 80H, 00H, 00H) ;
 31  1          DECLARE ZERO$FLOAT    (4)       BYTE PUBLIC
                        DATA (00H, 00H, 00H, 00H) ;
 32  1          DECLARE ZERO          (4)       BYTE PUBLIC AT (.ZERO$FLOAT) ;
 33  1          DECLARE ZERO$SINGLE   (2)       BYTE PUBLIC
                        AT (.ZERO$FLOAT + 2) ;
 34  1          DECLARE SIX$FIVE$FIVE$THREE$FIVE (4) BYTE PUBLIC
                        DATA (00H, 00H, 0FFH, 0FFH) ;
 35  1          DECLARE TWO$FIVE$FIVE$FLOAT (4) BYTE PUBLIC
                        DATA (08H, 0FFH, 00H, 00H) ;
 36  1          DECLARE TWO$FIVE$FIVE       (4) BYTE PUBLIC
                        DATA (00H, 00H, 00H, 0FFH) ;
 37  1          DECLARE THREE$TWO$SEVEN$SIX$SEVEN (4) BYTE PUBLIC
                        DATA (00H, 00H, 7FH, 0FFH) ;
 38  1          DECLARE TEN$THOUSAND$FLOAT  (4) BYTE PUBLIC
                        DATA (0EH, 9CH, 40H, 00H) ;
 39  1          DECLARE FOUR$FLOAT             (4) BYTE PUBLIC
                        DATA (03H, 80H, 00H, 00H) ;
 40  1          DECLARE FIVE$FLOAT             (4) BYTE PUBLIC
                        DATA (03H, 0A0H, 00H, 00H) ;

 41  1          DECLARE ONE$THOUSAND$FLOAT     (4) BYTE PUBLIC
                        DATA (0AH, 0FAH, 00H, 00H) ;
```

```
                $EJECT

 42  1          WAIT$FOR$ARITHMETIC$PROCESSOR: PROCEDURE PUBLIC ;
 43  2                DO WHILE (ARITHMETIC$BUSY$STATUS) <> 0 ;
 44  3                END ;
 45  2          END WAIT$FOR$ARITHMETIC$PROCESSOR ;

 46  1          PUSH$DATA$TO$APU: PROCEDURE (DATA$ADDRESS) PUBLIC ;
 47  2                DECLARE DATA$ADDRESS ADDRESS ;
 48  2                DECLARE DATA$BYTE BASED DATA$ADDRESS (4) BYTE ;
 49  2                DECLARE I BYTE ;
 50  2                DO I = 0 TO 3 ;
 51  3                      OUTPUT (ARITHMETIC$DATA) = DATA$BYTE(3 - I) ;
 52  3                END ;
 53  2          END PUSH$DATA$TO$APU ;

 54  1          PUSH$DATA$SINGLE: PROCEDURE (DATA$ADDRESS) PUBLIC ;
 55  2                DECLARE DATA$ADDRESS ADDRESS ;
 56  2                DECLARE DATA$BYTE BASED DATA$ADDRESS (2) BYTE ;
 57  2                OUTPUT (ARITHMETIC$DATA) = DATA$BYTE(1) ;
 58  2                OUTPUT (ARITHMETIC$DATA) = DATA$BYTE(0) ;
 59  2          END PUSH$DATA$SINGLE ;

 60  1          POP$RESULT$FROM$APU: PROCEDURE PUBLIC ;
 61  2                DECLARE I BYTE ;
 62  2                MATH$STATUS = INPUT (ARITHMETIC$STATUS) ;
 63  2                DO I = 0 TO 3 ;
 64  3                      RESULT(I) = INPUT (ARITHMETIC$DATA) ;
 65  3                END ;
 66  2          END POP$RESULT$FROM$APU ;

 67  1          POP$RESULT$SINGLE: PROCEDURE PUBLIC ;
 68  2                MATH$STATUS = INPUT (ARITHMETIC$STATUS) ;
 69  2                RESULT$SINGLE(0) = INPUT (ARITHMETIC$DATA) ;
 70  2                RESULT$SINGLE(1) = INPUT (ARITHMETIC$DATA) ;
 71  2          END POP$RESULT$SINGLE ;

 72  1          ADD: PROCEDURE (B$ADDRESS, A$ADDRESS) PUBLIC ;
 73  2                DECLARE (A$ADDRESS, B$ADDRESS) ADDRESS ;

 74  2                CALL PUSH$DATA$TO$APU (B$ADDRESS) ;
 75  2                CALL PUSH$DATA$TO$APU (A$ADDRESS) ;
 76  2                OUTPUT (ARITHMETIC$COMMAND) = DADD ;
 77  2                CALL WAIT$FOR$ARITHMETIC$PROCESSOR ;
 78  2                CALL POP$RESULT$FROM$APU ;
 79  2          END ADD ;

 80  1          SUBTRACT: PROCEDURE (B$ADDRESS, A$ADDRESS) PUBLIC ;
 81  2                DECLARE (A$ADDRESS, B$ADDRESS) ADDRESS ;

 82  2                CALL PUSH$DATA$TO$APU (B$ADDRESS) ;
 83  2                CALL PUSH$DATA$TO$APU (A$ADDRESS) ;
 84  2                OUTPUT (ARITHMETIC$COMMAND) = DSUB ;
 85  2                CALL WAIT$FOR$ARITHMETIC$PROCESSOR ;
 86  2                CALL POP$RESULT$FROM$APU ;
 87  2          END SUBTRACT ;
```

```
 88  1          MULTIPLY: PROCEDURE (B$ADDRESS, A$ADDRESS) PUBLIC ;
 89  2                DECLARE (A$ADDRESS, B$ADDRESS) ADDRESS ;

 90  2                CALL PUSH$DATA$TO$APU (B$ADDRESS) ;
 91  2                CALL PUSH$DATA$TO$APU (A$ADDRESS) ;
 92  2                OUTPUT (ARITHMETIC$COMMAND) = DMUL ;
 93  2                CALL WAIT$FOR$ARITHMETIC$PROCESSOR ;
 94  2                CALL POP$RESULT$FROM$APU ;
 95  2          END MULTIPLY ;

 96  1          DIVIDE: PROCEDURE (B$ADDRESS, A$ADDRESS) PUBLIC ;
 97  2                DECLARE (A$ADDRESS, B$ADDRESS) ADDRESS ;

 98  2                CALL PUSH$DATA$TO$APU (B$ADDRESS) ;
 99  2                CALL PUSH$DATA$TO$APU (A$ADDRESS) ;
100  2                OUTPUT (ARITHMETIC$COMMAND) = DDIV ;
101  2                CALL WAIT$FOR$ARITHMETIC$PROCESSOR ;
102  2                CALL POP$RESULT$FROM$APU ;
103  2          END DIVIDE ;

104  1          FLOAT: PROCEDURE (A$ADDRESS) PUBLIC ;
105  2                DECLARE A$ADDRESS ADDRESS ;
106  2                CALL PUSH$DATA$TO$APU (A$ADDRESS) ;
107  2                OUTPUT (ARITHMETIC$COMMAND) = FLTD ;
108  2                CALL WAIT$FOR$ARITHMETIC$PROCESSOR ;
109  2                CALL POP$RESULT$FROM$APU ;
110  2          END FLOAT ;

111  1          FLOAT$SINGLE: PROCEDURE (A$ADDRESS) PUBLIC ;
112  2                DECLARE A$ADDRESS ADDRESS ;
113  2                CALL PUSH$DATA$SINGLE (A$ADDRESS) ;
114  2                OUTPUT (ARITHMETIC$COMMAND) = FLTS ;
115  2                CALL WAIT$FOR$ARITHMETIC$PROCESSOR ;
116  2                CALL POP$RESULT$FROM$APU ;
117  2          END FLOAT$SINGLE ;

118  1          FIX: PROCEDURE (A$ADDRESS) PUBLIC ;
119  2                DECLARE A$ADDRESS ADDRESS ;
120  2                CALL PUSH$DATA$TO$APU (A$ADDRESS) ;
121  2                OUTPUT (ARITHMETIC$COMMAND) = FIXD ;
122  2                CALL WAIT$FOR$ARITHMETIC$PROCESSOR ;
123  2                CALL POP$RESULT$FROM$APU ;
124  2                IF MATH$OVERFLOW
                       THEN
125  2                   RESULT(0), RESULT(1), RESULT(2), RESULT(3) = 0FFH ;
126  2          END FIX ;

127  1          FIX$SINGLE: PROCEDURE (A$ADDRESS) PUBLIC ;
128  2                DECLARE A$ADDRESS ADDRESS ;
129  2                CALL PUSH$DATA$TO$APU (A$ADDRESS) ;
130  2                OUTPUT (ARITHMETIC$COMMAND) = FIXS ;
131  2                CALL WAIT$FOR$ARITHMETIC$PROCESSOR ;
132  2                CALL POP$RESULT$SINGLE ;
133  2          END FIX$SINGLE ;

134  1           ADD$SINGLE: PROCEDURE (B$ADDRESS, A$ADDRESS) PUBLIC ;
135  2              DECLARE (B$ADDRESS, A$ADDRESS) ADDRESS ;
```

```
136  2              CALL PUSH$DATA$SINGLE (B$ADDRESS) ;
137  2              CALL PUSH$DATA$SINGLE (A$ADDRESS) ;
138  2              OUTPUT (ARITHMETIC$COMMAND) = SADD ;
139  2              CALL WAIT$FOR$ARITHMETIC$PROCESSOR ;
140  2              CALL POP$RESULT$SINGLE ;
141  2           END ADD$SINGLE ;

142  1           STORE$RESULT: PROCEDURE (STORE$ADDRESS) PUBLIC ;
143  2                 DECLARE STORE$ADDRESS ADDRESS ;
144  2                 DECLARE STORE BASED STORE$ADDRESS (4) BYTE ;
145  2                 DECLARE I BYTE ;
146  2                 DO I = 0 TO 3 ;
147  3                       STORE(I) = RESULT(I) ;
148  3                 END ;
149  2           END STORE$RESULT ;

150  1           STORE$RESULT$SINGLE: PROCEDURE (STORE$ADDRESS) PUBLIC ;
151  2                 DECLARE STORE$ADDRESS ADDRESS ;
152  2                 DECLARE STORE BASED STORE$ADDRESS (2) BYTE ;
153  2                 STORE(0) = RESULT$SINGLE(0) ;
154  2                 STORE(1) = RESULT$SINGLE(1) ;
155  2           END STORE$RESULT$SINGLE ;

156  1           STORE$RESULT$SINGLE$ADDRESS: PROCEDURE (STORE$ADDRESS) PUBLIC ;
157  2              DECLARE STORE$ADDRESS ADDRESS ;
158  2              DECLARE STORE BASED STORE$ADDRESS (2) BYTE ;
159  2              STORE (0) = RESULT$SINGLE (1) ;
160  2              STORE (1) = RESULT$SINGLE (0) ;
161  2           END STORE$RESULT$SINGLE$ADDRESS ;

162  1           STORE$NUMBER: PROCEDURE (DESTINATION$ADDRESS, SOURCE$ADDRESS) PUBLIC ;
163  2                 DECLARE (DESTINATION$ADDRESS, SOURCE$ADDRESS) ADDRESS ;
164  2                 DECLARE SOURCE$DATA BASED SOURCE$ADDRESS (4) BYTE ;
165  2                 DECLARE DESTINATION$DATA BASED DESTINATION$ADDRESS (4) BYTE ;
166  2                 DECLARE I BYTE ;
167  2                 DO I = 0 TO 3 ;
168  3                     DESTINATION$DATA(I) = SOURCE$DATA(I) ;
169  3                 END ;
170  2           END STORE$NUMBER ;

171  1           STORE$NUMBER$SINGLE: PROCEDURE (DESTINATION$ADDRESS, SOURCE$ADDRESS) PUBLIC ;
172  2                 DECLARE (DESTINATION$ADDRESS, SOURCE$ADDRESS) ADDRESS ;
173  2                 DECLARE SOURCE$DATA BASED SOURCE$ADDRESS (2) BYTE ;
174  2                 DECLARE DESTINATION$DATA BASED DESTINATION$ADDRESS (2) BYTE ;
175  2                 DESTINATION$DATA(0) = SOURCE$DATA(0) ;
176  2                 DESTINATION$DATA(1) = SOURCE$DATA(1) ;
177  2           END STORE$NUMBER$SINGLE ;

178  1           FIX$ROUND: PROCEDURE (A$ADDRESS) PUBLIC ;
179  2                 DECLARE A$ADDRESS ADDRESS ;
180  2                 DECLARE TEMP$FLOAT (4) BYTE ;
181  2                 CALL STORE$NUMBER (.TEMP$FLOAT, A$ADDRESS) ;
182  2                 CALL FIX (.TEMP$FLOAT) ;
183  2                 IF MATH$OVERFLOW
                       THEN
184  2                     RETURN ;
185  2                 CALL PUSH$DATA$TO$APU (.TEMP$FLOAT) ;
```

```
186  2                CALL PUSH$DATA$TO$APU (.RESULT) ;
187  2                OUTPUT (ARITHMETIC$COMMAND) = FLTD ;
188  2                CALL WAIT$FOR$ARITHMETIC$PROCESSOR ;
189  2                OUTPUT (ARITHMETIC$COMMAND) = FSUB ;
190  2                CALL WAIT$FOR$ARITHMETIC$PROCESSOR ;
191  2                CALL PUSH$DATA$TO$APU (.POINT$FIVE$FLOAT) ;
192  2                OUTPUT (ARITHMETIC$COMMAND) = FSUB ;
193  2                CALL WAIT$FOR$ARITHMETIC$PROCESSOR ;
194  2                MATH$STATUS = INPUT (ARITHMETIC$STATUS) ;
195  2                IF NOT (MATH$NEGATIVE)
                       THEN
196  2                    CALL ADD (.RESULT, .ONE) ;
197  2            END FIX$ROUND ;

198  1            FIX$16$BIT: PROCEDURE (A$ADDRESS) PUBLIC ;
199  2                DECLARE A$ADDRESS ADDRESS ;
200  2                CALL FIX$ROUND (A$ADDRESS) ;
201  2                CALL PUSH$DATA$TO$APU (.SIX$FIVE$FIVE$THREE$FIVE) ;
202  2                CALL PUSH$DATA$TO$APU (.RESULT) ;
203  2                OUTPUT (ARITHMETIC$COMMAND) = DSUB ;
204  2                CALL WAIT$FOR$ARITHMETIC$PROCESSOR ;
205  2                MATH$STATUS = INPUT (ARITHMETIC$STATUS) ;
206  2                IF MATH$NEGATIVE
                       THEN
207  2                    RESULT$SINGLE(0), RESULT$SINGLE(1) = OFFH ;
                       ELSE
208  2                    DO ;
209  3                        RESULT$SINGLE(0) = RESULT(2) ;
210  3                        RESULT$SINGLE(1) = RESULT(3) ;
211  3                    END ;
212  2            END FIX$16$BIT ;

213  1            FIX$8$BIT: PROCEDURE (A$ADDRESS) PUBLIC ;
214  2                DECLARE A$ADDRESS ADDRESS ;
215  2                CALL FIX$ROUND (A$ADDRESS) ;
216  2                CALL PUSH$DATA$TO$APU (.TWO$FIVE$FIVE) ;
217  2                CALL PUSH$DATA$TO$APU (.RESULT) ;
218  2                OUTPUT (ARITHMETIC$COMMAND) = DSUB ;
219  2                CALL WAIT$FOR$ARITHMETIC$PROCESSOR ;
220  2                MATH$STATUS = INPUT (ARITHMETIC$STATUS) ;
221  2                IF MATH$NEGATIVE
                       THEN
222  2                    RESULT$SINGLE(1) = OFFH ;
                       ELSE
223  2                    RESULT$SINGLE(1) = RESULT(3) ;
224  2            END FIX$8$BIT ;

225  1            FIX$SINGLE$ROUND: PROCEDURE (A$ADDRESS) PUBLIC ;
226  2                DECLARE A$ADDRESS ADDRESS ;
227  2                DECLARE TEMP$FLOAT (4) BYTE ;
228  2                CALL STORE$NUMBER (.TEMP$FLOAT, A$ADDRESS) ;
229  2                CALL FIX$SINGLE (A$ADDRESS) ;
230  2                IF MATH$OVERFLOW
                       THEN
231  2                    RETURN ;
232  2                CALL PUSH$DATA$TO$APU (.TEMP$FLOAT) ;
233  2                CALL PUSH$DATA$SINGLE (.RESULT$SINGLE) ;
```

```
234  2                OUTPUT (ARITHMETIC$COMMAND) = FLTS ;
235  2                CALL WAIT$FOR$ARITHMETIC$PROCESSOR ;
236  2                OUTPUT (ARITHMETIC$COMMAND) = FSUB ;
237  2                CALL WAIT$FOR$ARITHMETIC$PROCESSOR ;
238  2                CALL PUSH$DATA$TO$APU (.POINT$FIVE$FLOAT) ;
239  2                OUTPUT (ARITHMETIC$COMMAND) = FSUB ;
240  2                CALL WAIT$FOR$ARITHMETIC$PROCESSOR ;
241  2                MATH$STATUS = INPUT (ARITHMETIC$STATUS) ;
242  2                IF NOT (MATH$NEGATIVE)
                      THEN
243  2                    CALL ADD$SINGLE (.RESULT$SINGLE, .ONE$SINGLE) ;
244  2           END FIX$SINGLE$ROUND ;

245  1           NEGATE: PROCEDURE (A$ADDRESS) PUBLIC ;
246  2                DECLARE A$ADDRESS ADDRESS ;
247  2                CALL PUSH$DATA$TO$APU (A$ADDRESS) ;
248  2                OUTPUT (ARITHMETIC$COMMAND) = CHSD ;
249  2                CALL WAIT$FOR$ARITHMETIC$PROCESSOR ;
250  2                CALL POP$RESULT$FROM$APU ;
251  2           END NEGATE ;

252  1           REAL$MULTIPLY: PROCEDURE (B$ADDRESS, A$ADDRESS) PUBLIC ;
253  2                DECLARE (A$ADDRESS, B$ADDRESS) ADDRESS ;
254  2                CALL PUSH$DATA$TO$APU (B$ADDRESS) ;
255  2                CALL PUSH$DATA$TO$APU (A$ADDRESS) ;
256  2                OUTPUT (ARITHMETIC$COMMAND) = FMUL ;
257  2                CALL WAIT$FOR$ARITHMETIC$PROCESSOR ;
258  2                CALL POP$RESULT$FROM$APU ;
259  2           END REAL$MULTIPLY ;

260  1           REAL$DIVIDE: PROCEDURE (B$ADDRESS, A$ADDRESS) PUBLIC ;
261  2                DECLARE (A$ADDRESS, B$ADDRESS) ADDRESS ;
262  2                CALL PUSH$DATA$TO$APU (B$ADDRESS) ;
263  2                CALL PUSH$DATA$TO$APU (A$ADDRESS) ;
264  2                OUTPUT (ARITHMETIC$COMMAND) = FDIV ;
265  2                CALL WAIT$FOR$ARITHMETIC$PROCESSOR ;
266  2                CALL POP$RESULT$FROM$APU ;
267  2           END REAL$DIVIDE ;

268  1           REAL$ADD: PROCEDURE (B$ADDRESS, A$ADDRESS) PUBLIC ;
269  2                DECLARE (A$ADDRESS, B$ADDRESS) ADDRESS ;
270  2                CALL PUSH$DATA$TO$APU (B$ADDRESS) ;
271  2                CALL PUSH$DATA$TO$APU (A$ADDRESS) ;
272  2                OUTPUT (ARITHMETIC$COMMAND) = FADD ;
273  2                CALL WAIT$FOR$ARITHMETIC$PROCESSOR ;
274  2                CALL POP$RESULT$FROM$APU ;
275  2           END REAL$ADD ;

276  1           REAL$SUBTRACT: PROCEDURE (B$ADDRESS, A$ADDRESS) PUBLIC ;
277  2                DECLARE (A$ADDRESS, B$ADDRESS) ADDRESS ;
278  2                CALL PUSH$DATA$TO$APU (B$ADDRESS) ;
279  2                CALL PUSH$DATA$TO$APU (A$ADDRESS) ;
280  2                OUTPUT (ARITHMETIC$COMMAND) = FSUB ;
281  2                CALL WAIT$FOR$ARITHMETIC$PROCESSOR ;
282  2                CALL POP$RESULT$FROM$APU ;
283  2           END REAL$SUBTRACT ;
```

```
 284  1          REAL$NEGATE: PROCEDURE (A$ADDRESS) PUBLIC ;
 285  2                DECLARE A$ADDRESS ADDRESS ;
 286  2                CALL PUSH$DATA$TO$APU (A$ADDRESS) ;
 287  2                OUTPUT (ARITHMETIC$COMMAND) = CHSF ;
 288  2                CALL WAIT$FOR$ARITHMETIC$PROCESSOR ;
 289  2                CALL POP$RESULT$FROM$APU ;
 290  2          END REAL$NEGATE ;

 291  1          SQUARE$ROOT: PROCEDURE (A$ADDRESS) PUBLIC ;
 292  2                DECLARE A$ADDRESS ADDRESS ;
 293  2                CALL PUSH$DATA$TO$APU (A$ADDRESS) ;
 294  2                OUTPUT (ARITHMETIC$COMMAND) = SQRT ;
 295  2                CALL WAIT$FOR$ARITHMETIC$PROCESSOR ;
 296  2                CALL POP$RESULT$FROM$APU ;
 297  2          END SQUARE$ROOT ;

 298  1           NEGATE$SINGLE: PROCEDURE (A$ADDRESS) PUBLIC ;
 299  2               DECLARE A$ADDRESS ADDRESS ;
 300  2               CALL PUSH$DATA$SINGLE (A$ADDRESS) ;
 301  2               OUTPUT (ARITHMETIC$COMMAND) = CHSS;
 302  2               CALL WAIT$FOR$ARITHMETIC$PROCESSOR ;
 303  2               CALL POP$RESULT$SINGLE ;
 304  2           END NEGATE$SINGLE ;

 305  1           DIVIDE$SINGLE: PROCEDURE (B$ADDRESS, A$ADDRESS) PUBLIC ;
 306  2               DECLARE (A$ADDRESS, B$ADDRESS) ADDRESS ;
 307  2               CALL PUSH$DATA$SINGLE (B$ADDRESS) ;
 308  2               CALL PUSH$DATA$SINGLE (A$ADDRESS) ;
 309  2               OUTPUT (ARITHMETIC$COMMAND) = SDIV ;
 310  2               CALL WAIT$FOR$ARITHMETIC$PROCESSOR ;
 311  2               CALL POP$RESULT$SINGLE ;
 312  2           END DIVIDE$SINGLE ;

 313  1           MULTIPLY$SINGLE: PROCEDURE (B$ADDRESS, A$ADDRESS) PUBLIC ;
 314  2               DECLARE (A$ADDRESS, B$ADDRESS) ADDRESS ;
 315  2               CALL PUSH$DATA$SINGLE (B$ADDRESS) ;
 316  2               CALL PUSH$DATA$SINGLE (A$ADDRESS) ;
 317  2               OUTPUT (ARITHMETIC$COMMAND) = SMUL ;
 318  2               CALL WAIT$FOR$ARITHMETIC$PROCESSOR ;
 319  2               CALL POP$RESULT$SINGLE ;
 320  2           END MULTIPLY$SINGLE ;

 321  1           SUBTRACT$SINGLE: PROCEDURE (B$ADDRESS, A$ADDRESS) PUBLIC ;
 322  2               DECLARE (A$ADDRESS, B$ADDRESS) ADDRESS ;
 323  2               CALL PUSH$DATA$SINGLE (B$ADDRESS) ;
 324  2               CALL PUSH$DATA$SINGLE (A$ADDRESS) ;
 325  2               OUTPUT (ARITHMETIC$COMMAND) = SSUB ;
 326  2               CALL WAIT$FOR$ARITHMETIC$PROCESSOR ;
 327  2               CALL POP$RESULT$SINGLE ;
 328  2           END SUBTRACT$SINGLE ;

 329  1           ADD$SINGLE$TO$DOUBLE: PROCEDURE (DOUBLE$ADDRESS, SINGLE$ADDRESS) PUBLIC ;
 330  2                 DECLARE (DOUBLE$ADDRESS, SINGLE$ADDRESS) ADDRESS ;
 331  2                 CALL PUSH$DATA$SINGLE (SINGLE$ADDRESS) ;
 332  2                 CALL PUSH$DATA$SINGLE (.ZERO$SINGLE) ;
 333  2                 OUTPUT (ARITHMETIC$COMMAND) = SADD ;
 334  2                 CALL WAIT$FOR$ARITHMETIC$PROCESSOR ;
```

```
 335  2                  MATH$STATUS = INPUT (ARITHMETIC$STATUS) ;
 336  2                  IF MATH$NEGATIVE
                         THEN
 337  2                     CALL PUSH$DATA$SINGLE (.MINUS$ONE$SINGLE) ;
                         ELSE
 338  2                     CALL PUSH$DATA$SINGLE (.ZERO$SINGLE) ;
 339  2                  CALL PUSH$DATA$TO$APU   (DOUBLE$ADDRESS) ;
 340  2                  OUTPUT (ARITHMETIC$COMMAND) = DADD ;
 341  2                  CALL WAIT$FOR$ARITHMETIC$PROCESSOR ;
 342  2                  CALL POP$RESULT$FROM$APU ;
 343  2              END ADD$SINGLE$TO$DOUBLE ;


 344  1              MULT$BY$2$PI: PROCEDURE (DATA$ADDRESS) PUBLIC ;
 345  2                  DECLARE DATA$ADDRESS ADDRESS ;
 346  2                  CALL REAL$MULTIPLY (DATA$ADDRESS, .TWO$FLOAT) ;
 347  2                  CALL PUSH$DATA$TO$APU (.RESULT) ;
 348  2                  OUTPUT (ARITHMETIC$COMMAND) = PUPI ;
 349  2                  CALL WAIT$FOR$ARITHMETIC$PROCESSOR ;
 350  2                  OUTPUT (ARITHMETIC$COMMAND) = FMUL ;
 351  2                  CALL WAIT$FOR$ARITHMETIC$PROCESSOR ;
 352  2                  CALL POP$RESULT$FROM$APU ;
 353  2              END MULT$BY$2$PI ;

 354  1              TANGENT: PROCEDURE (DATA$ADDRESS) PUBLIC ;
 355  2                  DECLARE DATA$ADDRESS ADDRESS ;
 356  2                  CALL PUSH$DATA$TO$APU (DATA$ADDRESS) ;
 357  2                  OUTPUT (ARITHMETIC$COMMAND) = TAN ;
 358  2                  CALL WAIT$FOR$ARITHMETIC$PROCESSOR ;
 359  2                  CALL POP$RESULT$FROM$APU ;
 360  2              END TANGENT ;

 361  1              SINE: PROCEDURE (DATA$ADDRESS) PUBLIC ;
 362  2                  DECLARE DATA$ADDRESS ADDRESS ;
 363  2                  CALL PUSH$DATA$TO$APU (DATA$ADDRESS) ;
 364  2                  OUTPUT (ARITHMETIC$COMMAND) = SIN ;
 365  2                  CALL WAIT$FOR$ARITHMETIC$PROCESSOR ;
 366  2                  CALL POP$RESULT$FROM$APU ;
 367  2              END SINE ;

 368  1              COSINE: PROCEDURE (DATA$ADDRESS) PUBLIC ;
 369  2                  DECLARE DATA$ADDRESS ADDRESS ;
 370  2                  CALL PUSH$DATA$TO$APU (DATA$ADDRESS) ;
 371  2                  OUTPUT (ARITHMETIC$COMMAND) = COS ;
 372  2                  CALL WAIT$FOR$ARITHMETIC$PROCESSOR ;
 373  2                  CALL POP$RESULT$FROM$APU ;
 374  2              END COSINE ;

 375  1      END ARITHMETIC$PROCEDURES ;


MODULE INFORMATION:

     CODE AREA SIZE     = 0637H   1591D
     VARIABLE AREA SIZE = 0079H    121D
     MAXIMUM STACK SIZE = 0006H      6D
```

732 LINES READ
0 PROGRAM ERROR(S)

END OF PL/M-80 COMPILATION

```
ASM80 :F1:WELD16.ASM DEBUG MOD85


ISIS-II 8080/8085 MACRO ASSEMBLER, V4.0        MOTOR    PAGE    1


 LOC  OBJ         LINE       SOURCE STATEMENT

                    1 ;ASSEMBLY LANGUAGE UTILITEIS TO HANDLE LOADING OF DATA TO THE STEPPING MOTOR
                    2 ;INTERFACE AND SERVICE INTERRUPTS (RST6.5) GENERATED BY THE MOTORS.
                    3
                    4         NAME    MOTOR
                    5         PUBLIC  SETDIR, STRMTR, MTRINT
                    6         EXTRN   XENBL,  YENBL,  ZENBL, RENBL
                    7         EXTRN   XPOS,   YPOS,   ZPOS,  RPOS
                    8         EXTRN   XZERO,  YZERO,  ZZERO
                    9         EXTRN   XDRPOS, YDRPOS, ZDRPOS
                   10         EXTRN   MTRUN,  MSK65,  SETMSK
                   11         EXTRN   XDIVSR, YDIVSR, ZDIVSR
                   12         EXTRN   XCOUNT, YCOUNT, ZCOUNT
                   13
                   14         CSEG
                   15
                   16
                   17 ;SETDIR
                   18 ;AN ASSEMBLY LANGUAGE ROUTINE TO SET THE DIRECTION/CONTROL PORT OF THE
                   19 ;STEPPING MOTOR INTERFACE.
                   20 ;INPUTS :-
                   21 ;          INPUT DATA IS PASSED AS A TABLE OF 6 DATA BYTES AS BELOW.
                   22 ;          BYTE 0   RENBL   FF=ENABLE R MOTOR      00=DISABLE R MOTOR
                   23 ;          BYTE 1   ZENBL   FF=ENABLE Z MOTOR      00=DISABLE Z MOTOR
                   24 ;          BYTE 2   YENBL   FF=ENABLE Y MOTOR      00=DISABLE Y MOTOR
                   25 ;          BYTE 3   XENBL   FF=ENABLE X MOTOR      00=DISABLE X MOTOR
                   26 ;          BYTE 4   RPOS    FF=R DIRECTION POS     00=R DIRECTION NEG
                   27 ;          BYTE 5   ZPOS    FF=Z DIRECTION POS     00=Z DIRECTION NEG
                   28 ;          BYTE 6   YPOS    FF=Y DIRECTION POS     00=Y DIRECTION NEG
                   29 ;          BYTE 7   XPOS    FF=X DIRECTION POS     00=X DIRECTION NEG
                   30
0000 AF            31 SETDIR: XRA     A
0001 47            32         MOV     B, A          ;CLEAR REG B
0002 0E08          33         MVI     C, 8          ;LOAD BYTE COUNTER
0004 210000    E   34         LXI     H, RENBL      ;POINT TO DATA TABLE
0007 CD1200    C   35 LOOP:   CALL    ROTATE        ;ROTATE LEAST SIG BIT OF DATA INTO REG B
000A 0D            36         DCR     C
000B C20700    C   37         JNZ     LOOP          ;JUMP IF NOT LAST BYTE
000E 78            38         MOV     A,B
000F D31D          39         OUT     1DH           ;DET DIRECTION/CONTROL PORT
0011 C9            40         RET
                   41
                   42 ;ROTATE
                   43 ;AN ASSEMBLY LANGUAGE ROUTINE TO ROTATE BIT 0 OF TABLE DATA INTO REG B AND
                   44 ;POINT TO NEXT TABLE DATA BYTE.
0012 7E            45 ROTATE: MOV     A, M          ;LOAD REG A WITH TABLE DATA
0013 0F            46         RRC                   ;ROTATE LEAST SIG BIT INTO CARRY FLAG
0014 78            47         MOV     A, B
0015 17            48         RAL                   ;ROTATE CARRY FLAG INTO LEAST SIG BIT
0016 47            49         MOV     B, A
0017 23            50         INX     H             ;INCREMENT TABLE POINTER
0018 C9            51         RET
                   52
                   53
                   54
```

```
LOC  OBJ          LINE       SOURCE STATEMENT

                    55 ;MTRINT
                    56 ;AN ASSEMBLY LANGUAGE ROUTINE TO HANDLE MOTOR INTERRUPTS (RST 6.5)
                    57 ;OUTPUTS :-
                    58 ;          THE MOTOR RUNNING FLAG IS SET FALSE
                    59 ;          RST 6.5 IS MASKED IS SET
                    60 ;          DATA IS STORED IN 8 BYTE TABLE AS BELOW
                    61 ;          BIT 0   RENBL  = 00 INDICATING R MOTOR IS DISABLED
                    62 ;          BIT 1   ZENBL  = 00 INDICATING Z MOTOR IS DISABLED
                    63 ;          BIT 2   YENBL  = 00 INDICATING Y MOTOR IS DISABLED
                    64 ;          BIT 3   XENBL  = 00 INDICATING X MOTOR IS DISABLED
                    65 ;          BIT 4   RPOS   = FF INDICATING R DIRECTION IS SET POSITIVE
                    66 ;          BIT 5   ZPOS   = FF INDICATING Z DIRECTION IS SET POSITIVE
                    67 ;          BIT 6   YPOS   = FF INDICATING Y DIRECTION IS SET POSITIVE
                    68 ;          BIT 7   XPOS   = FF INDICATING X DIRECTION IS SET POSITIVE
                    69
0019 210000   E     70 MTRINT: LXI     H, RENBL          ;POINT TO DATA TABLE
001C AF             71         XRA     A
001D 77             72         MOV     M, A              ;RENBL = FALSE
001E 23             73         INX     H
001F 77             74         MOV     M, A              ;ZENBL = FALSE
0020 23             75         INX     H
0021 77             76         MOV     M, A              ;YENBL = FALSE
0022 23             77         INX     H
0023 77             78         MOV     M, A              ;XENBL = FALSE
0024 3EFF           79         MVI     A, 0FFH
0026 23             80         INX     H
0027 77             81         MOV     M, A              ;RPOS = TRUE
0028 23             82         INX     H
0029 77             83         MOV     M, A              ;ZPOS = TRUE
002A 23             84         INX     H
002B 77             85         MOV     M, A              ;YPOS = TRUE
002C 23             86         INX     H
002D 77             87         MOV     M, A              ;XPOS = TRUE
002E CD0000   C     88         CALL    SETDIR            ;SET MOTOR DIRECTION/CONTROL PORT
0031 AF             89         XRA     A
0032 320000   E     90         STA     MTRUN             ;MOTOR RUNNING = FALSE
0035 3EFF           91         MVI     A, 0FFH
0037 320000   E     92         STA     MSK65             ;MSK65 = TRUE
003A CD0000   E     93         CALL    SETMSK            ;SET RST6.5 MASK
003D C9             94         RET
                    95
                    96
                    97
                    98 ;STRMTR
                    99 ;AN ASSEMBLY LANGUAGE ROUTINE TO LOAD THE COUNTERS, DIVIDERS AND DIRECTION/CONTROL PORT
                   100 ;OF THE STEPPING MOTOR INTERFACE.
                   101 ;INPUTS:-
                   102 ;       XDRPOS   = FF IF X MOTOR IS TO MOVE IN POSITIVE DIRECTION.
                   103 ;                = 00 IF X MOTOR IS TO MOVE IN NEGATIVE DIRECTION.
                   104 ;       YDRPOS   = FF IF Y MOTOR IS TO MOVE IN POSITIVE DIRECTION.
                   105 ;                = 00 IF Y MOTOR IS TO MOVE IN NEGATIVE DIRECTION.
                   106 ;       ZDRPOS   = FF IF Z MOTOR IS TO MOVE IN POSITIVE DIRECTION.
                   107 ;                = 00 IF Z MOTOR IS TO MOVE IN NEGATIVE DIRECTION.
                   108 ;       XZERO    = FF IF X MOTOR IS NOT TO BE STARTED.
                   109 ;                = 00 IF X MOTOR IS TO BE STARTED.
```

```
 LOC  OBJ          LINE       SOURCE STATEMENT

                    110 ;       YZERO    = FF IF Y MOTOR IS NOT TO BE STARTED.
                    111 ;                = 00 IF Y MOTOR IS TO BE STARTED.
                    112 ;       ZZERO    = FF IF Z MOTOR IS NOT TO BE STARTED.
                    113 ;                = 00 IF Z MOTOR IS TO BE STARTED.
                    114 ;       THE COUNT AND DIVISOR VALUES ARE PASSED IN A TABLE OF DATA AS BELOW
                    115 ;       BYTE 0    X DIVISOR MSB
                    116 ;       BYTE 1    X DIVISOR LSB
                    117 ;       BYTE 2    X COUNT   MSB
                    118 ;       BYTE 3    X COUNT   LSB
                    119 ;       BYTE 4    Y DIVISOR MSB
                    120 ;       BYTE 5    Y DIVISOR LSB
                    121 ;       BYTE 6    Y COUNT   MSB
                    122 ;       BYTE 7    Y COUNT   LSB
                    123 ;       BYTE 8    Z DIVISOR MSB
                    124 ;       BYTE 9    Z DIVISOR LSB
                    125 ;       BYTE 10   Z COUNT   MSB
                    126 ;       BYTE 11   Z COUNT   LSB
                    127 ;
                    128 ;OUTPUTS:-
                    129 ;       MTRUN     = FF INDICATES THAT THE MOTOR IS RUNNING.
                    130 ;                 = 00 INDICATES THAT THE MOTOR IS NOT RUNNING.
                    131
 003E AF            132 STRMTR: XRA     A
 003F 320000    E   133         STA     XENBL
 0042 320000    E   134         STA     YENBL
 0045 320000    E   135         STA     ZENBL
 0048 210100    E   136         LXI     H, ZCOUNT + 1           ; POINT TO BOTTOM OF DATA TABLE
 004B 3A0000    E   137         LDA     ZZERO
 004E A7            138         ANA     A
 004F C26F00    C   139         JNZ     TAB1                    ;JUMP IF ZZERO = TRUE
 0052 7E            140         MOV     A, M
 0053 D31A          141         OUT     1AH                     ;LOAD Z COUNTER LSB
 0055 2B            142         DCX     H
 0056 7E            143         MOV     A, M
 0057 D31A          144         OUT     1AH                     ;LOAD Z COUNTER MSB
 0059 2B            145         DCX     H
 005A 7E            146         MOV     A, M
 005B D316          147         OUT     16H                     ;LOAD Z DIVIDER LSB
 005D 2B            148         DCX     H
 005E 7E            149         MOV     A, M
 005F D316          150         OUT     16H                     ;LOAD Z DIVIDER MSB
 0061 3A0000    E   151         LDA     ZDRPOS
 0064 320000    E   152         STA     ZPOS                    ;SET Z DIRECTION
 0067 3EFF          153         MVI     A, 0FFH
 0069 320000    E   154         STA     ZENBL                   ;Z ENABLED = TRUE
 006C C37200    C   155         JMP     TAB2
 006F 2B            156 TAB1:   DCX     H
 0070 2B            157         DCX     H
 0071 2B            158         DCX     H                       ;SET UP TABLE POINTER FOR Y DATA
 0072 3A0000    E   159 TAB2:   LDA     YZERO
 0075 A7            160         ANA     A
 0076 C29700    C   161         JNZ     TAB3                    ;JUMP IF YZERO = TRUE
 0079 2B            162         DCX     H
 007A 7E            163         MOV     A, M
 007B D319          164         OUT     19H                     ;LOAD Y COUNTER LSB
```

```
 LOC  OBJ          LINE        SOURCE STATEMENT

 007D 2B           165         DCX     H
 007E 7E           166         MOV     A, M
 007F D319         167         OUT     19H                 ;LOAD Y COUNTER MSB
 0081 2B           168         DCX     H
 0082 7E           169         MOV     A, M
 0083 D315         170         OUT     15H                 ;LOAD Y DIVIDER LSB
 0085 2B           171         DCX     H
 0086 7E           172         MOV     A, M
 0087 D315         173         OUT     15H                 ;LOAD Y DIVIDER MSB
 0089 3A0000   E   174         LDA     YDRPOS
 008C 320000   E   175         STA     YPOS                ;SET Y DIRECTION
 008F 3EFF         176         MVI     A, 0FFH
 0091 320000   E   177         STA     YENBL               ;Y ENABLED = TRUE
 0094 C39B00   C   178         JMP     TAB4
 0097 2B           179 TAB3:   DCX     H
 0098 2B           180         DCX     H
 0099 2B           181         DCX     H
 009A 2B           182         DCX     H                   ;SET UP TABLE POINTER FOR X DATA
 009B 3A0000   E   183 TAB4:   LDA     XZERO
 009E A7           184         ANA     A
 009F C2BD00   C   185         JNZ     TAB5                ;JUMP IF XZERO = TRUE
 00A2 2B           186         DCX     H
 00A3 7E           187         MOV     A, M
 00A4 D318         188         OUT     18H                 ;LOAD X COUNTER LSB
 00A6 2B           189         DCX     H
 00A7 7E           190         MOV     A, M
 00A8 D318         191         OUT     18H                 ;LOAD X COUNTER MSB
 00AA 2B           192         DCX     H
 00AB 7E           193         MOV     A, M
 00AC D314         194         OUT     14H                 ;LOAD X DIVIDER LSB
 00AE 2B           195         DCX     H
 00AF 7E           196         MOV     A, M
 00B0 D314         197         OUT     14H                 ;LOAD X DIVIDER MSB
 00B2 3A0000   E   198         LDA     XDRPOS
 00B5 320000   E   199         STA     XPOS                ;SET X DIRECTION
 00B8 3EFF         200         MVI     A, 0FFH
 00BA 320000   E   201         STA     XENBL               ;X ENABLED = TRUE
 00BD 320000   E   202 TAB5:   STA     MTRUN               ;MOTOR RUNNING = TRUE
 00C0 CD0000   C   203         CALL    SETDIR              ;ENABLE MOTORS TO RUN
 00C3 AF           204         XRA     A
 00C4 320000   E   205         STA     MSK65
 00C7 CD0000   E   206         CALL    SETMSK              ;UNMASK RST6.5
 00CA FB           207         EI
 00CB C9           208         RET
                   209
                   210 END
```

PUBLIC SYMBOLS

```
MTRINT C 0019    SETDIR C 0000    STRMTR C 003E
```

EXTERNAL SYMBOLS

```
MSK65  E 0000    MTRUN  E 0000    RENBL  E 0000    RPOS   E 0000    SETMSK E 0000    XCOUNT E 0000    XDIVSR E 0000
XDRPOS E 0000    XENBL  E 0000    XPOS   E 0000    XZERO  E 0000    YCOUNT E 0000    YDIVSR E 0000    YDRPOS E 0000
YENBL  E 0000    YPOS   E 0000    YZERO  E 0000    ZCOUNT E 0000    ZDIVSR E 0000    ZDRPOS E 0000    ZENBL  E 0000
```

```
ZPOS   E 0000    ZZERO  E 0000

USER SYMBOLS
LOOP   C 0007    MSK65  E 0000    MTRINT C 0019    MTRUN  E 0000    RENBL  E 0000    ROTATE C 0012    RPOS   E 0000
SETDIR C 0000    SETMSK E 0000    STRMTR C 003E    TAB1   C 006F    TAB2   C 0072    TAB3   C 0097    TAB4   C 009B
TAB5   C 00BD    XCOUNT E 0000    XDIVSR E 0000    XDRPOS E 0000    XENBL  E 0000    XPOS   E 0000    XZERO  E 0000
YCOUNT E 0000    YDIVSR E 0000    YDRPOS E 0000    YENBL  E 0000    YPOS   E 0000    YZERO  E 0000    ZCOUNT E 0000
ZDIVSR E 0000    ZDRPOS E 0000    ZENBL  E 0000    ZPOS   E 0000    ZZERO  E 0000

ASSEMBLY COMPLETE,   NO ERRORS
```

```
ASM80 :F1:WELD13.ASM DEBUG MOD85


ISIS-II 8080/8085 MACRO ASSEMBLER, V4.0        INTRS    PAGE    1


  LOC  OBJ         LINE       SOURCE STATEMENT

                     1 ; ASSEMBLY LANGUAGE PROGRAM TO HANDLE INTERRUPTS.
                     2 ; ALL PROGRAM REGISTERS ARE SAVED AND THE APPROPIATE PLM80 ROUTINE CALLED.
                     3 ; ALL REGISTERS ARE RESTORED BEFORE RETURNING FROM INTERRUPT.
                     4
                     5
                     6          NAME    INTRS
                     7          EXTRN   MTRINT, TIMINT, ARCINT
                     8          PUBLIC  INT65, INT55, INT75
                     9
                    10          CSEG
                    11
  0000 F5           12 INT65:  PUSH    PSW
  0001 E5           13         PUSH    H
  0002 C5           14         PUSH    B
  0003 D5           15         PUSH    D
  0004 CD0000   E   16         CALL    MTRINT
  0007 D1           17         POP     D
  0008 C1           18         POP     B
  0009 E1           19         POP     H
  000A F1           20         POP     PSW
  000B FB           21         EI
  000C C9           22         RET
                    23
  000D F5           24 INT55:  PUSH    PSW
  000E E5           25         PUSH    H
  000F C5           26         PUSH    B
  0010 D5           27         PUSH    D
  0011 CD0000   E   28         CALL    TIMINT
  0014 D1           29         POP     D
  0015 C1           30         POP     B
  0016 E1           31         POP     H
  0017 F1           32         POP     PSW
  0018 FB           33         EI
  0019 C9           34         RET
                    35
  001A F5           36 INT75:  PUSH    PSW
  001B E5           37         PUSH    H
  001C C5           38         PUSH    B
  001D D5           39         PUSH    D
  001E CD0000   E   40         CALL    ARCINT
  0021 D1           41         POP     D
  0022 C1           42         POP     B
  0023 E1           43         POP     H
  0024 F1           44         POP     PSW
  0025 FB           45         EI
  0026 C9           46         RET
                    47
                    48 END


PUBLIC SYMBOLS
INT55  C 000D    INT65  C 0000    INT75  C 001A

EXTERNAL SYMBOLS
```

```
ARCINT E 0000    MTRINT E 0000    TIMINT E 0000

USER SYMBOLS
ARCINT E 0000    INT55  C 000D    INT65  C 0000    INT75  C 001A    MTRINT E 0000    TIMINT E 0000

ASSEMBLY COMPLETE,   NO ERRORS
```

```
ASM80 :F1:WELD17.ASM DEBUG MOD85


ISIS-II 8080/8085 MACRO ASSEMBLER, V4.0        ARCINT   PAGE    1


  LOC  OBJ         LINE       SOURCE STATEMENT

                     1 ; AN ASSEMBLY LANGUAGE ROUTINE TO HANDLE 'ARC OUT' INTERRUPTS (RST$7$5)
                     2         NAME    ARCINT
                     3         PUBLIC  ARCINT
                     4         EXTRN   MSK75, SETMSK, ARCDID, SPOWER, STRTEN, STOPEN, WIRENB
                     5
                     6         CSEG
  0000 3EFF          7 ARCINT: MVI     A, 0FFH
  0002 320000    E   8         STA     ARCDID          ; ARC$DIED = TRUE
  0005 320000    E   9         STA     MSK75           ; RST$7$5$MASK = TRUE
  0008 CD0000    E  10         CALL    SETMSK          ; SET INTERRUPT MASK
  000B AF           11         XRA     A
  000C 320000    E  12         STA     STRTEN          ;START$ENABLE = FALSE
  000F 320000    E  13         STA     WIRENB          ;WIRE$ENABLE = FALSE
  0012 2F           14         CMA
  0013 320000    E  15         STA     STOPEN          ;STOP$ENABLE = TRUE
  0016 CD0000    E  16         CALL    SPOWER          ;SET POWER SOURCE CONTORL
  0019 C9           17         RET
                    18         END

PUBLIC SYMBOLS
ARCINT C 0000

EXTERNAL SYMBOLS
ARCDID E 0000    MSK75  E 0000    SETMSK E 0000    SPOWER E 0000    STOPEN E 0000    STRTEN E 0000    WIRENB E 0000

USER SYMBOLS
ARCDID E 0000    ARCINT C 0000    MSK75  E 0000    SETMSK E 0000    SPOWER E 0000    STOPEN E 0000    STRTEN E 0000
WIRENB E 0000

ASSEMBLY COMPLETE,   NO ERRORS
```

```
ASM80 :F1:WELD6.ASM DEBUG MOD85


ISIS-II 8080/8085 MACRO ASSEMBLER, V4.0        SETMSK   PAGE    1


 LOC  OBJ         LINE       SOURCE STATEMENT

                    1 ; A ASSEMBLY LANGAUGE PROGRAM TO ALLOW PLM80 PROGRAMS TO SET THE INTERRUPT MASK.
                    2 ;    IF MSK75 = TRUE THEN RESTART 7.5 IS MASKED
                    3 ;    IF MSK65 = TRUE THEN RESTART 6.5 IS MASKED
                    4 ;    IF MSK55 = TRUE THEN RESTART 5.5 IS MASKED
                    5
                    6         NAME    SETMSK
                    7         EXTRN   MSK75, MSK65, MSK55
                    8         PUBLIC  SETMSK, RSTFF
                    9
                   10         CSEG
                   11
 0000 17           12 ROTATE: RAL
 0001 78           13         MOV     A, B
 0002 17           14         RAL
 0003 47           15         MOV     B, A
 0004 C9           16         RET
                   17
 0005 0601         18 SETMSK: MVI     B, 1
 0007 3A0000   E   19         LDA     MSK75
 000A CD0000   C   20         CALL    ROTATE
 000D 3A0000   E   21         LDA     MSK65
 0010 CD0000   C   22         CALL    ROTATE
 0013 3A0000   E   23         LDA     MSK55
 0016 CD0000   C   24         CALL    ROTATE
 0019 78           25         MOV     A, B
 001A 30           26         SIM
 001B C9           27         RET
                   28
                   29 ;AN ASSEMBLY LANGUAGE ROUTINE TO ALLOW PLM80 PROGRAMS TO RESET THE
                   30 ;RST 7.5 FLIP FLOP
                   31
 001C 3E10         32 RSTFF:  MVI     A, 00010000B
 001E 30           33         SIM
 001F C9           34         RET
                   35
                   36 END


PUBLIC SYMBOLS
RSTFF  C 001C    SETMSK C 0005

EXTERNAL SYMBOLS
MSK55  E 0000    MSK65  E 0000    MSK75  E 0000

USER SYMBOLS
MSK55  E 0000    MSK65  E 0000    MSK75  E 0000    ROTATE C 0000    RSTFF  C 001C    SETMSK C 0005

ASSEMBLY COMPLETE,   NO ERRORS
```

```
ASM80 :F1:WELD23.ASM DEBUG MOD85


ISIS-II 8080/8085 MACRO ASSEMBLER, V4.0        VECTRS   PAGE    1


  LOC  OBJ         LINE       SOURCE STATEMENT

                     1 ;##############################################################################################
                     2
                     3 ;                              TIG$WELD INTERRUPT VECTORS
                     4
                     5 ;                         0024H   CONTROL IS PASSED TO 5000H
                     6 ;                         002CH   CONTROL IS PASSED TO INT$55
                     7 ;                         0034H   CONTROL IS PASSED TO INT$65
                     8 ;                         003CH   CONTROL IS PASSED TO INT$75
                     9 ;
                    10 ;##############################################################################################
                    11
                    12          NAME    VECTRS
                    13          EXTRN   INT55, INT65, INT75
                    14
                    15
  0024              16          ORG     0024H
  0024 C30050       17          JMP     5000H
                    18
  002C              19          ORG     002CH
  002C C30000   E   20          JMP     INT55
                    21
  0034              22          ORG     0034H
  0034 C30000   E   23          JMP     INT65
                    24
  003C              25          ORG     003CH
  003C C30000   E   26          JMP     INT75
                    27
                    28 END


PUBLIC SYMBOLS


EXTERNAL SYMBOLS
INT55  E 0000    INT65  E 0000    INT75  E 0000

USER SYMBOLS
INT55  E 0000    INT65  E 0000    INT75  E 0000

ASSEMBLY COMPLETE,   NO ERRORS
```