THE UNIVERSITY OF LIVERPOOL

THE DESIGN AND PRODUCTION OF AN AUTOMATED TIG (Tungsten Inert Gas) WELDING MACHINE

Thesis submitted in accordance with the requirements of the University of Liverpool for the degree of Doctor in Philosophy by MARK PENDLEBURY

Department of Electrical and Electronic Engineering

July, 1986.

ACKNOWLEDGEMENTS

I would like to thank Dr. J. Lucas for his supervision and guidance of this research work. I also wish to thank Professor J.D. Parsons, Head of the Department of Electrical and Electronic Engineering, and all the other members of the department for providing the facilities needed and their advice and assistance.

The work has been funded by the Wolfson Corporation ,The Welding Institute and the Science and Engineering Research Council under the CASE award scheme.

Thank you also to Apricot Research & Development for the loan of the computer and word processor used in producing this thesis.

ABSTRACT

The TIM robot was designed five years ago for use as a research high precision welding machine using the TIG (Tungsten-Inert-Gas) process. An original research program showed the practicality of microprocessor control of the welding process. This research was to improve and expand the original system to industrial standards and allow adaptation for future research.

The iAPX 8086 processor was used for the control of the entire welding operation. The robot position, movement, pulsed arc welding current and peripheral handling are controlled in accordance with a preprogrammed and stored set of parameters.

The hardware system was designed and assembled using, in the main, the Intel single board computer hardware system with some custom design. The software operating system was written mainly in high level language (PL/M 86) to achieve maximum flexibility and to keep the code as modular and expandable as possible. Assembly language (ASM 86) was used to achieve fast response and greater hardware control at lower levels. The system was built using the iRMX original end manufaturer (OEM) software operating system.

The software was developed using 'Top Down' methodology and emphasis, at all times during the research, was placed on expandability and modular construction to accommodate present and future peripheral research and development work.

CONTENTS

CHAPTER 1.

INTRODUCTION.

The subject of this research work is the automation of the Tungsten-Inert-Gas (TIG) welding process. This work is the continuation of preliminary work [1], at Liverpool University, relating to the control of the TIG process. It is also being done concurrently with other research work into peripherals to improve the quality of TIG welding. In order to highlight the features which have influenced the design and development of the final Superweld system, this chapter gives basic background on TIG welding and the past and present research work into TIG welding peripherals at Liverpool University.

1.1 ARC WELDING.

1.1.1 General Arc Welding Considerations.

Arc welding is a fusion process for joining metals [2], where the heat needed for melting the metal is supplied by an electric arc. A pool of molten metal forms under the arc which extend down into the workpiece, as shown in figure 1.1. It is usual to prevent the weld pool from oxidizing in some way. This can be done by using a powder (as in submerged arc welding) or an inert gas (carbon dioxide as in Metal-Inert-Gas welding or argon in TIG welding), to saturate the welding area. The gas may not be strictly inert, but is treated as such if it harmless to the welding process. If the thickness of the metal to be joined is not excessive (less than 5mm) then the joint may be achieved by simply melting the workpieces along the seam, thus forming a weld.

However for thicker metal (greater than 5mm) a multipass technique must be employed to successfully join the two pieces with full penetration. This multipass technique is shown in figure 1.2. The workpiece preparation, figure 1.2(a), allows the first pass, figure 1.2(b), to penetrate fully . Futher passes then require a metal to be fed into the weld pool as the joint progresses, in order to fill the joint, figures 1.2(c) and 1.2(d). Filler wire is usually of a similar material to that of the workpieces and takes the form of a wire or rod fed in by a wirefeed unit. This is one of the peripherals which must be controlled by the Superweld system.

The heat input to the weld pool is an important factor, since it has a major influence on the resulting weld. The higher the heat input rate the deeper the penetration and the larger the weld pool, with the subsequent cooling rate being lower. Too low a cooling rate gives rise to a coarse metallurgical grain structure, hence a less ductile seam. Too high a cooling rate may give rise to cracking of the seam. The heat input rate must therefore be strictly controlled. This rate is determined by both the welding power supply output and the traverse speed of the torch both of which must be strictly controlled by any welding system.

1.1.2 The Tungsten-Inert-Gas Process.

This form of welding produces some of the highest quality welds possible and is therefore very important to the Aircraft, Chemical and Nuclear Power Industries. The

TIG process is illustrated in figure 1.3(b). An electrical arc is struck between the workpiece and a tungsten electrode. The heat generated produces the weld pool in the workpiece but is not sufficient to melt the electrode. The electrode's polarity is usually negative with respect to the workpiece, since about twice the heat is generated in the anode region as in the cathode region and also the molten metal would tend to 'spatter' towards the electrode were the polarities reversed. The shielding gas, usually argon, is supplied through the welding torch to saturate the area above the weld pool. The gas may alternatively be 5% hydrogen and 95% argon as this increases the weld pool temperature.

Pulsed TIG is a development of the basic direct current TIG process. Here the current is pulsed between the peak and background current (I_p and I_b) (see figure 1.4). I_b is chosen to maintain a stable arc without causing any melting of the workpiece, 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 the degree of overlap is determined by the pulse parameters and the torch velocity. This method allows greater control of penetration depth than does the D.C. TIG welding.

The pulsed TIG process is used where precision, high integrity welds are required. It can be used on a wide variety of metals, such as, steels, copper and aluminium as shown in figure 1.5. TIG welding now accounts for around 28% of all welds [3].

1.2 Past and Present TIG Research.

In order to automate the TIG welding process Liverpool University has been researching into TIG welding controllers and peripherals for 8 years starting with a straight line TIG welder [4], which controlled a low current power supply and a single axis traverse system.

1.2.1 The TIG Intelligent Machine (TIM) Robot.

The TIM robot was developed and built by the Electrical Engineering mechanical workshops in 1980 and is currently under redesign by a research project in the Mechanical Engineering Department. The basic design shown in figure 1.6 is an XYZ cartesian co-ordinate robot with various different wrist configurations. A table (Y-axis) (shown in figure 1.7) provides support for the workpiece and an overhead linear tracking system provides the X and Z axes, moving the torch directly. All axes are driven using stepping motors driven by high performance bipolar-bilevel drivers [5]. The linear axes are coupled to their respective motors via ground leadscrews with recirculating-ball nuts, giving minimum backlash and friction. To keep the velocity constant we need to know the gearing of the X, Y and Z axes. The storage of the distances moved by each of the axes is done in steps, these correspond to 0.01mm on the X and Y axes and 0.005mm on the Z axis.

1.2.2 The Quarndon Welding System.

The Quarndon Welding System (QWS) [1] was developed

at Liverpool University 1980-1982. It is based round the eight bit 8085 Intel processor and a specialized maths card, with 64K bytes of mixed EPROM and RAM with a magnetic cassette tape interface for mass storage. The system controlled a 50 Amp power supply and the TIM robot.

The QWS was able to accurately describe and store a path which was entered by the operator using a hand held pendant to drive the robot. Current values were entered by the operator and could be stored on cassette tape along with the path description. The Welding Institute used the system for a while for welding research and found it welded successfully. The system though had its restrictions with a limit on path size and no ability to edit a path once entered. Therefore paths of great complexity had to be entered several times until correct. Storage was also found to be slow and somewhat unreliable using the cassette system.

The major drawback though was its lack of expandability due to it being an eight bit system only able to access 64K byte and relatively slow processing speed compared to the newer 16 bit processors. With the small QWS system already saturated with just the problem of robot and current control and with the research group's embarkation on research into the following peripherals, the need for a large faster expandable system was seen and this project was undertaken.

1.2.3 Seam Trackers.

One of the main difficulties with welding is the tendency of the heat produced to cause warping of the

workpiece during the welding process and render an accurately input path useless. The answer is to know at all times where the path is and update the controller on the present position of the seam during welding.

Four seam trackers are at present being developed. The first two are visual techniques [6], one using white light, the other laser light. They spread a band of light across the seam and a camera attached to the robot monitors the band of structured light. A computer then receives the picture, digitises it and evaluates it to find the gap in the band of light. Once it has identified this gap as the seam it can monitor the movement of it relative to the torch tip and hence keep the torchtip always over the centre of the seam. In future feedback techniques can be used to inform the main controller of any path deviations and keep the torch on the seam.

The other seam trackers are low cost units, one using infra-red transmitters and receivers to bounce signals off the workpiece. Then using a mechanical scanning technique it finds the seam and monitors its movement again relative to the torch head [7].

The final seam tracker is an acoustic model using ultrasonic techniques to detect the seam [8].

All these trackers will in the future need some method of communication to relate information, about the torch's position relative to the seam, back to the main controller. At present Local Area Networks (LAN's) and parallel buses are being looked into.

1.2.4 Weld Penetration Monitoring.

The most important thing to control in the welding process is also one of the hardest, this being the control of the weld penetration. The penetration depth as stated before is dependent on the arc energy supplied, this being dependent on both the arc temperature and the traverse velocity. The present research into arc penetration monitoring has produced several techniques, one of which is the monitoring of the weld pool height using lasers [9]. As the weld pool grows the molten metal expands and the height of the weld pool increases forming a bead on the surface of the plate. Once penetration occurs however the bead drops slightly due to gravity and moves, with respect to the torch, due to gas pressure. With lasers the exact time at which this occurs can be detected. Lasers are used due to the very small size of movement. If this information is relayed to the controller it can update its current waveform to drop to the background current (I_b). If penetration does not occur the controller may also extend the pulse current time until it does so or until a maximum pulse time is reached.

The controller therefore needs to monitor the penetration monitor's data and adjust its current output accordingly. The controller must also be notified if the weld penetration monitor is being used or not, if present.

1.2.5 Other Sensors.

All the above sensors are intelligent microprocessor controlled units, but large numbers of simple sensors also exist, such as gas flow, wirefeed and arc voltage

monitors. These will all have to be included in a final system and the more advanced sensors may wish to obtain information from them directly. This means the Superweld system had to be designed with possible expansion as the most important feature.

This thesis shows the design and development of a welding control system to cater for all the above requirements and it's integration into the welding research currently being undertaken at Liverpool University.

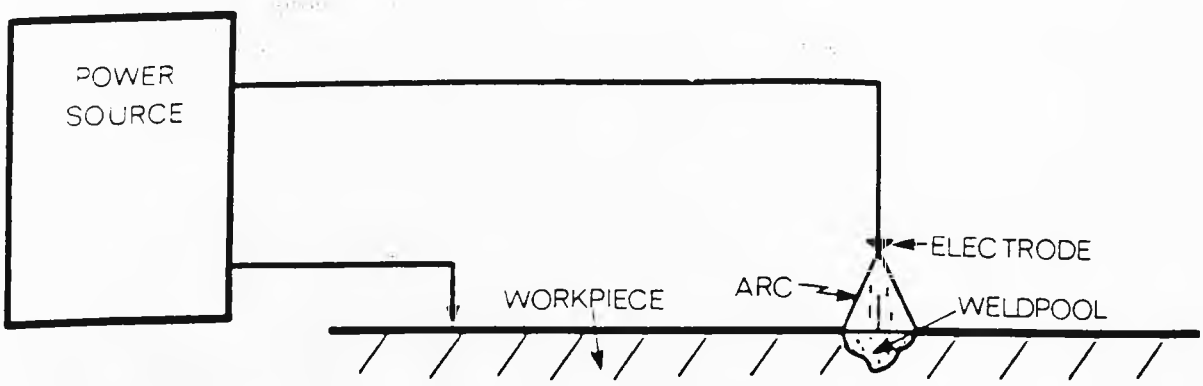


Figure 1·1 WELD-POOL PENETRATION

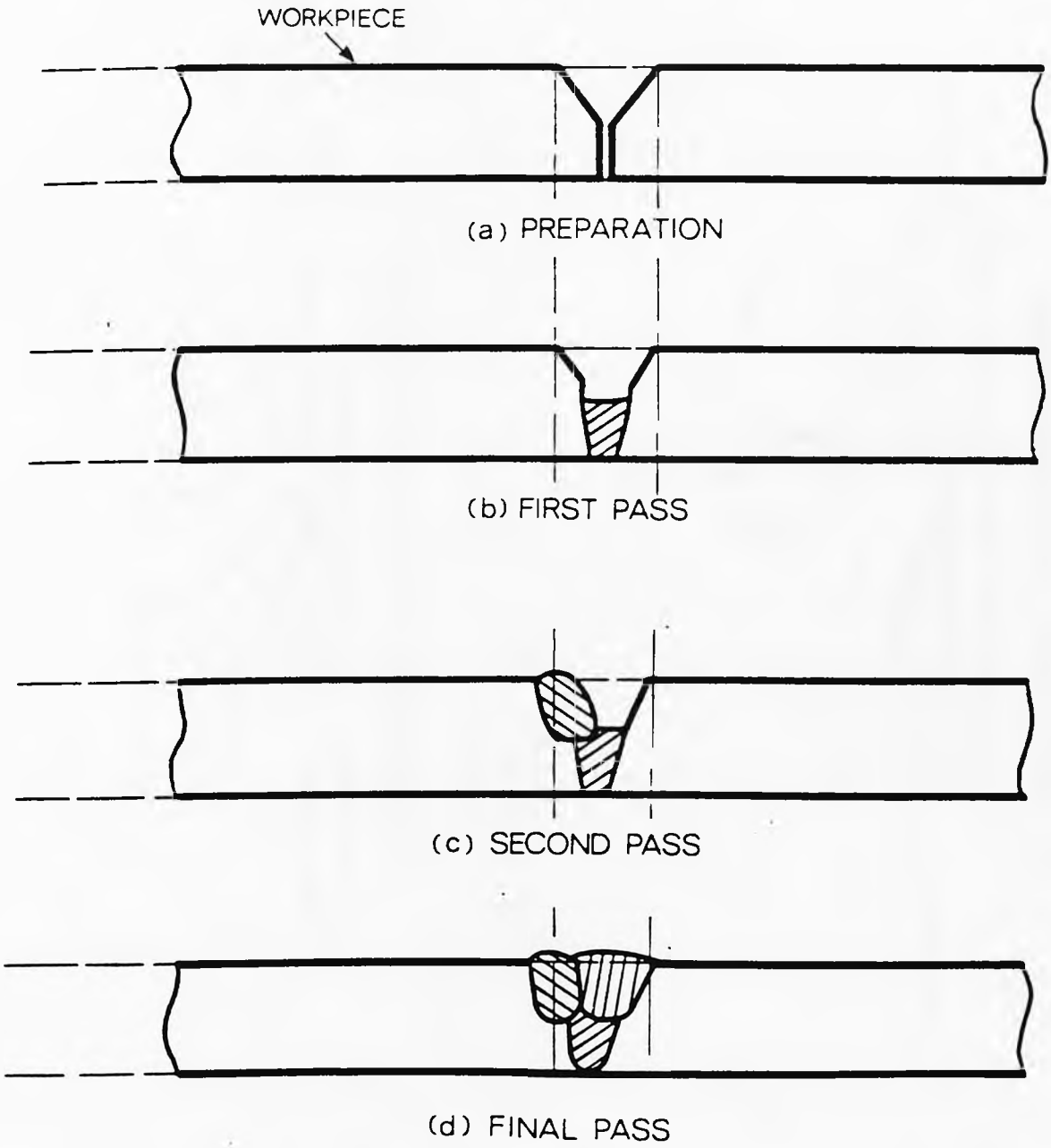
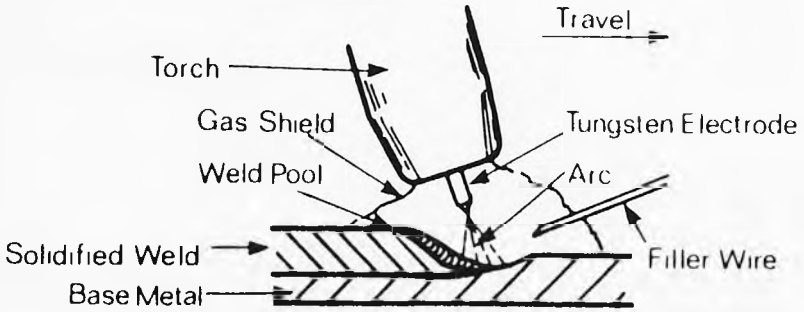
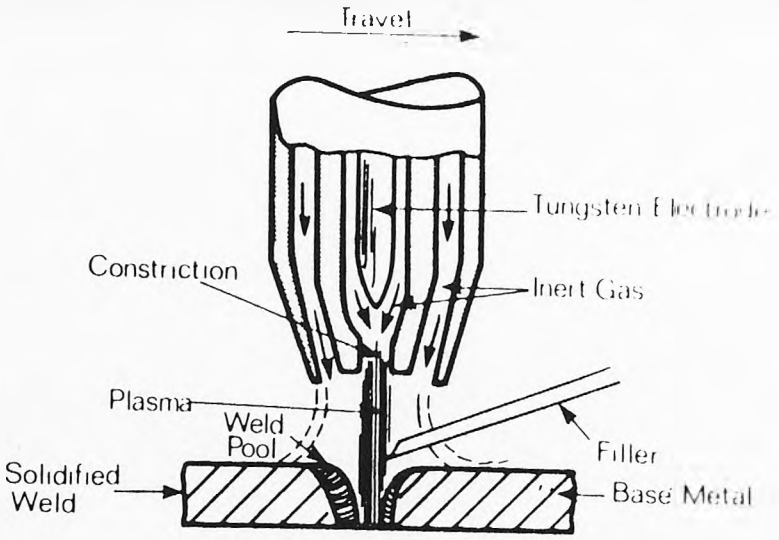


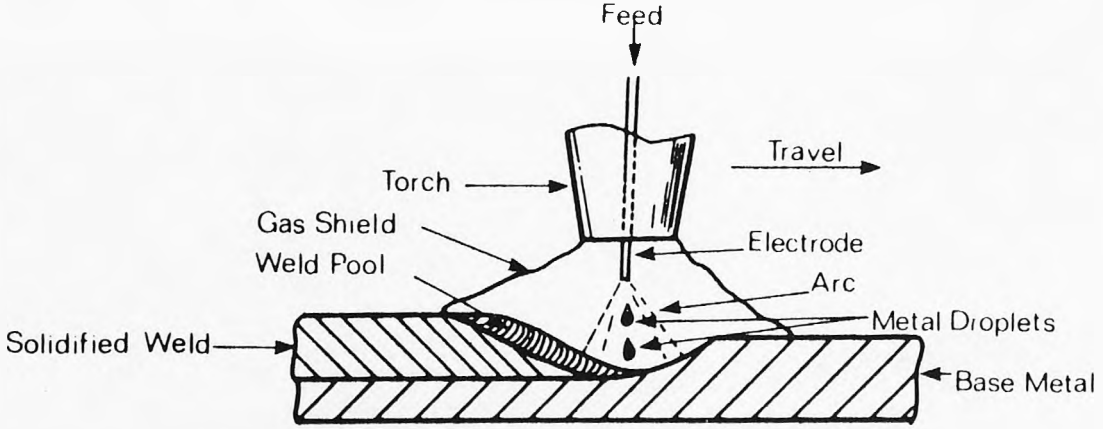
Figure 1·2 WELDING OF 'THICKER' METAL



(b) TUNGSTEN INERT GAS TIG WELDING

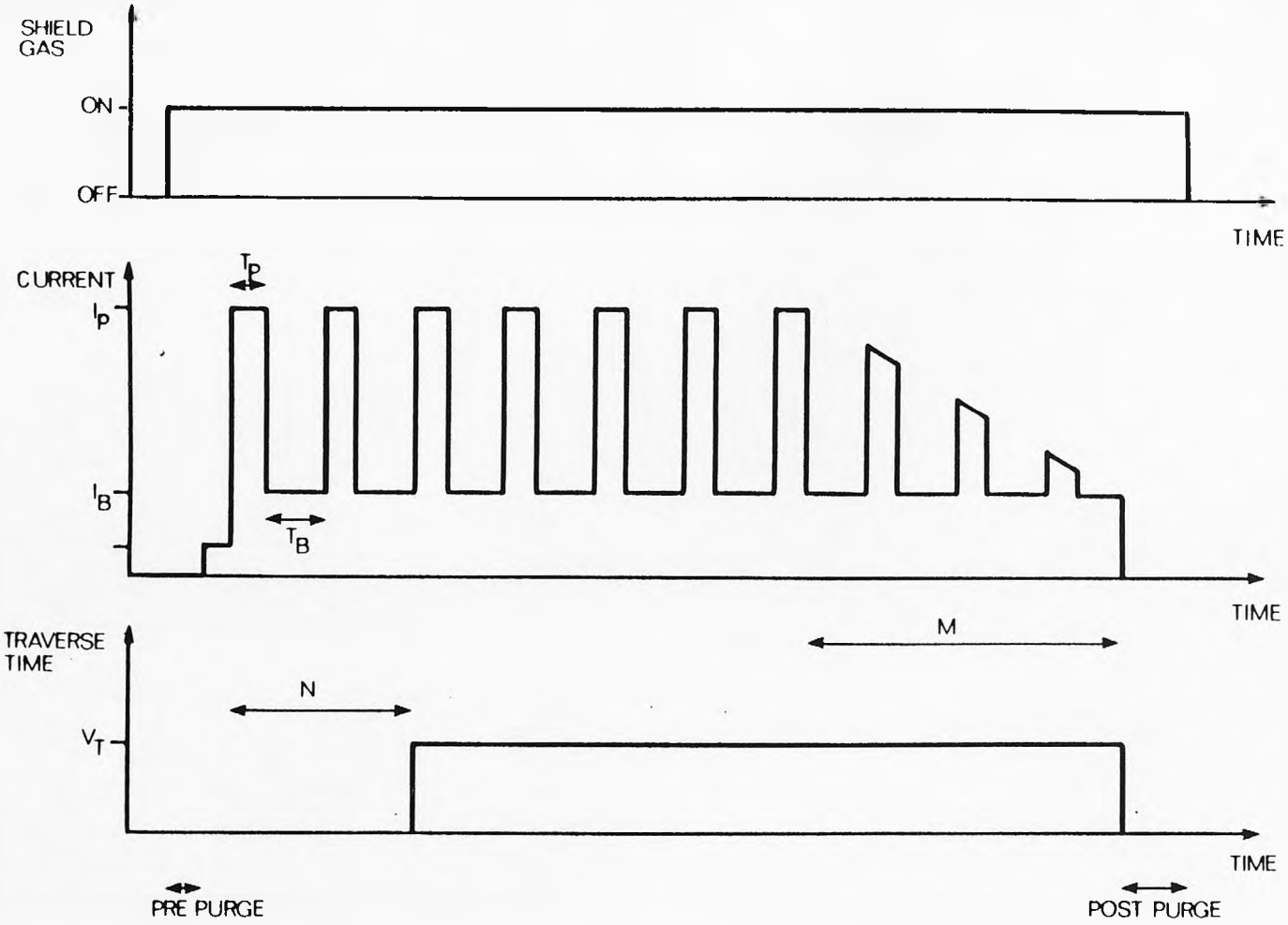


(c) PLASMA WELDING



(a) METAL INERT GAS MIG WELDING

Figure 1·3 ARC WELDING PROCESSES



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 1·4 PULSED TIG PARAMETERS

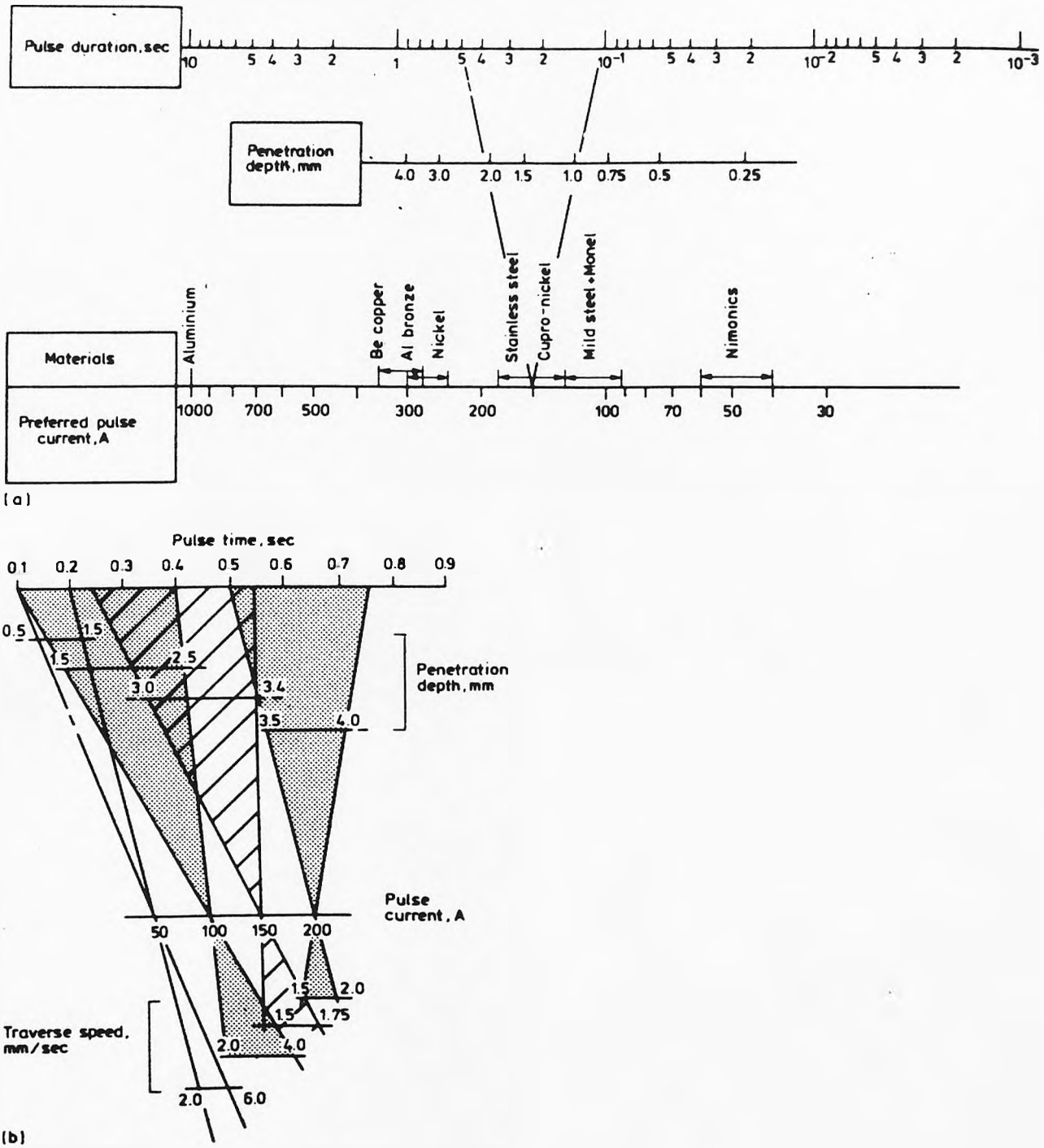
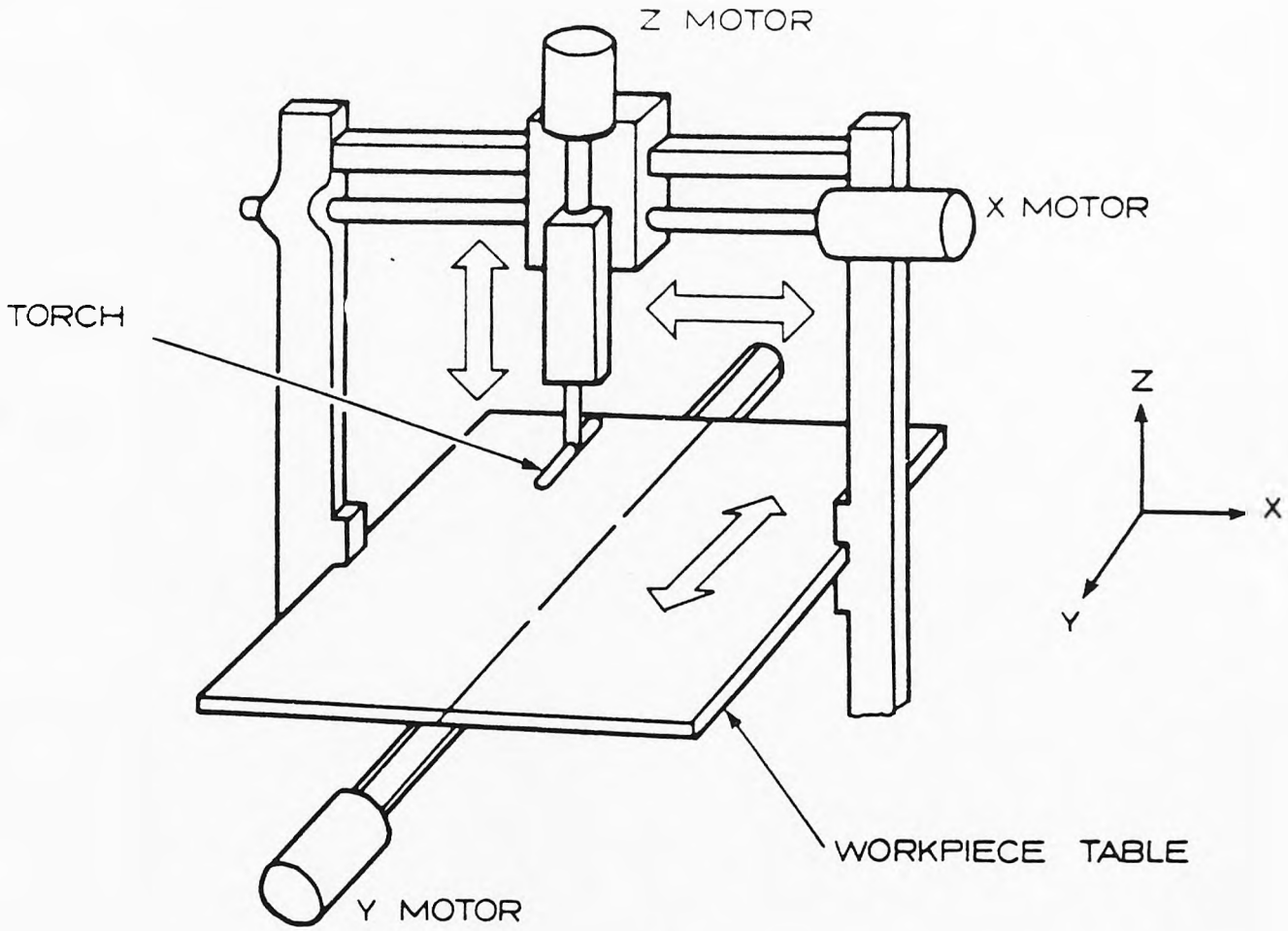
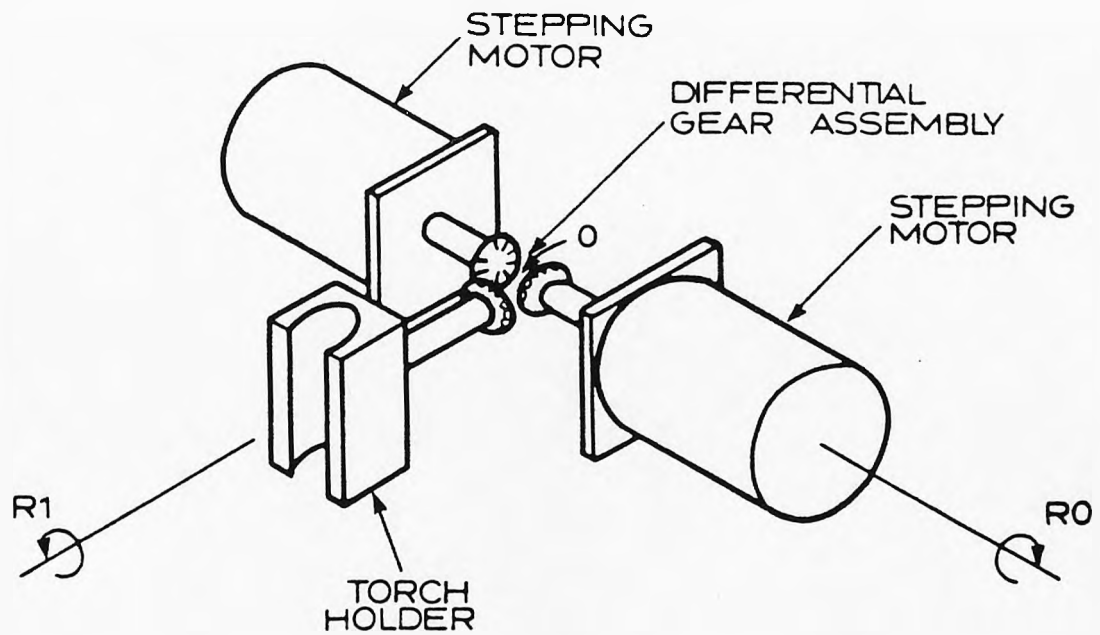


FIG 1.5 PULSED TIG NOMOGRAPHS TO AID PULSE CURRENT SELECTION



(a)



(b)

Figure. 1·6 THE TIM-3 ROBOT TIG MACHINE.

(a) TORCH AND WORKPIECE TABLE ARRANGEMENT.
(b) WRIST ACTION FOR PREPOSITIONING OF THE TORCH.

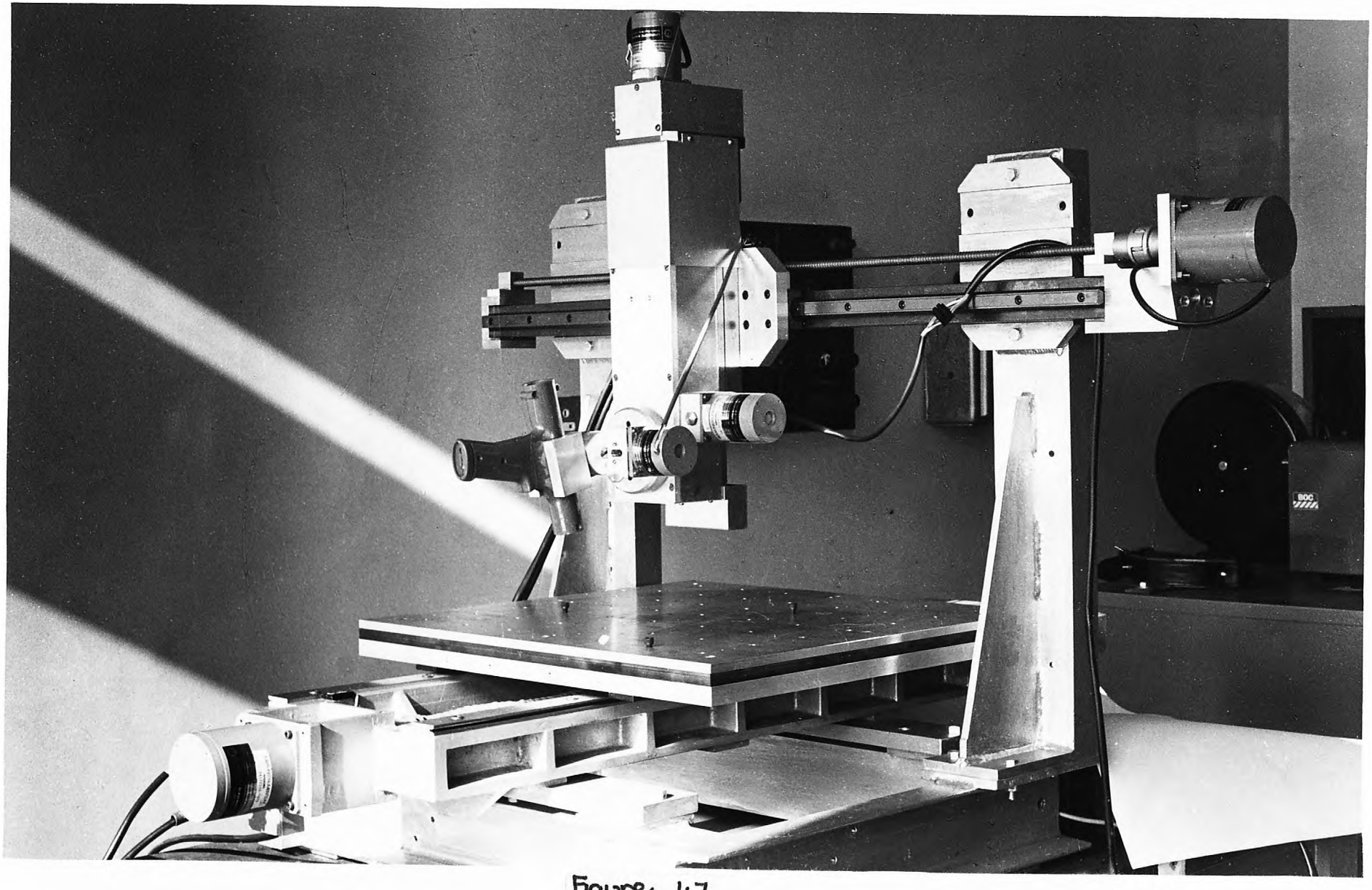


Figure 1.7

CHAPTER 2.

SUPERWELDER HARDWARE

2.1 Hardware Overview.

The Superwelder hardware (Figure 2.1) is based round the industrial Intel single board computer system (iSBC). The hardware is a stand alone unit. It needs to be capable of driving the robot, the welding power supply unit and have interface capability to multiple sensor units.

The Intel iSBC system is based around the iAPX 8086 processor. The iSBC system itself consists of a vast array of boards from which a system may be built to suit the application. The processor boards are all capable of running the iRMX operating software (Chapter 3) which is used to build the operating system. The application engineer needs little programming knowledge of the iSBC boards as the iRMX operating system provides software drivers for all the iSBC processor and peripheral boards.

2.2 Chassis Hardware.

The whole system is housed in an 18" RETMA rack (figure 2.2). It consists of the computer chassis, the dual 8" floppy disk drives and two termination boards, (one analogue, one digital) for input and output to the welding hardware (iCS 910 & iCS 920).

On the top of the RETMA rack is placed the terminal, a DEC VT100. The terminal is connected directly to the USART port on the iSBX 86/30 processor board.

The computer chassis [10] is an industrial iCS 80 cardcage. It contains a large switch mode power supply

the iSBC 640 [11], giving the following voltage lines ;-

Voltage	Current (Max)
+12V	6.8 A
+5V	45 A
-5V	3.2 A
-12V	3.2 A

The chassis also contains two iSBC card holders (iSBC 604) [12], which can hold four cards each onto the Multibus backplane. These backplanes contain the full Intel Multibus (shown in figure 2.9) containing the bus priority lines which can be hardwired in either serial or parallel priority mode. The bus priority schemes are for when more than one bus master is present on the Multibus, with only two bus masters (discussed later) the priority is wired in serial mode.

The backplanes only contain four main iSBC boards.

1) iSBC 86/30 single board computer.

    i) iSBC 337 numeric data processor.
    ii) iSBX 328 analogue output multimodule.

2) iSBC 208 floppy disk controller.

3) iSBC 012B memory board.

4) Custom built stepper motor controller card.

Two card holders are required though because the multimodule boards are attached piggy back upon the iSBC 86/30, taking up too much room to allow the four main boards to fit into one card holder.

2.3 iSBC 86/30 Single Board Computer.

The iSBC 86/30 [13] (figure 2.3) contains the main iAPX 8086 microprocessor with the relevant control hardware and 128K bytes of memory. All the on board hardware is taken care of and controlled by the iRMX

software. It contains one programmable interrupt controller (PIC) an 8259A which has eight interrupt input lines linkable from several possible sources. The iRMX software dictates how most of these are connected (Figure 2.4), but interrupt four is used by the application software and connected to the clock three on the programmable interval timer (PIT) an 8253.

This PIT has three clocks, the first of which is used as the system interrupt clock and is connected to interrupt one, as is required by the iRMX operating system. The second generates the rate clock for the board's programmable communications interface (PCI), which acts directly as the terminal driver for the system. The PCI is an 8251A, which in turn has its receive ready and transmit ready connected to interrupts six and seven respectively. The board also contains a programmable parallel interface in the form of a 8255A which contains three ports of eight lines each. These 24 lines can be configured in many different configurations of input and output, the form used by the application is shown in figure 2.5.

2.3.1 iSBC 337 Numeric Data Processor.

The iSBC 337 [14] is a small card containing a single chip, the 8087 real maths processor. It is placed next to the 8086 processor on the iSBC 86/30 board. The 8087 intercepts real maths instuctions from the 8086 bus and executes them, working in parallel with the 8086.

The 8087 is capable of 64 bit precision real maths at

high speed. The 8087 can perform a real number multiplication in 27 μs compared to the 8086's emulation execution time of 2,100 μS [15]. Interrupt zero is used by the 8087 to notify the 8086 when it has finished an instruction. This is to prevent the 8086 from using variables from the memory before the 8087 has placed them there. The 8087's instruction set includes such operators as square root (FSQRT), partial arctangent (FPATAN) and integer multiplication (FIMUL).

2.3.2 iSBX 328 Analogue Output Module.

The mother board also contains two iSBX connectors (figure 2.6), which carry a simple bus with the address bus already decoded into chip select lines. One of these was used for the development of the three axis controller described later in this chapter (section 2.5). The other has an iSBX 328 [16] digital to analogue (D/A) multimodule present on it (fig. 2.1). This digital to analogue module has eight output channels which may be configured in bipolar output mode (-5V to +5V) or unipolar output mode (0 to +5V). The later is used by the application system. At present only channel one is used to supply the voltage reference to the welding current power supply but the other seven channels may be used in future to drive servo-motors.

2.3.3 The Termination Panels.

The analogue termination panel is an iCS 910 [17] containing 16 differential outputs or 32 single ended outputs. At present only one differential output is used to drive the analogue output from the iSBX 328 to the

welding power supply. The digital termination and conditioning panel is an iCS 920 [18] which has 24 configurable input/output digital lines. This is used for a variety of purposes, driving the key pendant and other assorted output lines to control external hardware as shown in figure 2.5.

2.4 iSBC 208 Floppy Disk Controller.

The iSBC 208 floppy disk controller [19] in the application system drives dual single sided Shugart 8" disk drives. The board though may be configured to drive 5.25" drives. Using this controller virtually any sort of disk drives may be used. It is envisaged that in a final system only one drive will be required with the system software being booted off one disk and then changed for the data storage disk. Alternatively the system software may be blown into EPROM and placed on the mother board. The RMX basic input output system (BIOS) includes software drivers for the iSBC 208 and so no programming knowledge is required of the board.

2.5 iSBC 012B Memory Board.

The iSBC 012B Memory board [20] used during development of the application system contains 512K bytes of RAM. The iSBC 86/30 board also contains 128K bytes giving a total of 640K bytes. This amount of memory was needed during development so that debugging code and system debug software could be included with the application software. The final memory requirement though is only around 256K bytes so this board may be removed and

replaced with a smaller board of 128K bytes. If the final application software is blown into EPROM instead of being resident in RAM this memory board can be removed altogether as the mother board's 128K bytes will suffice.

2.6 Custom Built Boards.

The custom built hardware was developed in two stages. Firstly a simple three axis driver was built purely for software testing and development, then later a full and more complex six axes driver was developed. Both will be described for reasons of completeness.

The custom built boards are to control the stepper motor drivers [5] on the TIM robot. The drivers require a digital line to provide direction control and a clock line. For each pulse on the clock line the stepper motors take one step (400 steps per revolution). The motor direction lines are the output lines of a D type latch.

To provide the clock lines, 8253 programmable interval timers (PIT's) are used [21]. These devices contain three timers, which are programmable in many modes. The application uses mode 2, the rate generation mode. In this mode the output of the timer goes low for one period (T) (approx 7uS) corresponding to the input clock frequency (147kHz) and then high for a time interval equal to the product of the clock period (T) and the count the timer has been loaded with. By varying the timer's count, the rate at which the low pulses are output changes, which in turn, varies the motor speed. Each timer also has a gate input. This input stops and starts the timers. The gates are supplied from ports and are referred

to as the motor enables.

2.6.1 Three Axis Stepper Motor Controller.

The 86/30 processor boards contain two iSBX expansion sockets . One of these is used by the iSBX 328 multimodule, the other is free. The iSBX expansion bus supplies all the control lines and a fully buffered data bus (MD0-MDF) (Figure 2.6) with two decoded chip select lines (MCS1/,MCS0/) and the lowest three address lines (MA0-MA2). For the three axis system a single 8253 PIT is required to enable three motors to be driven. A single eight bit port (74LS374) is also required to supply three direction lines to the motor drivers and three enable lines to the clock gates on the 8253. This leaves two bits of the port unused. With just using these two main chips we are able to use the iSBX bus.

Refering to figure 2.7, the port used is a 74LS374 octal D-type latch clocked by NOR-ing the IOWRT/ and MCS1/ lines and inverting this resultant line with another NOR gate as the 74LS374 clocks on the rising edge. The 8253 is selected directly using MCS0/ and is also supplied with the input/output read and write lines and the lower two address lines. The gate lines are supplied by the lowest three lines from the port, the next three lines being used as the output to the stepper motor driver direction lines. The 8253's three timers then need clock lines, these are supplied by the iSBX bus' clock (MCLK) divided down by a 74LS393 dual 4 bit binary counter to 147KHz. Using this design a small four chip circuit board is produced fitting

neatly on the back of the 86/30 mother board.

2.6.2 Six Axis Stepper Motor Controller.

Unlike the previous design, two 8253 programmable clocks are required to provide clock lines for the six motor drivers. This would use both the iSBX bus' chip select lines, using the 3 axes design, leaving no select lines for the ports. The six axes controller (figures 2.8 - 2.11) must therefore be interfaced to the main Intel Multibus shown in figure 2.12. To drive the six axis, six gate lines and six direction lines are required, therefore two 74LS374's are used.

Refering to figure 2.11, the data is buffered by two 8226 bidirectional buffers, IC's 18 & 19. The system data bus is connected straight to the data input pins. The board's data bus is taken from the chip's buffered data input and output pins which are connected together. This bus is connected to the two 8253's and the two 74LS374's, IC's 1,2,7 & 3 respectively. The direction control (DIEN/) for IC18 and IC19 is controlled by the I/O read command (IORD/) line via two inverters on IC14. If the read command is asserted by the bus master, the data buffer's driver mode is selected. At all other times the buffer's receiver circuits are enabled.

2.6.3 Board Address Decode Logic.

The board select is produced by a single decode chip IC16, a thirteen input NAND gate. Inverting the tenth address line with an inverter from IC17 (74LS04) produces a board select output (BENA/) in the address range 400H-4FFH. This line is fed to IC11 a 74LS138 three line

decoder which with address lines /A3-/A5 all inverted by IC17 produces four chip select lines (CS/). Two of these chip selects are put straight to the 8253's, the other two are NANDed by IC6 with the board write line (/BIOW) to supply the two 74LS374 clock lines.

The board select line is also combined with the IOWC/ and IORD/ lines using IC's 14 & 15 to produce the board enable line (BEN/) of the logic ;-

BEN/ = (BENA/ . IORD/) + (BENA/ . IOWC/).

The logic also provides a transfer acknowledge response, XACK/, to notify the bus master that write data provided by the bus master has been accepted or that read data requested is available on the system bus. Strictly the read circuitry is not required as only writes are ever executed but it is included to incorporate any future expansion.

Timing for the XACK/ signal is shown in figure 2.13. IC8, an 8224 crystal driver, is coupled with a 22.1184MHz crystal producing a square wave of approximately 45nS period at its output. Two inverters from IC5 then square up the wave and feed it to IC9 a high speed hex D-type flip-flop (74LS174) that is wired to function as a shift register. Its purpose is to provide a delay in the generation of XACK/ to allow sufficient time for I/O port accessing, which in the worst case is 400nS for the 8253's. The positive edges of OSC/ cause IC9 to first shift through a series of 1's then a series of 0's. The shift register outputs are decoded in such a manner that

the lower half of IC10's clock input is armed after approximately 90uS. 410uS later it goes high again clocking the XACK/ latch set. The Q side of the lower half of IC10 (74LS74) is inverted and driven on to the system bus by IC13 (74LS125). This driver is enabled by the same gate that provides the chip select for the data buffers.

The latches of IC10 remain set until the master removes either IOWC/ or IORD/. At this point /BEN will go low clearing IC9 and IC10. Since the I/O requests are not synchronous with the OSC/ clock. the leading edge timing of XACK/ includes a +45nS tolerance with respect to the read or write command. The second half of IC10 provides a delay for the IOWC/ command so as to allow the board's data bus to settle before the command arrives.

All outputs, both clocks and direction lines to the stepper motor drivers, are opto-isolated to prevent feedback of noise to the cicuitry and to provide open collector outputs which the CMOS inputs of drivers require (0-12V positive logic).

2.7 The Hand Held Pendant.

The other custom built part of the hardware is a crosswire matrix hand held key pendant shown in figure 2.14. This matrix has four input lines and six output lines giving a possible 24 keys. The input/output is from the iSBC 86/30 edge connector shown in figure 2.5. The iCS 920 Digital Termination Panel is used to opto-isolate the lines as shown in figure 2.1.

The keys are all standard push to make (RS number 338-765). The matrix of keys is a standard hand held

pendant for movement of up to 8 axes in both directions and several special keys for path input. These special keys are used to indicate start of weld points and adjust the robot speed etc.

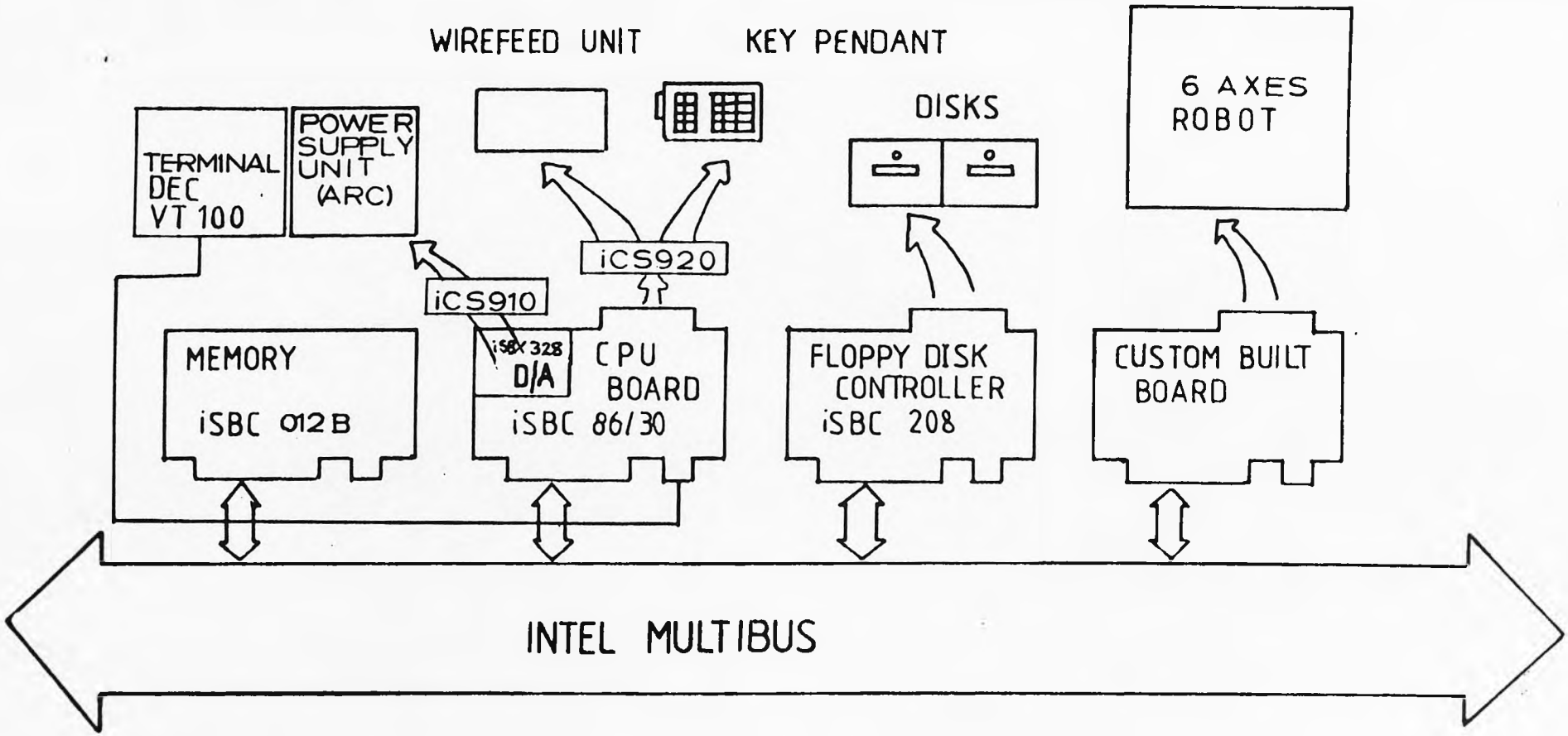


FIG. 2.1 SYSTEM HARDWARE.

Figure 2·2

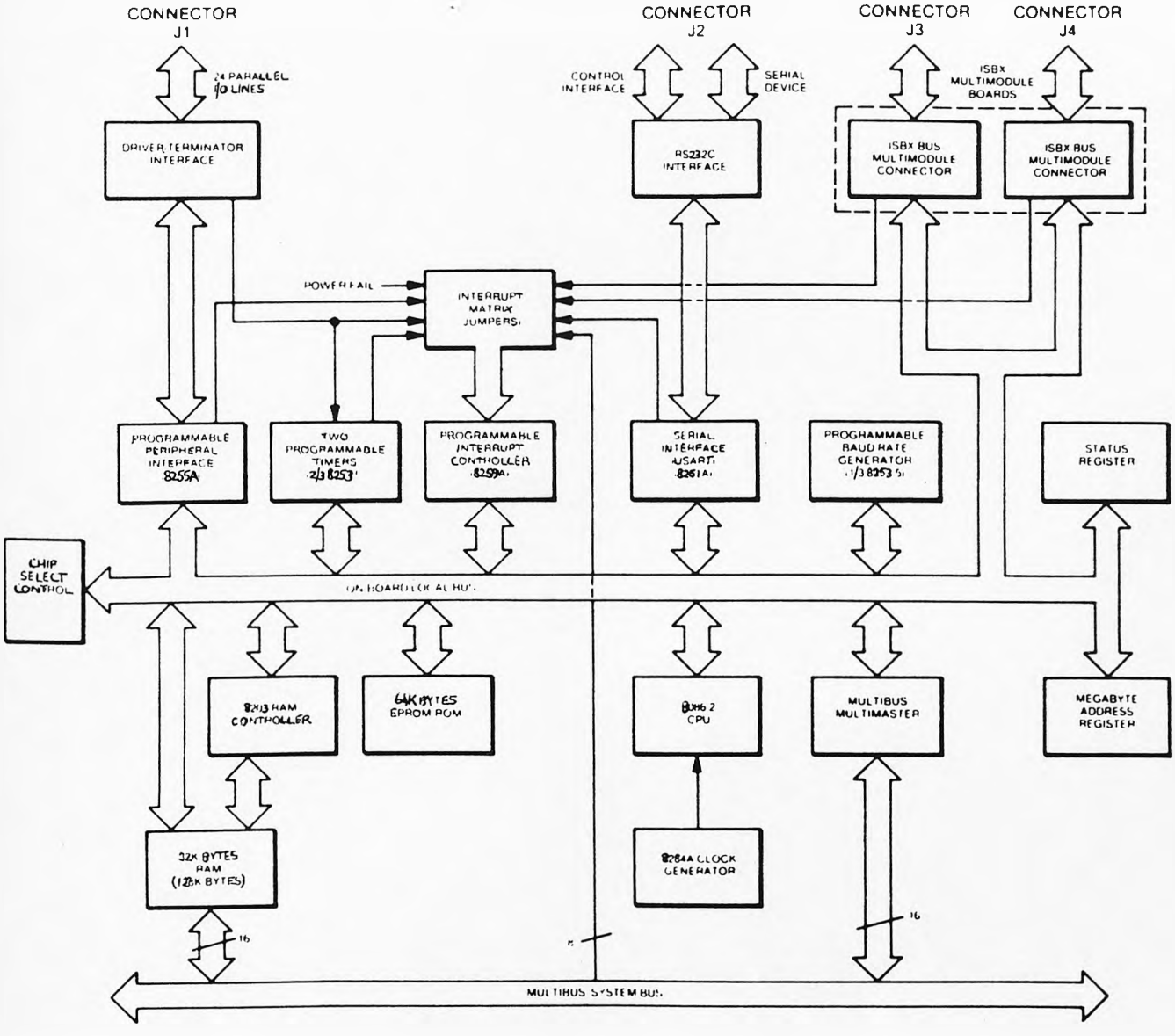


FIG 2.3 86/30 BLOCK DIAGRAM.

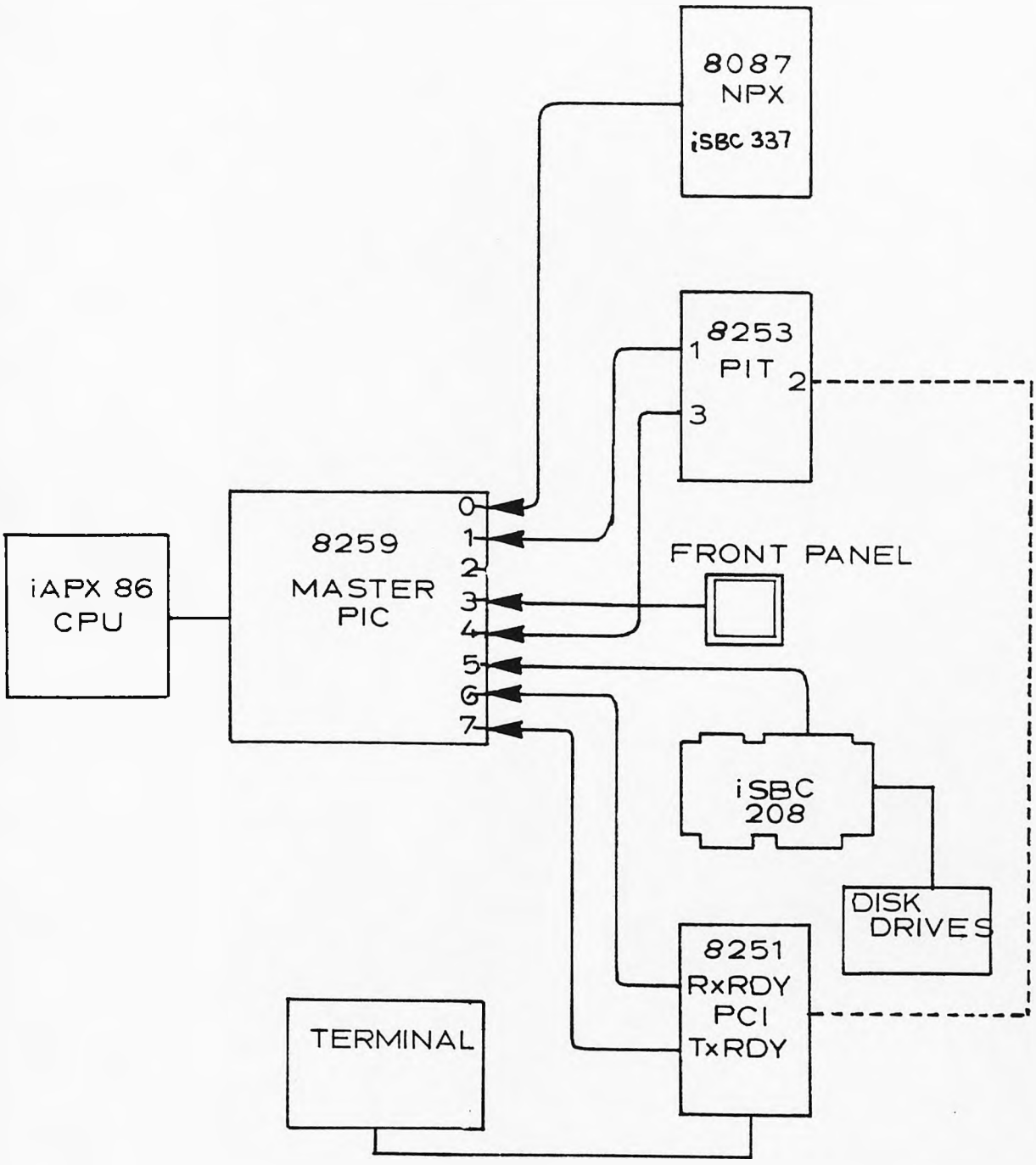


FIG 2.4 SYSTEM INTERRUPTS

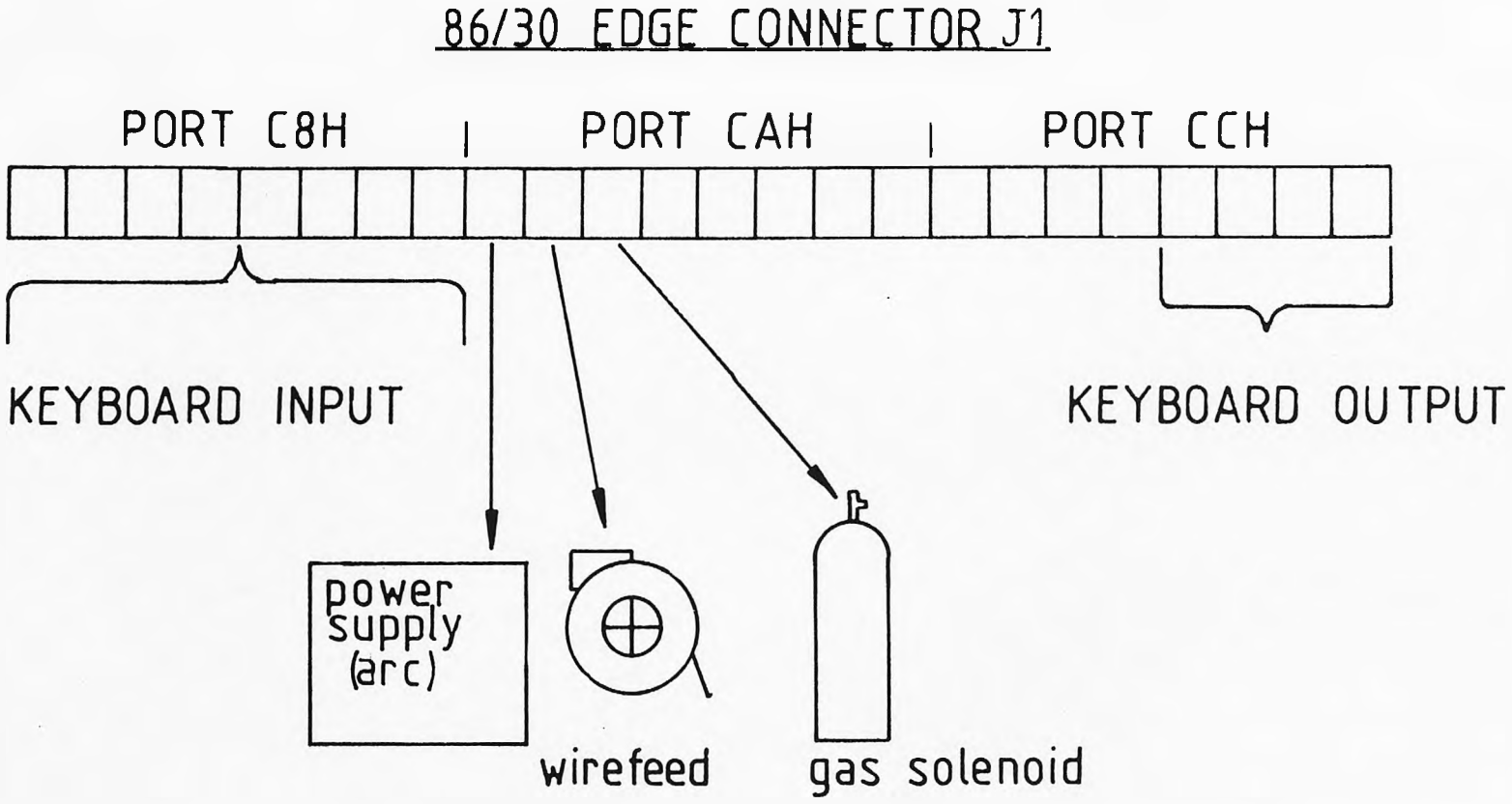


FIG 2.5 CONTROL PORT CONFIGURATION

Pin	Mnemonic	Description	Pin	Mnemonic	Description
1	+12V	+12 volts	2	-12V	-12 volts
3	GND	Ground	4	+5V	+5 volts
5	MRESET	Module Reset	6	MCLK	Module Clock
7	MA2	Address Bit 2	8	MPST/	Module Present
9	MA1	Address Bit 1	10		Reserved
11	MA0	Address Bit 0	12	MINTR1	Module Interrupt 1
13	IOWRT/	I/O Write Command	14	MINTR0	Module Interrupt 2
15	IORD/	I/O Read Command	16	MWAIT/	Wait-state Request
17	GND	Ground	18	+5V	+5 volts
19	MD7	Module Data Bit 7	20	MCS1/	Module Chip Select 1
21	MD6	Module Data Bit 6	22	MCS0/	Module Chip Select 0
23	MD5	Module Data Bit 5	24		Reserved
25	MD4	Module Data Bit 4	26		Reserved
27	MD3	Module Data Bit 3	28	OPT1	Option Line 1
29	MD2	Module Data Bit 2	30	OPT0	Option Line 0
31	MD1	Module Data Bit 1	32		Reserved
33	MD0	Module Data Bit 0	34		Reserved
35	GND	Ground	36	+5V	+5 volts
37	MDE	Module Data Bit E	38	MDF	Module Data Bit F
39	MDC	Module Data Bit C	40	MDD	Module Data Bit D
41	MDA	Module Data Bit A	42	MDB	Module Data Bit B
43	MD8	Module Data Bit 8	44	MD9	Module Data Bit 9

FIG. 2.6 iSBX INTERFACE BUS

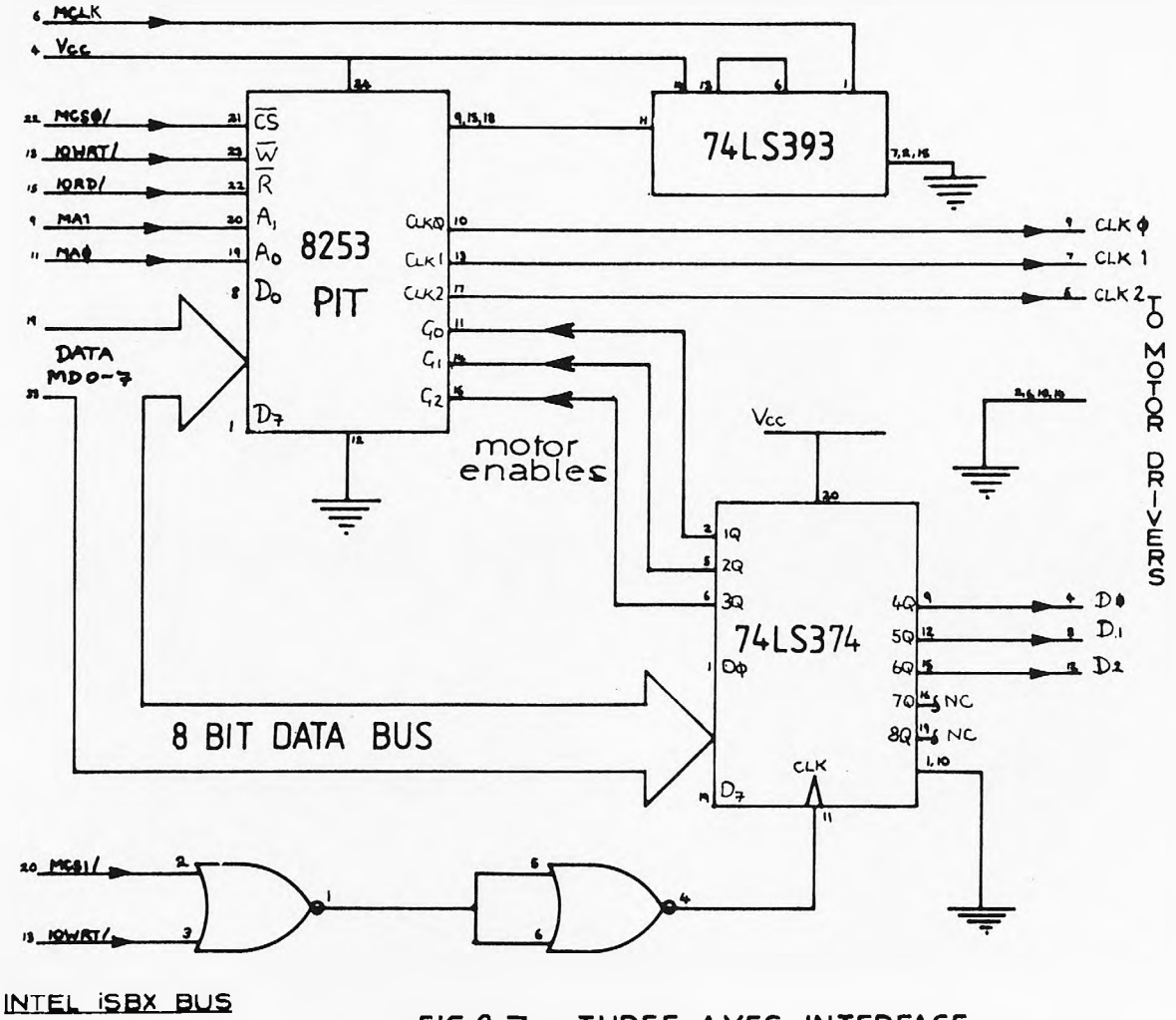


FIG.2..7 THREE AXES INTERFACE

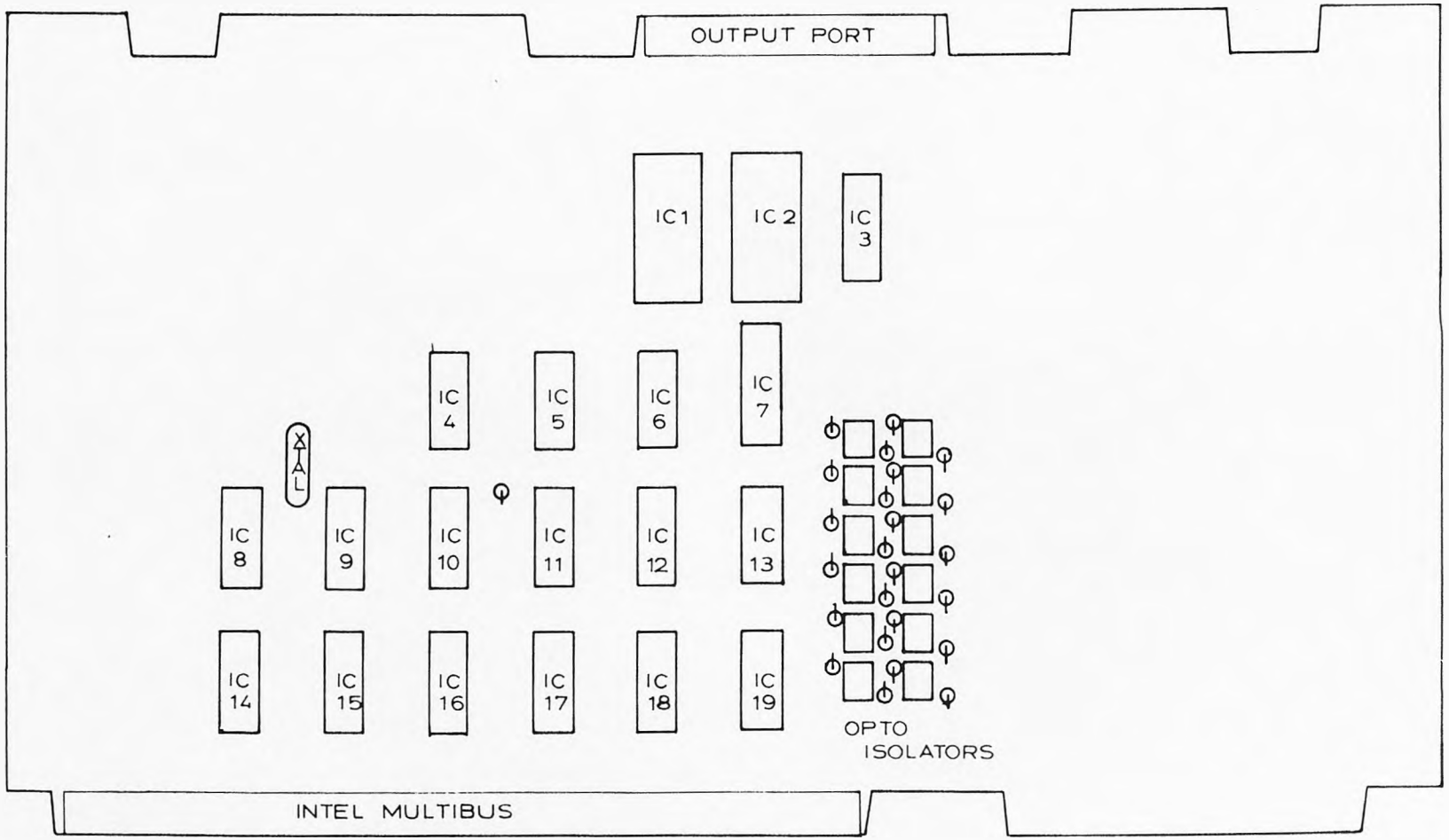


FIG 2.8 INTERFACE BOARD LAYOUT

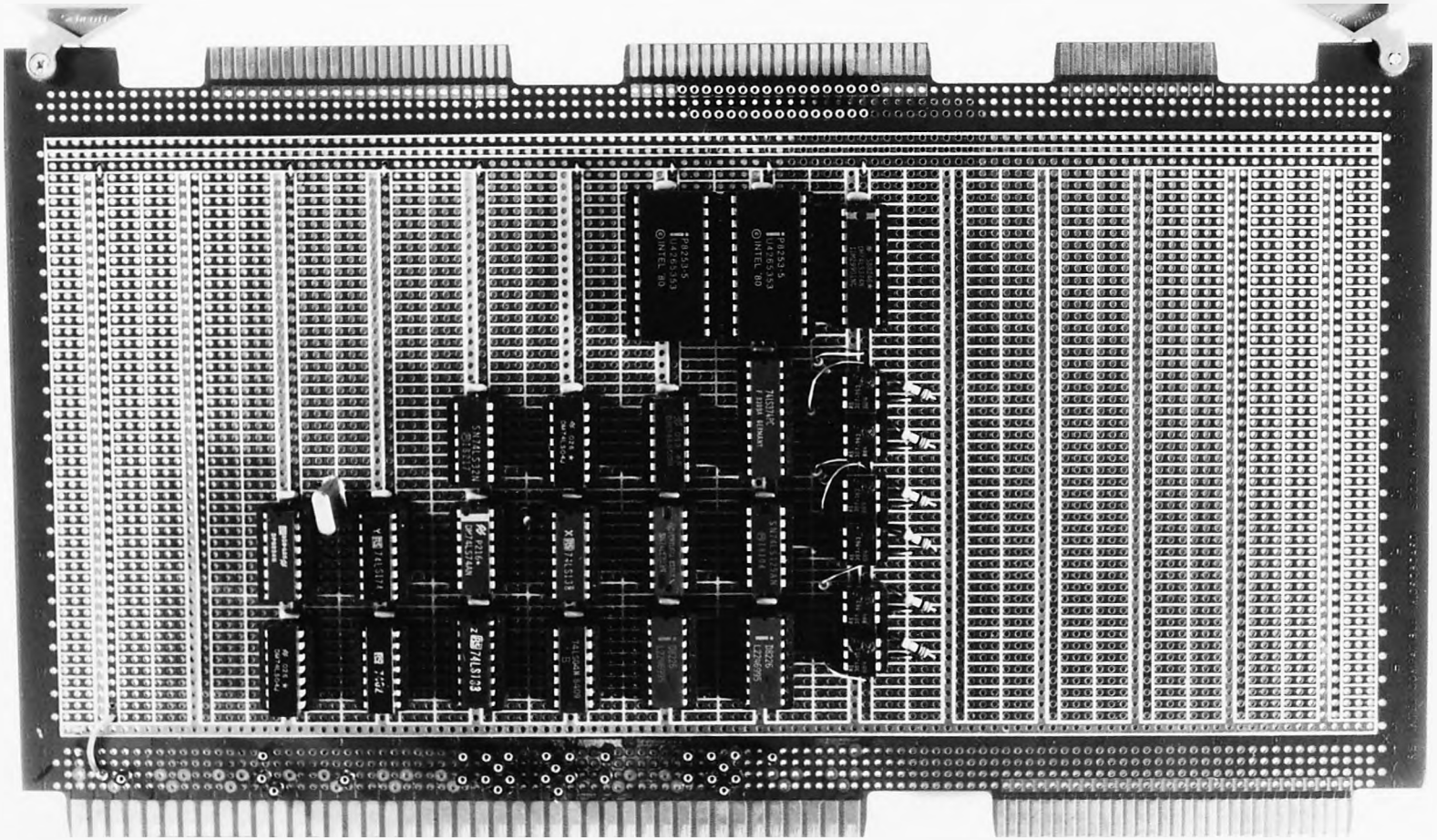


Figure 2.9

REFERENCE	TYPE
IC 1	8253
IC 2	8253
IC 3	74LS374
IC 4	74LS32
IC 5	74LS04
IC 6	74LS02
IC 7	74LS374
IC 8	8224
IC 9	74LS174
IC 10	74LS74
IC 11	74LS138
IC 12	74LS393
IC 13	74LS125
IC 14	74LS04
IC 15	74LS02
IC 16	74LS133
IC 17	74LS04
IC 18	8226
IC 19	8226
OPTO ISOLATORS	ALL TIL 113

FIG 2.10 IC REFERENCE TABLE

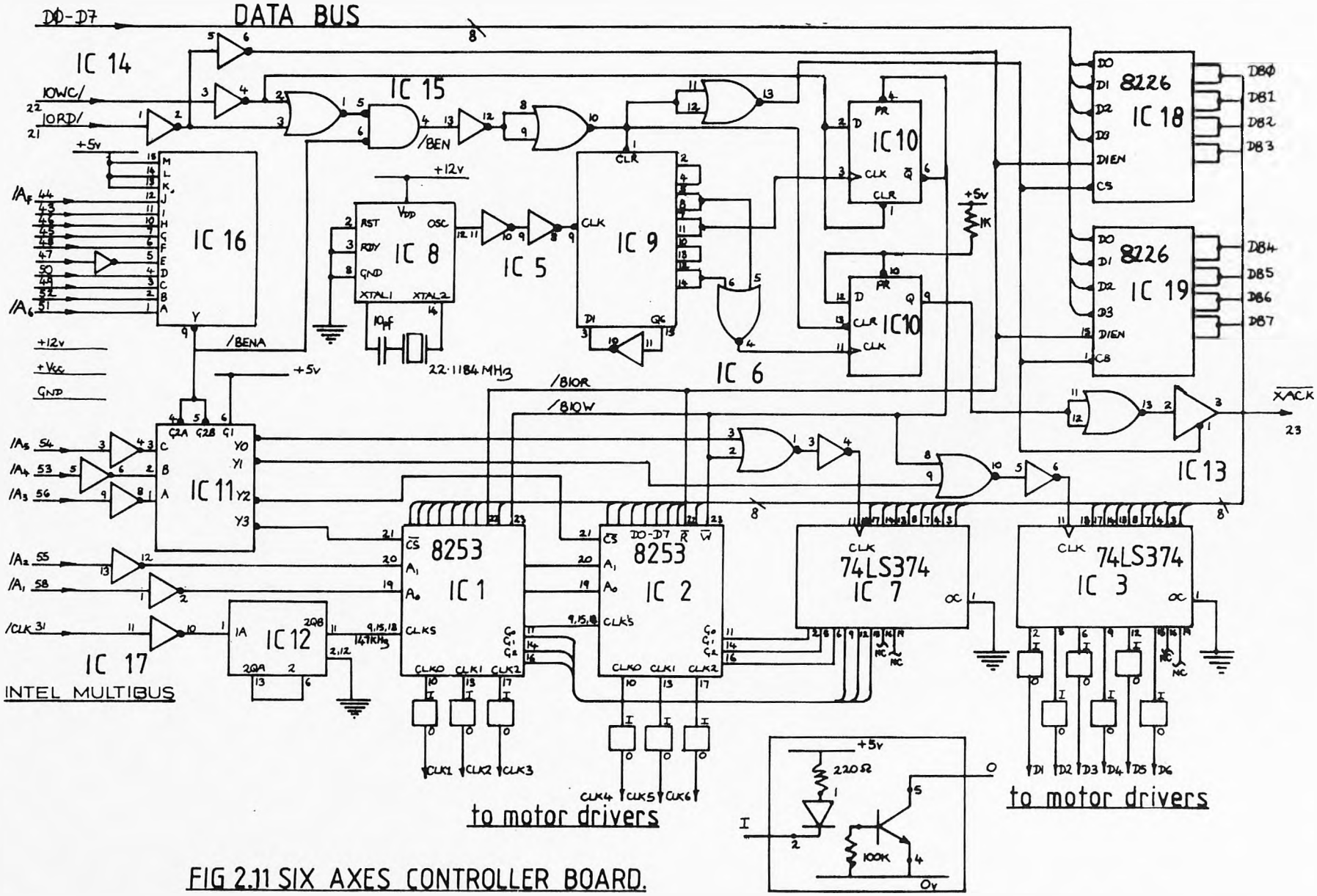
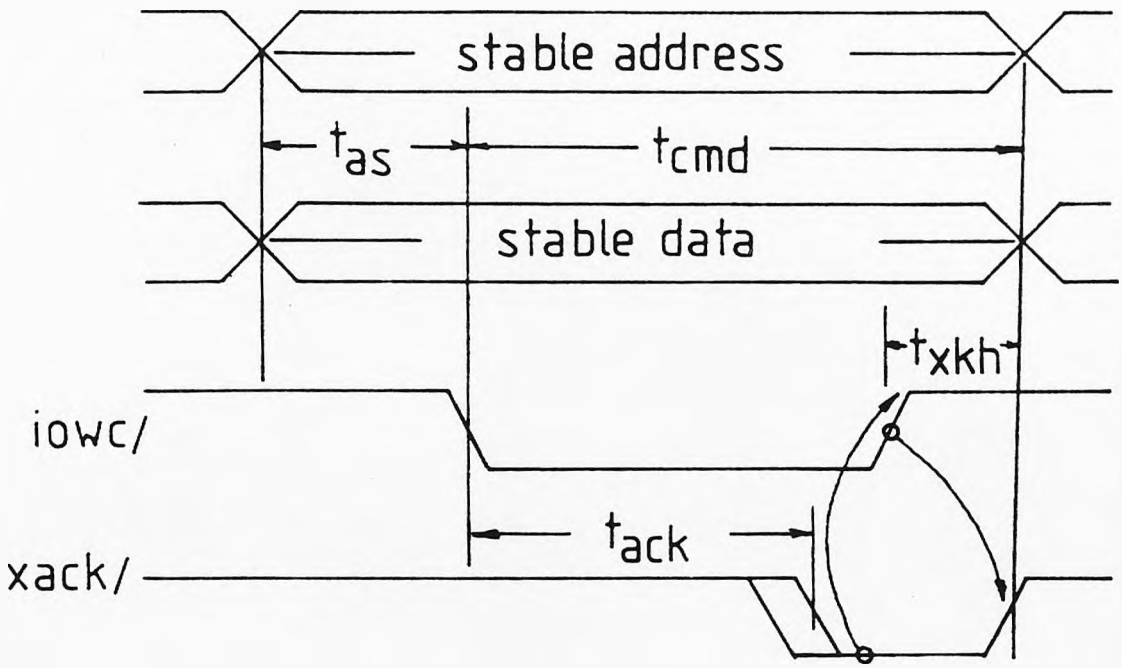


FIG 2.11 SIX AXES CONTROLLER BOARD.

	(COMPONENT SIDE)			(CIRCUIT SIDE)		
	PIN*	MNEMONIC	DESCRIPTION	PIN*	MNEMONIC	DESCRIPTION
POWER SUPPLIES	1	GND	Signal GND	2	GND	Signal GND
	3	+5V	+5 Vdc	4	+5V	+5 Vdc
	5	+5V	+5 Vdc	6	+5V	+5 Vdc
	7	+12V	+12 Vdc	8	+12V	+12 Vdc
	9		Reserved	10		Reserved
	11	GND	Signal GND	12	GND	Signal GND
BUS CONTROLS	13	BCLK/	Bus Clock	14	INIT/	Initalize
	15	BPRN/	Bus Priority In	16	BPRO/	Bus Priority Out
	17	BUSY/	Bus Busy	18	BREQ/	Bus Request
	19	MRDC/	Mem Read Cmd	20	MWTC/	Mem Write Cmd
	21	IORC/	I/O Read Cmd	22	IOWC/	I/O Write Cmd
	23	XACK/	XFER Acknowledge	24	INH1/	Inhibit 1 Disable RAM
BUS CONTROLS AND ADDRESS	25	LOCK/	Bus Lock	26		Reserved
	27	BHEN/	Byte High Enable	28	AD10/	
	29	CBRQ/	Common Bus Request	30	AD11/	Address
	31	CCLK/	Constant Clk	32	AD12/	Bus
	33	INTA	Interrupt Ack	34	AD13/	
INTERRUPTS	35	INT6/	Parallel	36	INT7/	Parallel
	37	INT4/	Interrupt	38	INT5/	Interrupt
	39	INT2/	Requests	40	INT3/	Requests
	41	INT0/		42	INT1/	
ADDRESS	43	ADRE/		44	ADRF/	
	45	ADRC/		46	ADRD/	
	47	ADRA/		48	ADRB/	
	49	ADR8/	Address	50	ADR9/	Address
	51	ADR6/	Bus	52	ADR7/	Bus
	53	ADR4/		54	ADR5/	
	55	ADR2		56	ADR3/	
	57	ADR0/		58	ADR1/	
DATA	59	DATE/		60	DATF/	
	61	DATC/		62	DATD/	
	63	DATA/		64	DATB/	
	65	DAT8/	Data	66	DAT9/	Data
	67	DAT6/	Bus	68	DAT7/	Bus
	69	DAT4/		70	DAT5/	
	71	DAT2/		72	DAT3/	
	73	DAT0/		74	DAT1/	
POWER SUPPLIES	75	GND	Signal GND	76	GND	Signal GND
	77		Reserved	78		Reserved
	79	-12V	-12 Vdc	80	-12V	-12 Vdc
	81	+5V	+5 Vdc	82	+5V	+5 Vdc
	83	+5V	+5 Vdc	84	+5V	+5 Vdc
	85	GND	Signal Gnd	86	GND	Signal GND

FIG. 2.12 MULTIBUS CONNECTIONS



t_{ack} acknowledge/access time
t_{as} address setup to command
t_{cmd} command width
t_{xkh} acknowledge hold time

FIG 2.13 MULTIBUS WRITE TIMING

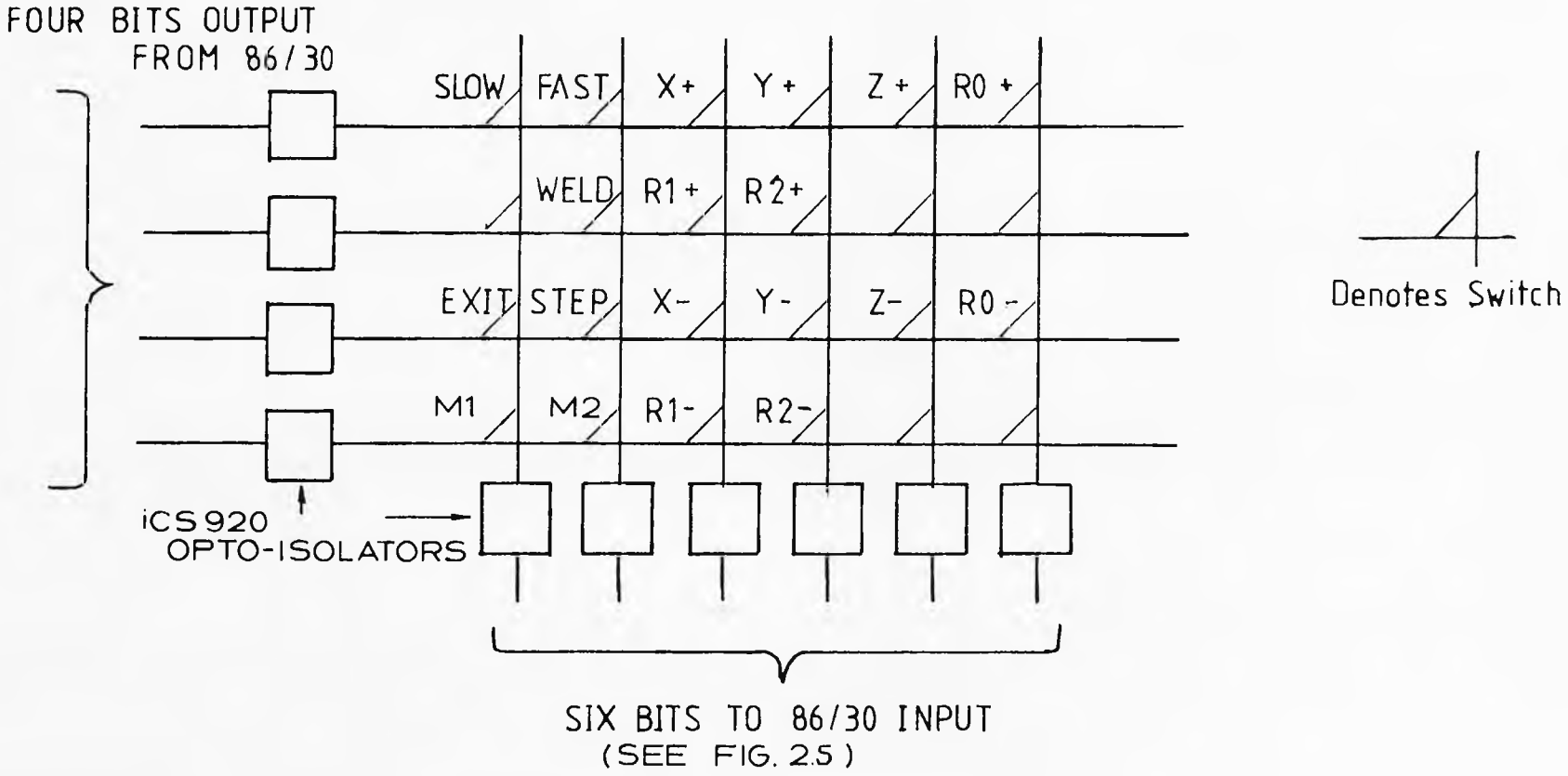


FIG. 2.14 KEYBOARD WIRING MATRIX.

CHAPTER 3.

THE iRMX 86 OPERATING SYSTEM

3.1 Introduction.

The iRMX 86 operating system is supplied as a user configurable software 'kit'. The operating system is divided into several portions, shown in figure 3.1, each with a fully defined and documented purpose. With all the portions configured in a fully functional development system is produced. This 'full' configuration of the operating system was used on an 86/330 hardware package [22] for application development. The operating system in this form can be described as a multiuser, multiprogramming multitasking, system. Multiuser is the ability to interface with more than one user at any given time. This is achieved by driving multiple serial channels controlling the user terminals and having a multitasking enviroment. Multitasking is the ability to run more than one 'task' or 'piece of code' seemingly simultaneously. In reality this is acheived by time sharing the processor, so that each task in trun runs on the processor for a small portion of time (usually in the order of milliseconds). Multiprogramming is the ability to divide the hardware environment into portions so that programs have complete control over these sub-enviroments or 'jobs'. User jobs require at minimum, an area of memory for their tasks to be resident in. Disk drives, printers and other such hardware are shared resources so they are part of special jobs called device drivers with which user programs must

communicate to obtain access to the shared hardware.

The iRMX 86 operating system uses an object-oriented architecture [23]. Objects are predefined units within the operating system such as Jobs, Tasks and Semaphores. The philosophy of such an architecture is that the handling of each type of object is very similar, so that in learning how to handle one type of object makes the handling of all types of objects relatively easy. It is a real time, multitasking development operating system for original end manufacturers (OEM's). The OEM is able to reduce development time and costs by using the iRMX 86 system. It supplies large numbers of features that can used in application systems, such as floppy disk handlers and multitasking ability. The design engineer can then focus his attention on developing the specialized code to complete the application system [23 pp3-1].

The software units are ;-

1)	Nucleus	(NUC)
2)	Basic Input/Output system	(BIOS)
3)	Extended Input/Output System	(EIOS)
4)	Human Interface	(HI)
5)	Universal Development Interface	(UDI)
6)	Terminal Handler	(TH)
7)	System Debugger	(SDB)

3.2 The Nucleus.

The Nucleus [24], as the name suggests, is the heart of the iRMX 86 Operating System. This level must always be included in a system. Its job is to control the computer's environment, timing and objects. The objects are all

controlled by the Nucleus except for the composites which will be described with the BIOS which controls them.

The objects are as follows ;-

1) Jobs
2) Tasks
3) Segments
4) Semaphores
5) Mailboxes
6) Interrupt Handling (not an object but handled by the Nucleus
7) Composites or Connections (dealt with under BIOS)

3.2.1 Jobs.

Jobs are environments for code to run in. They are whole software 'jobs' to be executed, but as well as software they also include memory, objects and hardware. The initial job contains all hardware and software of the computer system, when other jobs are created these resources are divided and shared as they are required. These are called child jobs. The initial software that executes in a job is called the root task and every job must, to exist, contain at least one task. The job has one unique feature, that of an object directory. When a token is created by an operating system call, it may be catalogued in the object directory under an ASCII name. Other tasks,if they then need to use the object, can look it up in the directory using the ASCII name.

3.2.2 Tasks.

Tasks are the actual code that runs. The multitasking

is achieved by allowing tasks to exist in several states (Figure 3.2). When created (by the parent task) a task is given a priority in the range of 0-255 (0 is the highest). If the task is the highest priority it will be the code that is running, provided that it is not suspended or asleep. A task is asleep when it is waiting to receive objects from another task via mailboxes etc (see section 3.2.5). A task can either suspend itself or be suspended by another task, it can never run once suspended until it is unsuspended. Suspension can be done repeatedly and increase the suspension depth, so that it must be repeatedly unsuspended. A task may also be suspended whilst asleep.

With this multitasking feature coding is greatly simplified. Instead of writing a single complex multimodule program to process N events, the programmer can code N tasks, one to take care of each event.

When creating any object the operating system returns a WORD token. This token is unique to the object created and to use the object or refer to it (e.g. suspend a task) the token must be sent to the operating system in the system call which executes the required action.

When a task is created the code must already be loaded and the operating system requires the start address of this code. Other information about the task, such as whether the task will require the iNPX 8087 numeric data processor and the maximum number of objects the task will be allowed to create, is also required.

3.2.3 Segments.

Segments are no more than areas of memory. A segment is created in multiples of 16 bytes. Thus if a task temporarily needs more memory, it creates a segment of the required size, borrowing it from the job's memory pool. In this way memory can be used sparingly. If a task needs large amounts of memory, but only during some of its running time, it may borrow and then pay it back by to the job's memory pool by calling the delete segment call later. Segments may also be used for intertask communication. This is achieved by the use of mailboxes and will be described later in section 3.2.5.

3.2.4 Semaphores.

Semaphores are the operating system flags. A semaphore is an object to which units may be sent and received. When an object is created, the task creating it must specify the maximum units the semaphore can hold. To achieve intertask co-ordination a semaphore is created and a task sleeps at it waiting to receive a number of units. Another task, may then, when it requires the first to run, send the required number of units. Semaphores like ordinary software flags have many uses, such as preventing memory lock out and providing intertask synchronisation.

3.2.5 Mailboxes.

Mailboxes provided a more versatile method of intertask communication. Tokens for other objects may be sent to mailboxes where they are queued until other tasks 'call' at the mailbox to receive them. Alternatively of course tasks may queue at the mailbox waiting for objects

to be 'posted'. As mentioned before, the most powerful method of intertask communication is to create a segment of memory (the token received for a segment is also the base address of the segment itself) and then place the information in the segment and 'post' it to a mailbox. A second task may then receive the segment token from the mailbox and interograte the segments contents. This is the method of communication the operating system itself uses with other tasks. For instance, a task may require information about a stored file and after issuing the required call, will receive a segment of memory containing the information. The receiving task, after examining the information, must delete the segment, returning it to the jobs memory pool.

3.2.6 Interrupt Handling.

Interrupt handling is also taken care of by the nucleus. Through nucleus calls the programmer is able to cause whole tasks (interrupt handlers) to be assigned to interrupts or make the vector point to some internal task code. Interrupts may also be enabled, disabled and acknowledged by a task. If an interrupt handler is assigned to an interrupt then the nucleus is also able to execute calls that wait for specific interrupts and cause interrupts to instigate the interrupt handling task.

All operating system calls dealing with these objects are extremely similar and make the system easy to learn and use.

3.3 The Basic Input/Output System (BIOS).

The BIOS [25] contains the only object type not

handled directly by the nucleus. These objects are called either composites or connections. The BIOS controls all the hardware device handling and file handling. When configuring the BIOS a vast selection of hardware device drivers are available to include in the system. These drivers are given physical names which enable them to be 'attached' using BIOS system calls and a token is given for the connection, which can then be used to communicate with the physical device. The devices may be anything from floppy disk drives to bubble memories. Communication with these devices is done using the read and write BIOS calls. All of the BIOS calls are asynchronous, that is when an attachment or read or write is requested, it is not done immediately. The concerned task, when it requires the result, must wait at a designated mailbox to receive the information. In the case of device reads, the task makes a BIOS read call, specifying the number of bytes required. The task can then either wait at the specified mailbox for a result segment, which can either contain error information or information on the actual number of bytes read. Instead of issuing a receive call to the mailbox, the task can execute the BIOS wait call which returns just the bytes read information and not the full results segment. Synchronous calls on the other hand would be complete on return from the initial request and would require no waiting at mailboxes.

The BIOS is also responsible for organizing the file handling and directory/file tree structure, an example of

which is shown in figure 3.3. Attaching files and directories is done in exactly the same way as physical devices, except the physical device on which they are contained must be attached first. Unlike physical devices files may also be created, deleted, truncated etc. These BIOS calls and the order in which they may be used are shown in figure 3.4. Other BIOS calls are also available to obtain information about files and directories such as when they where created etc.

The BIOS therefore makes device and file handling extremely easy, in application software. It is used in the Superweld application to handle the terminal communication and floppy disk handling for the mass storage.

3.4 The Extended Input/Output System (EIOS).

The EIOS [26] or any of the following operating system modules except the System Debugger are not used in the Superweld application system. All modules are used to form the complete iRMX development operating system.

The EIOS is similar to the BIOS except all the calls are synchronous and files may be logically attached. When a file or device is logically attached it is given an abbreviated ASCII name enclosed by colons, such as :LP: for a line printer. The advantage of this is that other tasks need only know the logical name of the file/device to use it and so all assume an agreed name for an agreed file/device. The task that makes the logical attachment can of course change the file/device without the other tasks knowing.

3.5 The Human Interface (HI).

The HI [27] is the software that communicates via the terminal handler with the user. It basically is a command parser and buffer handler. Once the command line has been received from the user it is parsed and the correct actions taken. This usually involves using the application loader to load command software and execute it. Error decode and prompt displaying are also handled by the HI.

3.6 The Universal Development Interface (UDI).

The UDI [28] is a much simplified call interface for use by command files. File handling calls and parsing calls are all included for use by system programmers for developing system utilities and language compilers etc.

3.7 The System Debugger (SDB).

The SDB [29] is a large piece of debugging software that enables the user to examine the state of tasks and objects. On request it can supply a large amount of information about an object. It is instigated by the execution of an interrupt 3, either from the front panel interrupt button or by the execution of a software interrupt 3 coded in the application software.

3.8 The Interactive Configuration Unit (ICU).

The Interactive Configuration Unit [30] is a large piece of system software enabling the applications programmer to take any of the above levels of software and build a custom operating system. The configuration is done in four distinct stages. **Firstly** a copy of the 86/30.DEF file is taken and renamed, in this case **APPL.DEF** was used. The file 86/30.def is provided with the iRMX kit and

contains default hardware and software configuration information for the ICU. The 86/30.DEF file contains the default configuration for the 86/30 based system which must be altered by using the ICU which is instigated by typing ICU86 APPL.DEF. The configurator then asks a series of questions. The first screen of questions determines the levels of software required, in this case the NUC, BIOS and SDB are required. Questions are then asked about the nucleus, the port addresses, the address of certain chips etc, all these are left as the defaults. The next questions are for the BIOS configuration and are all left unchanged until the DRIVERS screen appears. A large number of Intel drivers are then displayed and the operator must choose which are required. The iSBC 208 floppy disk driver is the only one necessary. Question are then put about the disk drives and the disks being used with the iSBC 208 card. These are all straightforward and include the type of disk drives, the physical size, the disk size, the number of drives and the physical name of the drives so they may be physically attached in the program using this name.

The Application Jobs must now be configured. This process again is straight forward and easily completed. The system needs to know such facts as whether the iNPX 8087 is being used by the job, where the start address for the root task is, the maximum number of objects that are going to be created and finally the priority required for the root task. This process must be repeated for the SDB if it is being included, as iRMX 86 version 5 treats the

SDB as an application job. The values for the SDB application job screen are given in the SDB manual [29].

A Utilities screen must then be completed to tell the ICU where the NUC, BIOS etc modules are kept and where the languages and utilities (linkers, locators and libraries etc) are kept on the development system.

With the first stage complete the ICU will then generate a macro file called **APPL.CSD** as the **second** stage automatically. The **third** stage is then ready and the operator should exit the ICU. The macro is then submitted for execution, this for the NUC, BIOS system takes around 10 minutes, for a full system it takes nearly 30 minutes. During this final configuration the macro produces several error and warning messages mainly from the locator, these are all 'correct' and should be ignored. 'Incorrect' error and warning messages are never produced as the ICU will not allow 'out of range' data to be entered at the configuration stage.

When the third stage is complete a file called **APPL.SYS** will have been produced, the system must then be completed using the librarian. Entering LIB86 the linked and located application code is added to the file **APPL.SYS** along with the SDB code. Before this is done it should be checked that the start address of the application code is where the ICU was told it would be by checking the final locate map, as when altering the root task code the start address often moves due to the insertion of code and/or data before the start address.

The full application system should then be complete and is placed on a floppy disk under the name **/SYSTEM/RMX86**. When the disk is booted, using the bootstrap loader contained in the monitor ROM on the 86/30, the /SYSTEM/RMX86 is the default file which is booted and run.

The ICU produces a system with the memory map as shown in figure 3.5. This map can be changed by altering the memory configuration page in the first stage of the ICU. The memory gaps are altered such that they coincide with the size of the operating system portions. In this way the ICU can be forced to place pieces of code at certain locations. The application system, during development, is placed at the top, so as it expands no other code is moved.

Overview

The above described objects are used extensively in the Superweld application system. The application is first divided into several tasks. The root task creates the other main tasks, which all then wait at a semaphore for a single unit. These main tasks deal with disk handling, data and path handling, and welding. The semaphores therefore turn the main tasks on and off as required.

Mailboxes are used to communicate with the smaller utility tasks. The utility tasks execute utilities that the main tasks require, such as file creation. So like subroutines they need to have information passed to and from them. In the case of the create file utility, it needs to know in which directory to create a file. So the

directory token is posted to it, through an agreed mailbox and it returns a file token if the creation is successful.

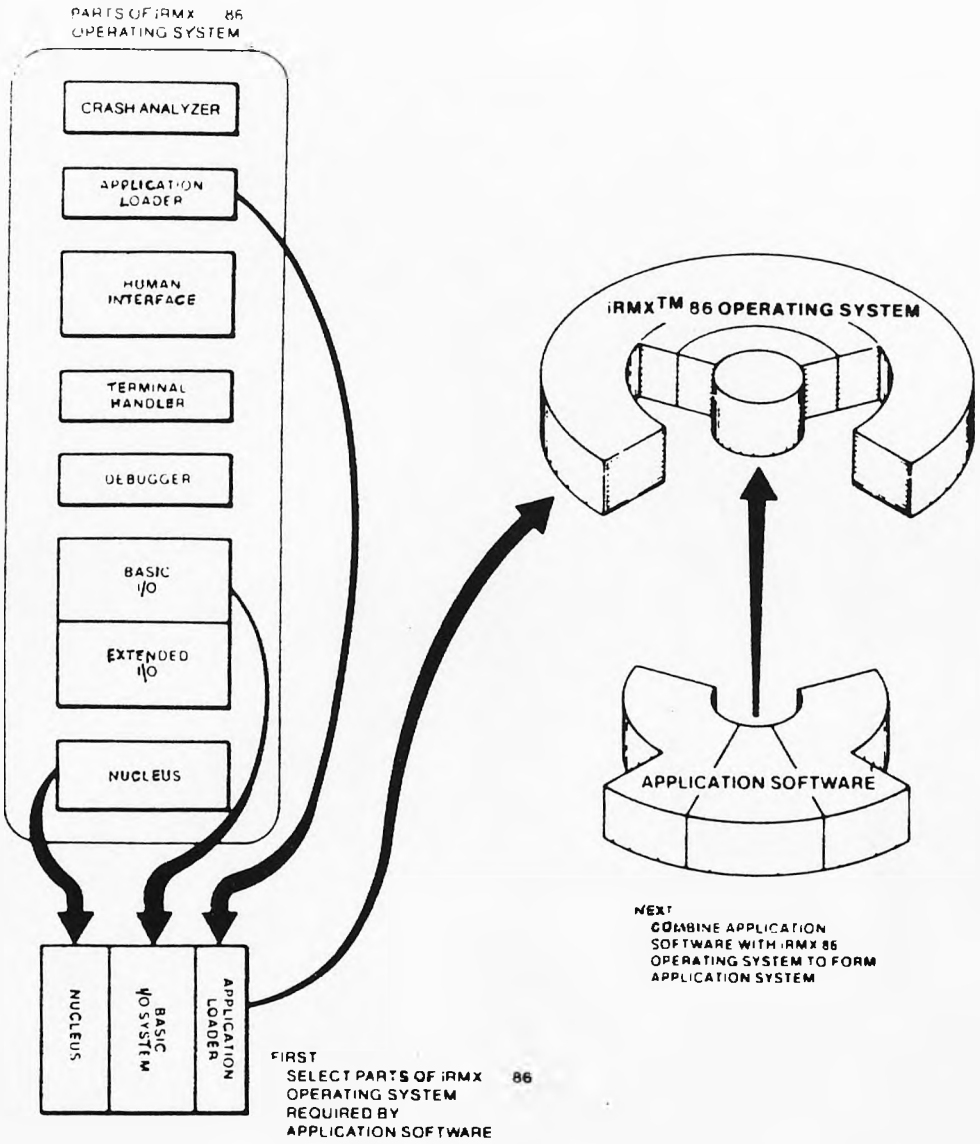


FIG 3.1 CONSTRUCTION OF AN iRMX APPLICATION SYSTEM.

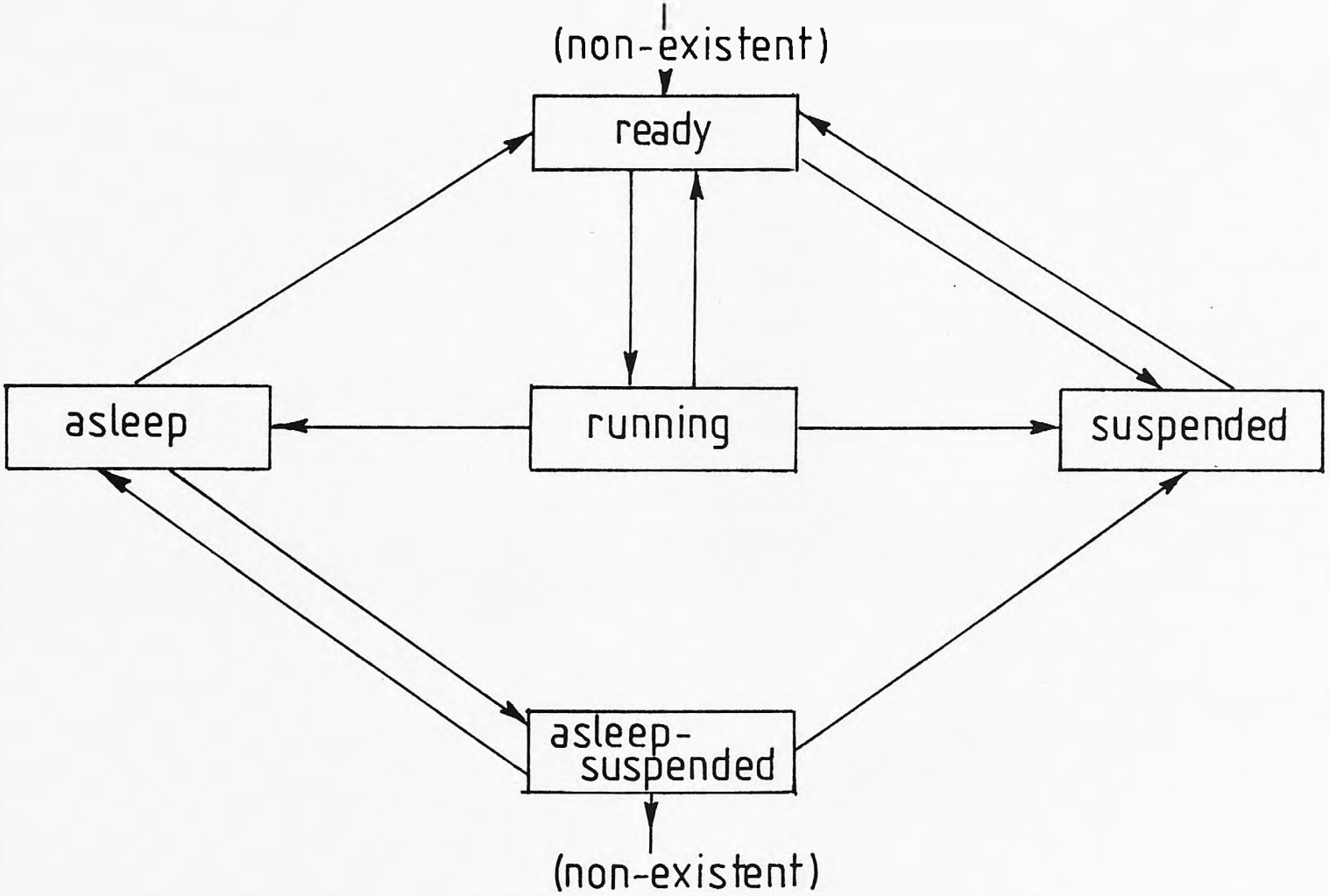


FIG 3.2 TASK STATE TRANSITIONS

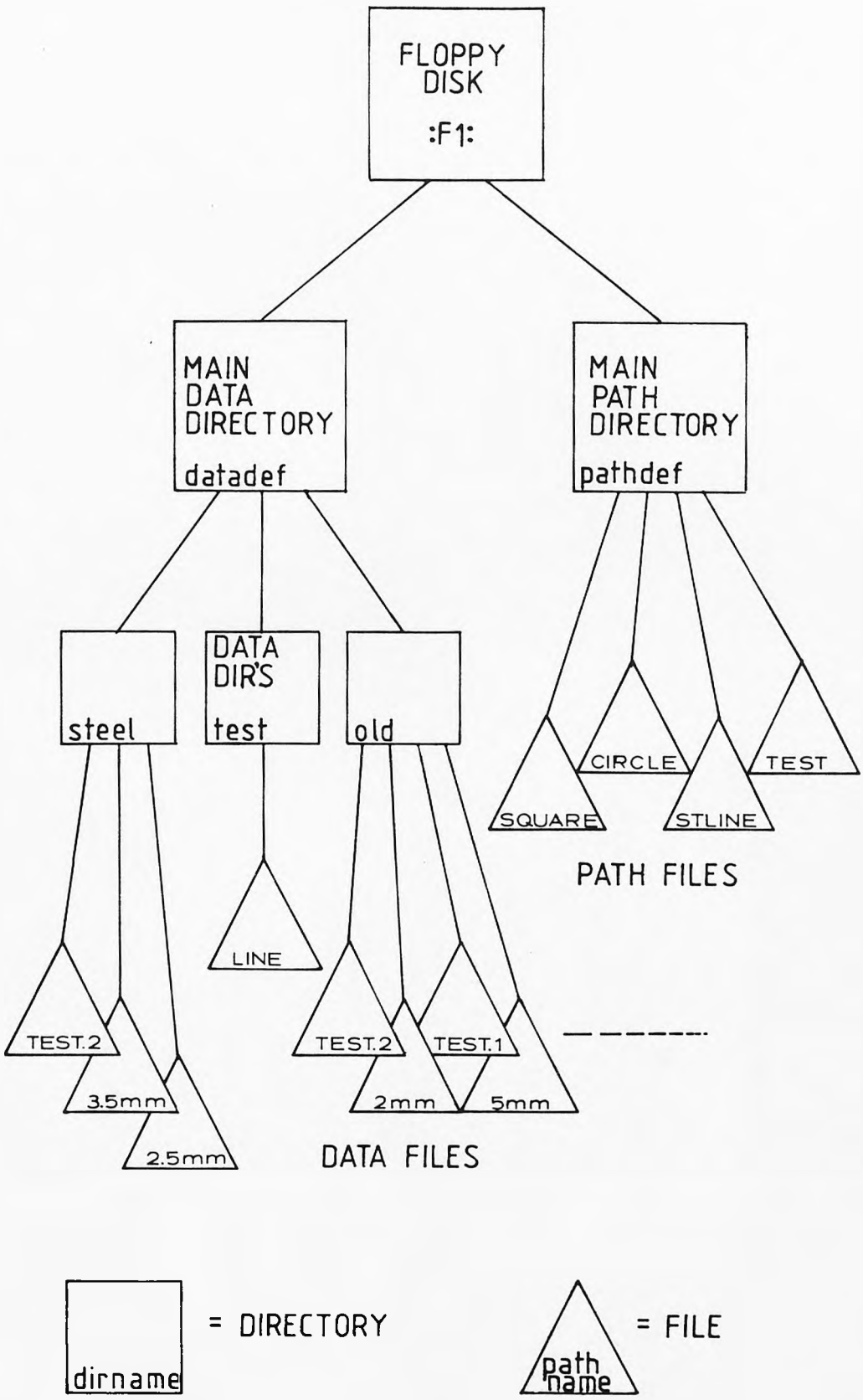


FIG 3.3 DATA AND PATH FILE STORAGE

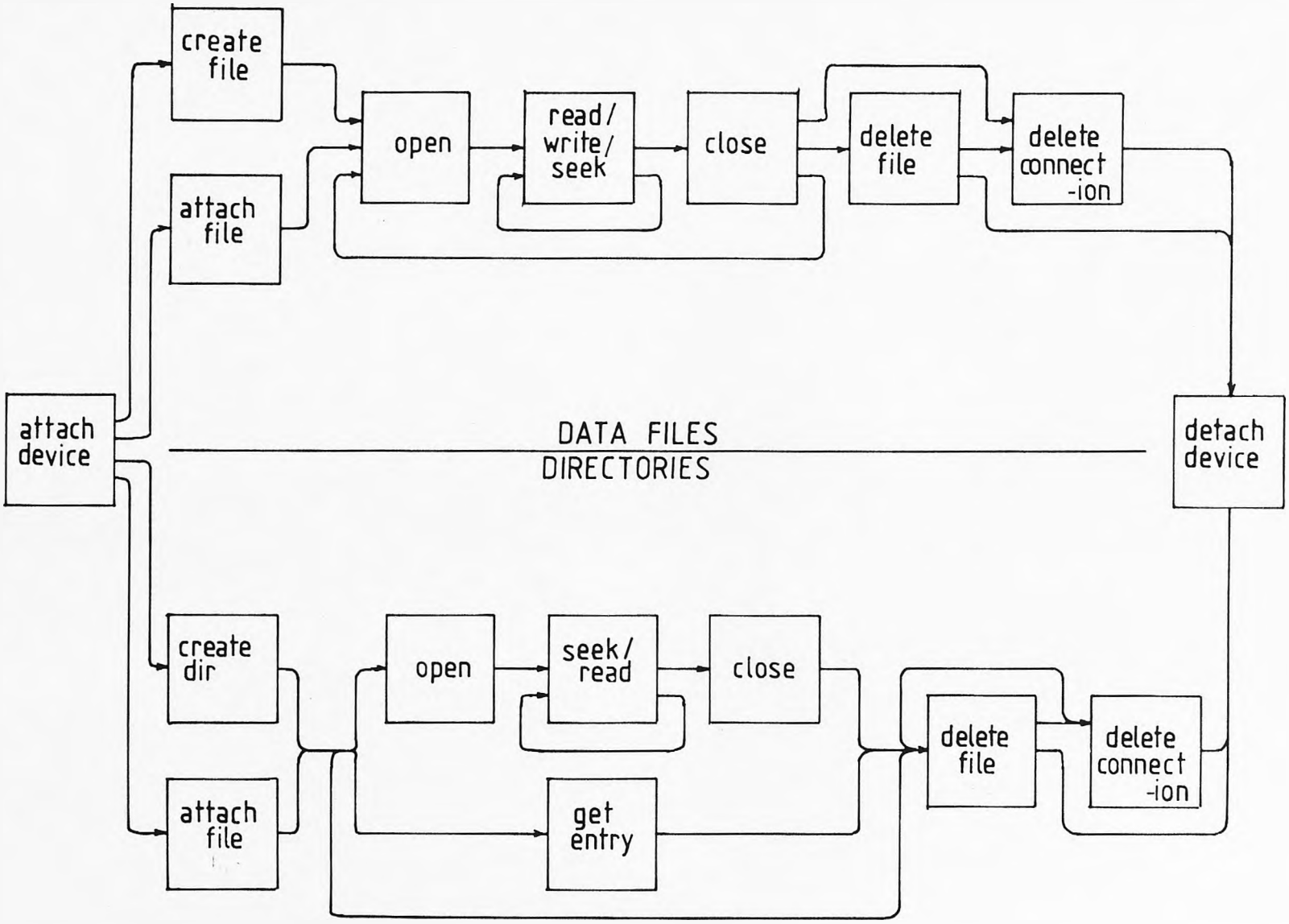


FIG 3.4 SYSTEM CALLS FOR FILE HANDLING

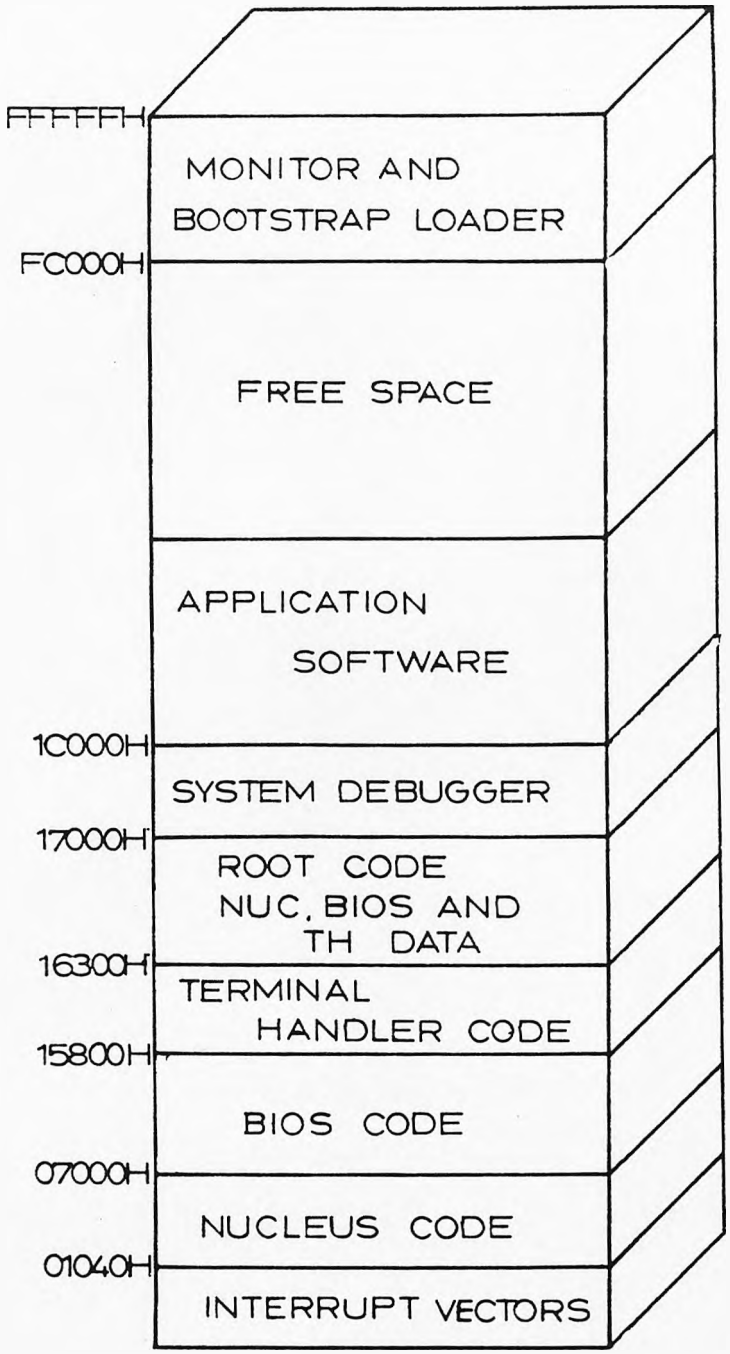


FIG 3.5 TYPICAL MEMORY MAP

CHAPTER 4.

THE APPLICATION SOFTWARE AND OPERATING SYSTEM.

4.1 Design Method and Structure.

The development of the application software was done using the 'Top Down' methodology. This means the high level software was written first and the lower levels where designed and added later. The software written can be split into three levels ;-

1 Main Command Level — **Language Used**

a). Supervisor task. PL/M 86

Provides software and hardware initialisation and initial command line choice.

b). Disk Handling Task. PL/M 86

Checks the mass storage disk.

c). Data Library Task. PL/M 86

Performs all the parameter input and data file handling for the welding parameters.

d). Path Library Task. PL/M 86

Performs all the path input and robot control for the path description and file handling.

e). Welding Task. PL/M 86

Reads data and path files for welding and performs all calculation pre-welding.

2 Service Utilities Level

a). Terminal Write Task. PL/M 86

Controls all the formatted writing to the user's terminal during all processes.

b). Terminal Read Task. PL/M 86

Reads the terminal for user input when required by other tasks.

c). Get Filename Task. PL/M 86

Gets user input via b) and checks the validity for use as a filename.

d). List Directory Task. PL/M 86

Lists the required directory's contents to the user's terminal in the correct format.

e). Createfile Task. PL/M 86

Creates files in the required directory with a unique filename input by the user.

f). Attachfile Task. PL/M 86

Attaches a valid existing file to a task.

3 Hardware Drivers

a). Robot Drivers. ASM 86

Used by the path task to control the robot.

b). Welding Drivers. ASM 86

Controls both power supply and robot while welding is in operation.

The software is given, in full, in appendix A. All references to the software are given as line numbers in the listings. e.g. {258} in the Supervisor is the program entry START$TASK. Line numbers preceded by 'D' indicate the assembly language driver of the particular task is being referred to. The listing are fully commented and fully explanatory, but an overview of the design and working of each task is given below. The utility tasks are described immediately after the Supervisor task as they

are used extensively by the rest of the main level tasks. The two drivers file are described with their main level task, the Path Library Task in the case of the Robot Drivers and the Welding Task in the case of the Welding Drivers.

4.2 The Supervisor Task

The Supervisor task is also the root task of the application job. The root task must still be written as a main procedure, even though it is the main routine of the software, as the iRMX operating system requires this. As soon as the application job is started, by the operating system, it must call RQENDINIT$TASK to tell the iRMX operating system that it can continue running {259}.

The iNPX 8087 numeric processor is then initialised {260} and the hardware is reset {262,245-257}, this turns all the peripherals off, such as the power supply and robot, then sets up the I/O port for communication to the peripherals. The main semaphores and mailboxes are then all created and catalogued in the job directory {263,175-192}, so other tasks may look them up and use them. The terminal handler tasks (described in sections 4.3.1 and 4.3.2) are then created {264,202-223}. These two tasks allow reading and writing to the terminal by sending a segment of memory to the correct mailbox. The utility tasks are then created {265,224-234}, these perform basic functions for the application system which more than one main task will use. The functions have been written as utility tasks only if more than one other task wishes to

use the function. The utility tasks are all described later.

The terminal bell is then rung {267-268} to indicate that the start up process is complete and the Disk Handling task is created {269,235-239}. The Disk task on creation waits for a semaphore unit, the Supervisor sends it this unit {270},which causes it to run and attach a data storage disk. The exact process is described in section 4.4.

Once this is all completed the other three main command tasks are created {271,193-201}. Using the two terminal communication tasks, the Supervisor task then writes a prompt to the screen and requests input from the terminal {273-277}. The input can be one of four commands {278-284} and depending on the input the correct task is started by sending a unit to the correct semaphore. As all the other tasks are higher priority (see figure 4.1) the Supervisor task stops and allows the other requested task to run, which will then immediately suspend the Supervisor until it has finished.

When the requested task unsuspends the Supervisor task, indicating it has finished, the Supervisor will loop and request another input from the terminal.

The Task Priority Diagram shown in figure 4.1, shows the relative priorities that the Supervisor gives the other tasks on their creation. Any task that uses another task must be of lower priority than the one it uses, so it will start running immediately.

4.3 The Utility Tasks.

4.3.1 The Terminal Write Task.

The terminal write task is a single routine with no procedures. When initially created it will start running due to it's higher priority than that of it's creator, the Supervisor task. The initial priority of all tasks, except the root task, is set in the call to create the task. The root tasks priority is set in the configuration process by the ICU. The Terminal Write Task first creates it's own mailbox {136}, the th_in_mbx and catalogues it in the job directory under the ASCII name THINMBX {137}. This process is repeated {138-139} with the th_out_mbx. The task then creates a user, number 65535. This user number is used to make a connection to the Terminal Driver in the BIOS. A connection made by user 65535 may be used by any other user. The physical connection is made {141} to the 'T0' driver, which is in fact the BIOS USART driver named 'T0' in the configuration process.

The token for this connection is received at the response mailbox specified in the attachment call and a file is created on this physical device. This file is then opened {145} and the connection catalogued {146}. The code 'ESC=' is then written to the terminal {147}, which initialises the terminal correctly, the Supervisor waits for this to be completed {148}. The buffer used for writing to the terminal is then filled with the terminal escape code which positions the cursor on the screen {149} and the task then sleeps at the th_in_mbx waiting for a segment of memory. When received the segment should

contain, in the first two bytes, the X,Y location for the cursor and, in the third byte, the number of bytes to be printed. The position cursor code is therefore completed {153-156} and the ASCII string in the segment copied into the write buffer {157-158}. This ASCII string is then written to the terminal connection and the segment is returned to the sending task {159-161}. The task then loops back to sleeping at its mailbox.

4.3.2 The Terminal Read Task.

The Terminal Read Task, again is a single routine task like all the utility tasks. This task, on creation, creates and catalogues two mailboxes, the read_in and read_out mailboxes {127-130}. The task then looks up the terminal connection catalogued by the write task {131} and, like the write task, then sleeps at its mailbox waiting for a segment of memory. When this segment is received the task writes to the terminal connection to position the cursor at the start of the answer line {135-136}. A buffer is then sent to the terminal driver which places any typed characters into this buffer and returns it when a carriage return is typed {137-138}. The first byte of the segment is then filled with the count of the characters received and the answer line is cleared {139-141}. The segment is then returned to the requesting task via the response mailbox {142} and the task loops to wait for the next segment.

4.3.3 The Get Filename Task.

This task is nothing more than an extension of the read and write tasks. With a large amount of file

handling, filenames are often requested from the user, this task obtains the filenames and checks they are valid.

It creates its own mailbox, the gname_mbx, and looks up the read and write task mailboxes {121-126}. The task then waits at its own mailbox for a segment of memory which is immediately sent to the read task to be filled in {129-134}. On receipt of the segment the ASCII string is checked to see if any illegal characters exist and the count is checked to see if its between 3 and 9 {135-142}, the legal length for a filename. If any of the checks fail the error line is written {134-148} and the terminal read is repeated. Otherwise the correct filename is sent back to the requesting task {150}.

4.3.4 The List Directory Task.

This task is to copy the contents of a specified directory to the terminal, displaying the filenames in four columns on the terminal screen.

The task on creation, creates its own list_dir_mbx and looks up the write task mailboxes {129-132}. The task then waits at its own mailbox for a directory token.

When this token is received the screen is cleared and the write buffer is set up for the first filename {143-145}. The task then loops {146-164} requesting each directory entry in turn and if the entry is not empty it displays it on the screen {151-161}. If the entry is neither empty nor valid an end of directory code is assumed {146} and the program falls out of the loop, deleting the write buffer segment and returning the

directory token, indicating to the sending task that the listing has been completed {165-166}. The task then loops and waits for the next directory token.

4.3.5 The Createfile Task.

File creation is required by both the Data Library and Path Library for data and path storage. This task is used to obtain a token for a new file. The task requires the directory in which the file is to be created, so it waits at the cre_file mailbox for a directory token. If successful in it's file creation a file token is returned. The create_file task creates it's mailbox as usual and looks up the other tokens it requires {132-138}. It then waits for a directory token {133} and sends it to the List Directory task {142}. Once the token is returned {143} the Createfile task displays the file name prompt {144-146} and uses the Get Filename task {147-148} to obtain a name for the file to be created.

The actual createfile call {149} can produce one of two things. The call may either be successful and return the file token or be unsuccessful (the file may already exist) and return an error segment {150}. Whatever the result, the token is sent to the requesting task {152}, which determines the result and takes the appropriate action.

4.3.6 The Attachfile Task.

This task serves both the Data and Path Libraries by obtaining tokens for already existing files . This enables stored data to be retrieved and used.

Once the att_file_mbx has been created, catalogued

and the other required mailboxes have been looked up, the task waits at its mailbox for a directory token. As in the Createfile task, this directory is then listed to the screen {144-145} so the operator can choose the file to be attached. The task then repeatedly displays the filename prompt, gets a filename {153-154} and attempts to attach the file {155}.

The token received is then checked to see if it is a file connection token or an error segment token {158-160}. If it is an error segment token, the segment is deleted and the process is repeated {161}. If the attachment was successful the file token is returned to the requesting task {169}.

4.4 The Disk Supervisor.

The Disk Supervisor task is responsible for attaching the floppy disk drives and checking the drive contains a disk. Once a disk is found to be present it is checked to make sure in contains the correct storage directories.

The Disk Supervisor is a command level task. When this task is created, it looks up all the tokens it requires from the job directory and then creates a mailbox for internal use {216-222}. The task then waits at the disk_sem semaphore for a single unit from the Supervisor.

On receipt of this unit the Disk Supervisor suspends the main Supervisor and creates a segment for terminal communication {227-229}. The task then places a prompt on the terminal to request a correctly formatted data disk to be inserted in the right hand drive and waits for a

carriage return to indicate this has been done {232-236}. The disk_flag is then checked to see if the drive has been attached previously, if it has, the drive is detached {237,188-194} and it's token taken out of the catalogue.

The drive is then re-attached {239,176-187}. If this attachment fails, an error segment is returned instead of the connection token. This segment is deleted and the disk_flag is set {182-184}. The error message is then displayed and the request for a data disk is then repeated {171-175}.

If the physical attachment is successful then the default storage files 'DATADEF' and 'PATHDEF' are checked for their presence {246-248}. The check_dir call used {195-214} attempts to attach the filename passed as a parameter. If it fails, the error is flagged, reported to the screen and the drive is detached {200,209}. If it succeeds then the file is detached {212} and the routine returns.

Once all these checks have verified the disk, the connection is recatalogued {244} and the screen is cleared {249-252}. The Disk Supervisor then unsuspends the main Supervisor task, to allow it to continue running, and returns to waiting for the next semaphore unit.

4.5 The Data Library Task.

The Data Library Task (DLT) contains all the utilities concerned with organising the data files. The data files contain all the power supply data and some preset peripheral information. The utilities enable manipulation of these files, as shown in chapter 5, and

are as follows.

1. INPUT. Allows the user to input data to a specified file.
2. EDIT. Allows the user to transfer data from one file to a new file and then edit the data in the latter.
3. DIR. Causes a listing of the specified directory on the terminal screen.
4. LIST. Lists the data contain in the specified file on the terminal screen.
5. LOAD. Loads a data file ready for the welding task.
6. DELETE. Erases the specified file.

The data file storage structure, shown in figure 3.3, is a dual level system. The data files are relatively small and so are split into directories.

The DLT, when created, looks up all the required tokens {711-721} and waits at the data semaphore, data_sem, for a single unit. On receipt it suspends the Supervisor task {726} and attaches the default data directory {732-735}. The DLT prompt is then displayed and the read task waits for one of the above commands. The command is then parsed {747-757}. A single carriage return with no text results in the DLT resuming the Supervisor {744,770}. If a command is identified the correct procedure is called.

The INPUT routine {612-699}, first uses the create directory procedure {237-276}, which lists the data default directory (shown in figure 3.3) and uses the get

name task to obtain a directory name. The create directory routine then attempts to attach this directory {252-261} to see if it already exists. If it does then this directory connection token is returned, else a new directory is created {263-270} and this token is returned.

The Createfile task is then used to create a new file in the directory attached. The token returned may be an error segment, so the token is checked {620-627} and the command aborted if it is not a file connection token.

The file is then opened and the data screen is displayed (described later) with default values next to all the parameter headings {631-635}. A new prompt is then displayed containing the DATA INPUT commands which are now available {636-640}.

The data is changed by typing the letter next to the parameter followed by a space and the new value. The INPUT line parser {645-655} finds which parameter is being changed and passes the ASCII real number to the INPUT_REAL_NUMBER routine {189-236} which converts it to the internal real number format. The data is placed in the parameter matrix defined in line 166 and displays the new value on the screen. In this way data may be changed as the operator requires.

Other commands may also be input such as 'NEXT'. This causes the input routine to check the data {657} using the verify data routine {453-504}. If no errors are found in the data, it then stores the parameter matrix in the file and allows the next data set to be input. Up to 255 data sets may be input to one file to accommodate multipass

welding and multiple welds on a single path. 'END' causes the data to be checked and stored {676-686} and the input routine left once the file is closed and stored. 'QUIT' does the same except it deletes the file before returning to the DLT prompt level.

The 'EDIT' command {505-611} operates in a similar way to the 'INPUT' command except an already existing file must be attached {509-510} as well as a new one created {511-512}. The first file is then copied into the new one and the data is displayed and changed as in the 'INPUT' command.

The 'DIR' command {409-414} simply sends a directory token to the List Directory task for listing on the terminal. The 'DELETE' command {415-423}, attaches a file and deletes it. It then attempts to delete the containing directory {421}, this will fail if the directory is not empty, but prevents empty directories from existing.

The 'LIST' command {370-408} connects a file in the usual way and reads in the data one set at a time. The data is displayed using the data screen as in 'INPUT' and waits for a carriage return to exit or for a space and a carriage return to display the next set in the file {396-399}.

The 'LOAD' command {424-436} attaches a file which the operator specifies for welding with 'NEXT'. The routine purges the data mailbox to prevent any old data being left in it, then sends the attached file token to the mailbox. When welding is requested, this mailbox is

where the Welding task will expect to receive a data file.

The simple command tree structure, shown in figure 4.2, reflected in the software structure, enables new routines to be added simply by adding their ASCII instigation name and routine name to the command parser. The existing routines and utility tasks then make simple work of coding new functions.

4.6 The Path Library Task.

The Path Library Task (PLT) consists of two modules. The first is written in PL/M and contains the Path commands and the file handling. The path files, as shown in figure 3.3, are single level and are not filed in an intermediate directory. This is because the path files are larger in size and fewer in number, compared to the data files. The other module is written in assembly language and contains the software drivers for the hand held keypendant and the TIM welding machine.

After the usual initialisation the PLT waits for a unit to be sent to it's semaphore. Then, as in the DLT, it displays a prompt and waits for commands to be entered {650-680}. The command parser is the same as the DLT's, it calls the correct routine for the command entered.

The commands are as follows ;-

1. DIR. Gives a list of all paths present on the storage disk.
2. LIST. Takes the text from the specified path file and displays it to the user.
3. DELETE. This deletes the path file specified.
4. SCALE. Allows a path file to be copied from one

file to another and scaled in the process.

5. MOVE. Allows the robot to be moved using the hand held pendant.

6. TRACE. Replays a specified path with no operator interaction.

7. INPUT. Allows the operator to input a path to a file. Text is entered to identify the path and then the user drives the robot around the required path using the hand held pendant.

8. EDIT. The robot moves around the specified path and allows the operator to move points as required.

Only the last four of the above commands use the assembly language drivers. 'DIR' {580-583} works the same as the DLT command 'DIR' as does the 'DELETE' command {267-272} and the 'LOAD' command {273-283}.

The path file contains all the information necessary to retrace the input path. The information is stored as follows. The path description matrix has six columns, one for each motor, these are stored consecutively at the end of the file in the order, X,Y,Z,R0,R1,R3. These columns are divided into word locations, each containing a count of the number of steps the motor must take to travel to the next point on the path. The exception to this is when all the counts are 8000H (except for the X axis). This indicates a control point and the X entry contains a value indicating as follows ;-

X Entry	Indicates
01H	Start of weld point.
00H	End of weld point.
04H	End of path point.

Before this matrix, in the file, is stored five lines of text. This is entered by the operator when inputing a path for storage. The text is to identify the path with greater explanation than just the filename can achieve.

At the beginning of the file the path information block (PIB) is stored. It contains four variables. The first contain the number of bytes in each column of the matrix, which is the same for every column. The second is a count of the number of words in each column that contain point offsets and not control points. The third indicates the number of pairs of start and stop weld control points that exist on the path and therefore the number of welds. The fourth is as yet unused.

The 'LIST' command {431-514} attaches a file in the usual way {437} and reads the PIB. It then displays the filename on the screen and reads the five lines of text, displaying them on the screen {444-464}. The read_path routine is then called and each column of the matrix is read into the memory {344-352}. Once a carriage return is read from the terminal {463,412-426} each point in the matrix is converted to ASCII {474-492} and displayed in six columns on the screen. If the point is a control point, a message is displayed to indicate the point {494-502}. The 'LIST' routine then waits for a futher carriage return {510}, before returning to the PLT command level.

The 'SCALE' command {284-343}, attaches an already existing file {289-290} and creates a new file {291-292}. The whole contents of the old file are then read {302-314} and the PIB and text are written to the new file. The routine then obtains a scaling factor from the user {315,207-266} and multiplies each X, Y and Z axis offset point by the scaling factor {317-334} and overflows are checked for. The new matrix is then written to the new file and both files are closed and detached {335-342}.

The 'INPUT' command creates a new file {373-382} and call the text input routine {353-371}. This routine reads five lines of text from the terminal and places them in the new path file. The keypendant prompt is then displayed {385-388} and the keypendant mask, 'PINPUT', is set to allow the input keys only. The assembly language routine key_driver is then called. For a full explanation the reader should refer to the listings.

The keypendant is scanned by an interrupt routine which is executed every 10mS. This interrupt system is described fully in the welding software. The resultant information from the keypendant allows the motor enables to be set {D 327-337} and the directions in which the motors are to be moved to be calculated. The clocks on the custom board are programmed with a known speed {D 423-468} and set running {D 343-349}. The speed is set so that the number of steps the motors will execute in the next 10mS is known and this is added to the step counts {D 354-386}.

The special keys are scanned {D 387-408} and their

routines are executed if no motors are running. The speed keys move a pointer up and down a sliding scale of counts which the clocks are loaded with {D 695-715}. The enter key moves the present offset counts into the matrix and clears them, ready for the next offset count {D 603-652}. This can also be forced by the offsets getting too large causing the AUTOPT flag to be set.

Special points such as the start and stop weld keys cause control points to be placed in the matrix {D 656-675} and enter a normal point indicating where this control action should take place.

Once the path has been entered the exit key is pressed. The exit key {D 737-820} routine checks the total offsets of each axis to make sure the torch is within the minimum return distance. The total offsets are kept on the 8087 real stack and are added to each time the motors move in the same way as the offset counts. If the torch is within range, the exit routine calculates, from the total offsets of each axis, the points needed to be placed into the matrix to return the torch back to where it started. It then sets the AUTOFLG so that the trace routine finishes the path.

On return to the PL/M the path store routine is called which places the matrix into the file along with the PIB and closes the file.

The 'MOVE' routine works similarly to the 'INPUT' routine except the matrix is not completed by the driver and the exit routine allows the 'MOVE' command to leave the assembly code without returning the torch to the

origin. This allows the operator to move the torch to anywhere in the robot workspace.

The 'TRACE' routine allows the operator to replay any stored path. A path file is attached and read {552-564} and the variable PINPUT is set to 0FFH. This indicates to the driver that trace mode is required and no user input is to be allowed. The assembly language routine, trace, is then called. This routine takes the offset counts from the matrix and calculates the speeds for each motor and time the line will take to describe {D 973-1127}. The motors are then set running and the time is counted out. The process is then repeated until the maths routine finds an 'end of path ' control point. The FINISHED flag is then set, the interrupts are stopped and the code returns to the PL/M.

The 'EDIT' function is the same as the 'TRACE' function, except the driver waits at each point until the operator presses the enter button. The drivers will then move the robot to the next point. If the move button is pressed, the input drivers are switched to {D 325-326} so that the robot can be moved by the hand held pendant and the offset counts are updated. When the second move button is pressed the driver returns to normal after moving the point to the new location using the new offsets.

In this way paths can be input, edited, stored and retrieved for welding on the Superweld system. The Motor Drivers for this file are written as 10mS interrupt routines (described in section 4.7.2). Thus each cycle of

the software routines must always execute in under 10mS, in order to keep the timing correct. The software has also as usual been written with adaptability in mind. Because of these two considerations some of the code takes a long hand form. The routines XYZ_MOTORS and ROT_MOTORS, for instance, are identical and do nothing but load the same speed count to all six custom board 8253 clocks. The routines could have been written as a single looping routine, incrementing the address by two each time. This however would have been slightly slower due to the loop instructions and time saving is of the essence. The routines are also easily modified. More axes may be added at different addresses (the three axis board may be added as well as the six axes board). Some axes may be changed, say to servo-motors, and these routines would need to be changed in part, showing the need to keep each axis separate.

4.7 The Welding Task.

The welding software is the most complex part of the SuperWelder. It basically handles current control and robot movement but also has to deal with peripheral units such as wirefeed, seam trackers and weld penetration monitors. The software can therefore be categorised into the following sections.

- Current Control. This is not simply pulsing the current at the correct time and producing the correct current waveform. It includes communication with the robot movement software to stop the torch during arc striking and stand-

by times etc.

- Robot Control. This is the accurate movement of up to six axes as the path file describes. It must also take note of signals from the current control to stop movement while certain current conditions prevail. This part of the software also detects when each single weld has been completed and momentarily returns to the main software to pick up the next set of current data and calculate new seam times etc.
- Wirefeed Control. Integrated into the current control software the wirefeed control emits a digital pulse at the same frequency that the current is being pulsed. The velocity of such wirefeeds are usually set by the wirefeed control units themselves.
- Penetration Control. When penetration control is required all high current pulse times are increased by 20% and during these pulses the penetration port is sampled. Once a high is received at this port then the current is returned to a normal background current pulse.
- Gas Control. Pre- and post-gas purge times are set in the data file. These basically turn the gas solenoid on and off at set times before and after the welding to ensure inert gas saturation of the welding area.

Before welding two files must have been loaded, one path file and one data file. Both files are loaded by their respective librarians by typing 'LOAD' and then the full filename. Once each file has been loaded the system will return to the main operating system level allowing direct entry into the other librarian. Once the two files have been loaded 'WELD' may be typed at the main operating system level. No more user input is now required, the system has all the information it needs to weld. If both or either of these have not been loaded then the welder aborts and reports the fact.

The welding software consists of two main sections the first of these is written in PL/M86 and is basically preparation for the welding including translation of the data and calculation of the seam times etc. The second is the real time software written in ASM86 to control the system during the welding process.

4.7.1 Data Preparation

Firstly the path file is read into memory {420,236-274} including the weld and point counters. If no reading errors are detected then the data file is opened and the number of sets of data is read {421,222-235}. The filenames are then displayed on the console screen {425,361-382} and the number of welds is compared with the number of sets of data. If they are not equal then the number of sets of data must be one and this set will be used repeatedly for all welds.

The main welding loop is then entered {431}, this loop is repeated until all welds have been done. First one

set of data is read from the data file {432,275-291} and an assembly language routine SEAMTIME is called {298}. This routine scans the path matrix for a start of weld indicator and using Pythagoras' theorem calculates the total length of the path up to the next stop weld indicator. The velocity being known from the data file, the total seam time may then be calculated. This routine is placed in assembly language due to the lack of a square root function in PL/M. As the total seam time is now known, the time at which slope down starts may be calculated so that the slope down finishes just before the arc is extinguished {320}. All times taken from the data file are then converted to multiples of 10mS as this is to be the basic time unit {312-319}. Currents are put in units of Imax/4096 (being the accuracy of the digital to analogue converter) and are shifted left four bits to allow room for the channel number on the D/A {328-332}.

Once the data is prepared the routine selection word may be filled in {342-351}. This word, shown in figure 4.3, reserves a bit for each routine present in the current handler, if the bit is set then the routine is to be called, repeatedly in some cases, until the routine resets the bit itself. Thus if slope up is required the slope up bit is set and at the correct time the current handler will call the routine. The slope up routine then makes the required alteration to the current and decrements its time counter. This process repeats until the time counter is zero. The routine will then reset its

own routine flag so that it is not called again and the next routine may be called. Some bits, however, are always set, these are for routines that are always to be called whatever the data but they are treated in the same way keeping the current handler modular.

The priority of the welding task is then raised {439} to stop other tasks such as the systems disk controller from pre-empting it. The assembly language welding routine, which is dealt with in section 4.7.2, is then called. Once the welding routine has finished one weld it will return. If any welds are left another data set is read (if this fails then the last one is read again) otherwise the system falls into the weld routine again not to weld but to finish off the path. This is achieved because the system traces expecting to find a start of weld indicator as usual, instead it finds an end of path indicator causing it to return and as the end of path flag is set {431} the PL/M falls out of the main welding loop.

The task is then returned to its usual priority and the files are closed and detached {451-459}.

4.7.2 The Welding Process

The real time welding software is completely written in ASM86. Once the PL/M has called the ASM86 weld routine using a FAR call the welding routine first sets up its segment registers {D272-314} then proceeds to initialise its variables. The majority of the variables are flags all of which are reset to zero, the D/A is reset {D297-301} and the current handlers flags are loaded into DX with a pointer to the start of the jump table for the routines in

BX.

Attention is then paid to the onboard 8253, which is a fully programmable timer. Timer one on the chip (the only one available as the others are used for USART and system clocks) is then placed in mode one. This mode produces a single pulse on the clock output after the count has expired, such that as it is being driven with a 2.36MHz clock and loaded with a count of 5FB7H we get a 10mS pulse. The output of this clock is jumpered on the board to master interrupt four, so the system then assigns to this interrupt level (via the RMX86 call RQSETINTERRUPT) the welding routine {D279-290}. The timer's gate is then taken high through a port on the 86/30 and the program hits a halt instruction {D315}.

When, after 10mS, the interrupt is received the clock is immediately reloaded with the same count {D423-428} and as soon as it has been serviced it is re-enabled {D439-453}. Thus a 10mS tick is received causing, as long as the interrupt routine is shorter than 10mS, our software to loop every 10mS so that the actions required are executed accurately in real time.

After every interrupt return the program will continue from the instruction below the halt which simply tests the 'WFINISH' flag to see if the weld is completed {D316-317}, if not then immediately the program returns to the halt to wait for the next interrupt. If the flag is set then the timer is turned off to prevent any more interrupts occurring. The program then returns to PL/M

which decides whether the complete path has been finished or not by looking at the 'PATH FINISHED' flag in its main welding loop.

On entering the interrupt routine the timer is reloaded as described and the path and current portions of the routine are serviced as described below. On return from these routines the conditions of the 'WELD STOP' and 'CURRENT' flags are checked to see if a weld has been finished requiring a return to PL/M, if so the path index is saved to mark the extent of the path matrix that has been completed and the 'WFINISH' flag is set {D435-438}. The interrupt is then re-enabled and the program executes an interrupt return.

4.7.3 Path Servicing

The path servicing routine is called on every execution of the interrupt routine. First of all the 'WELD STOP' flag is checked {D645-646} to see if it is set. If it is (which is never so on the first execution of this due to the main routine clearing all flags on initial entry) then all enable lines are cleared to the external timers thus stopping the motors and path servicing is complete. If the 'MATH_DONE' flag is not set when it is checked {D647-648} then the maths needs recalculating, that is, the last line has been completed and new counts for the timers need calculating.

The welding maths {D518-614} also takes care of all the path's flag servicing as it deals with the path matrix directly. Firstly the offsets are checked to see if the point is a control point, signified by the number 8000H in

all the matrix legs except the X. If it is then it can be one of three controls and the following actions are taken {D519-539}.

XARRAY contents	Action
04H = Path end	'WELD STOP' is set to stop the motors, 'WFINISH' to allow the interrupt cycle to end and 'PATH FINISHED' to tell the PL/M the path has been completed.
00H = weld finished	'WELD STOP' alone is set, the current may still be on so 'WFINISH' is not yet set.
01H = weld start	The 'CURRENT' flag is set. This flag determines whether the current handler is called or not. 'WELD STOP' is also set so movement stops while the arc is lit.

If any of these conditions are present then the routine returns immediately, otherwise the maths is done as follows. The 8087 processor is employed to do the majority of the maths to increase the speed. The 8087 is first loaded with the X, Y and Z offsets and squares them all, then calculates the length of the line using Pythagoras {D540-561}. The length of the line is then divided by the required velocity to give the time it will take to traverse this line in units of 10mS. The routine

set sign is then called to set the direction bits for the direction port and remove the sign from the magnitude. Then the routine set_enable {D380-414} which ascertains which motors actually need enabling is called setting up the timer enable byte. The time count is then stored {D565} and a copy is multiplied by the external timer's clock frequency and the result by each offset in turn to give the count+1 for each timer as shown below.

line offsets= X_{off}, Y_{off}, Z_{off} in 1/100 ths mm (hmm)

velocity = V mm/S

line length (by Pythagoras) = L hmm

clock freq = 153.6 KHz

timer counts = C_x, C_y, C_z

traverse time T=L/V (10mS units=Hundredths of seconds hS)

max count in 10mS = 153.6 KHz x 10mS
= 1536

max count in T = 1536 x T

C_x+1 = (1536 x T)/X_{off}

Therefore the three main counts are calculated and loaded into each counter {D586-606}. The motors are then started immediately by outputing the direction byte and the enable byte to there respective ports {D607}. The maths then finally sets its own flag to indicate the velocities are done, moves up the path pointer past the offsets just used and returns. The movement routine will from now on just decrement the time counter each time the routine is passed through until it equals zero, at which time the maths flag is reset causing the maths routine to be repeated.

4.7.4 Current Servicing

Once the movement has been serviced then the 'CURRENT' flag is tested {D433-434} to see if the maths routine has found a weld start control point yet. Unless this flag is set the current handler is not called.

The full current waveforms are shown in chapter 5. The full current waveform, shown in figure 4.4, consists of the pre-stand by current (I_s1) which is the current output while the arc is struck, this current will prevail with the torch stationary for time t_s1 (pre-stand by time). The torch will then start moving and the current will slope up from the pre-stand by current to the pulse current over the slope up time (t_su). Current pulsing then takes place with the pulse current (I_p) prevailing for the pulse on time (t_p) and the background current (I_b) then being output for the pulse off time (t_b). This continues until it is time to slope down over the slope down time (t_sd) down to the post-stand by current. The post-stand by current (I_s2) is then held constant over the post-stand by time (t_s2) until the arc is extinguished.

This waveform is produced by the current handler which is a simple re-entrant jump table driven routine. Register BX holds the index to the address of the routine presently in use or last used. The current flags (figure 4.3) are rotated into carry and if the bit is set then the present routine is jumped too {D679-686}. If the bit is not set then BX is incremented twice such that it points

to the next routines address and rotation on the current flags repeated. The routines themselves are responsible for rotating the flags back and if they do not require executing again they clear the bit. When all flags are down the 'CURRENT' flag is cleared stopping the current handler being called again {D690}.

Most of the current handler routines are self explanatory such as the 'GAS ON' routine which clears its own flag immediately, as it is only executed once, then sets the gas solenoid port bit on and returns.

One complex routine however is the 'PULSE' routine {D694-753} used by most of the other routines to output current values. The pulse routine basically has two times, the background current time (t_b) and the pulse current time (t_p), and two currents, the background current (I_b) and the present current (PRES_I). The pulse routine switches between the two currents counting out their respective times. This however is complicated by sloping up and sloping down as the present current may be less than the background current, so first of all PRES_I is check to see if it greater I_b before pulsing is allowed {D699}.

The penetration control software is also included in the 'PULSE' routine {D704-708}. The routine checks the data to see if penetration control is required, if it is then all pulse times will have been made 20% larger. Now switching to background is not just a matter of waiting for the pulse time to expire but also the single bit port penetration peripheral is monitored and if this is at

anytime set then this indicates penetration has occured and the background current is switched to regardless of the pulse time count.

Wirefeed pulsing is required to be at the same frequency as the pulsing of the current so again the 'PULSE' routine takes care of this, flipping the wirefeed output bit on the control port everytime it swaps between the two currents.

In this way a single trace up to a start weld point and the weld following the control point are done. When the assembly language driver returns to the PL/M, the PL/M determines whether any more welds need to be done. If not then the ASM86 drivers are called once more so the final trace to the origin is completed and the path_end control point is found {D524}. The weld is now complete.

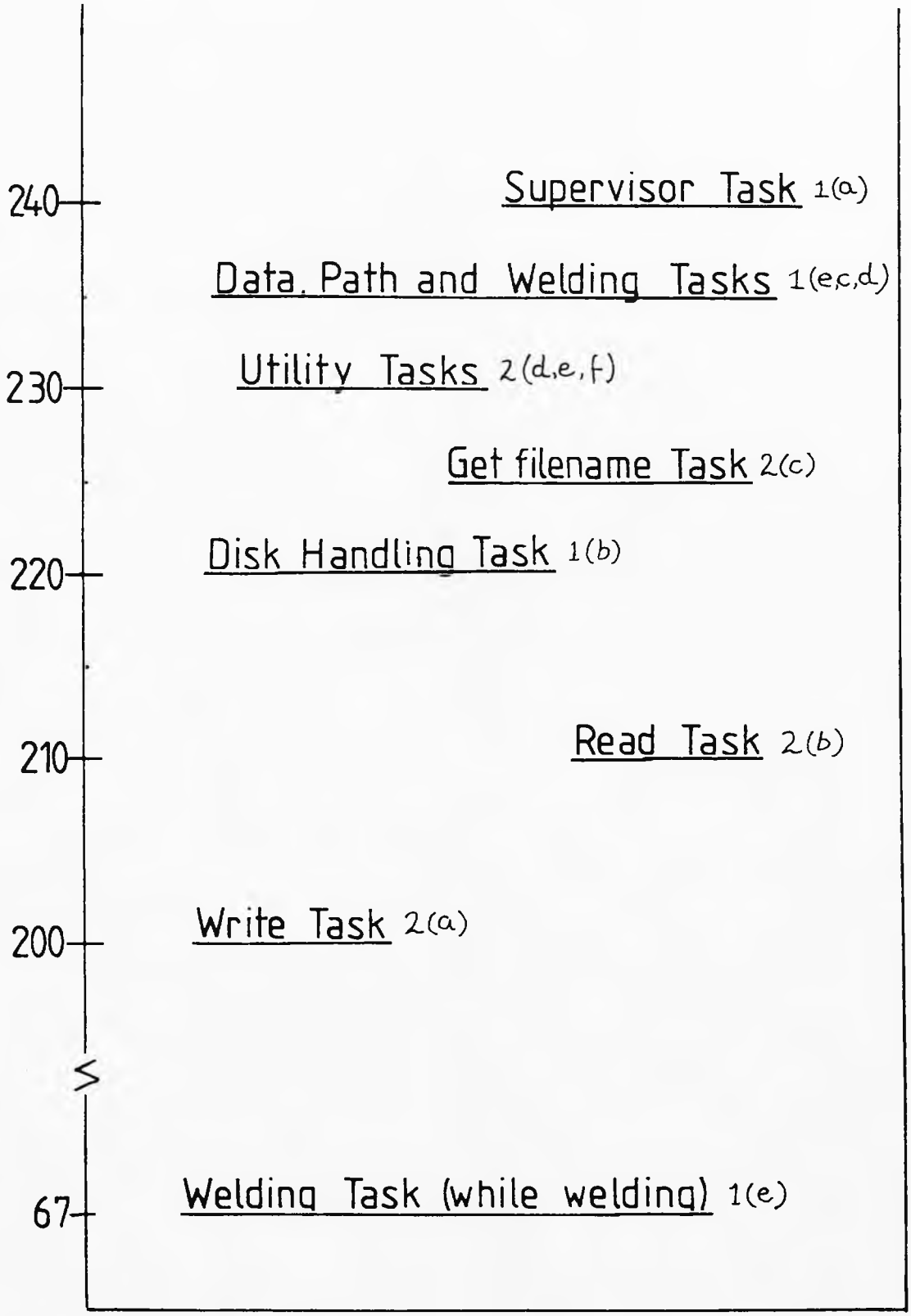


FIG4.1 TASK PRIORITY DIAGRAM

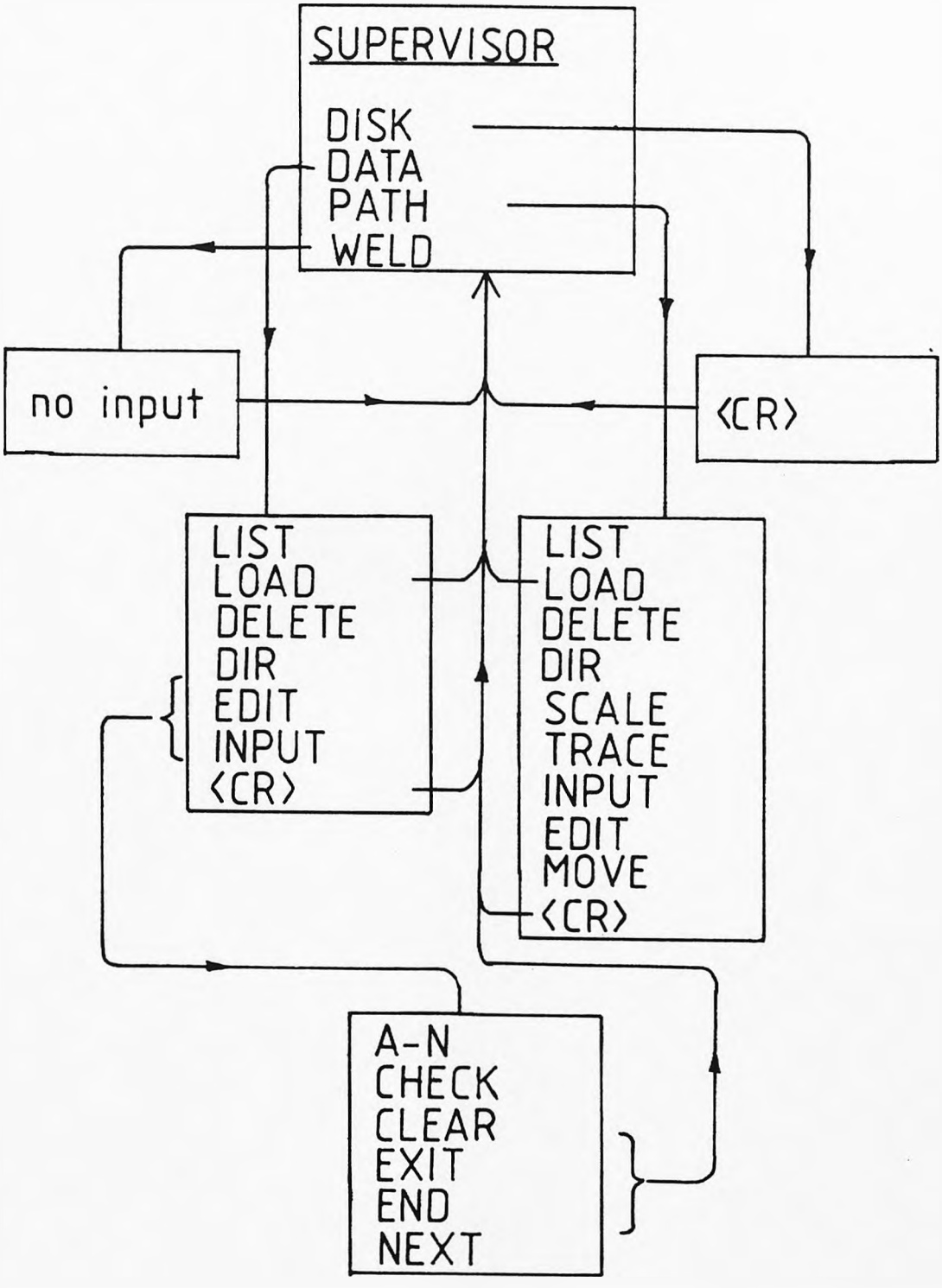
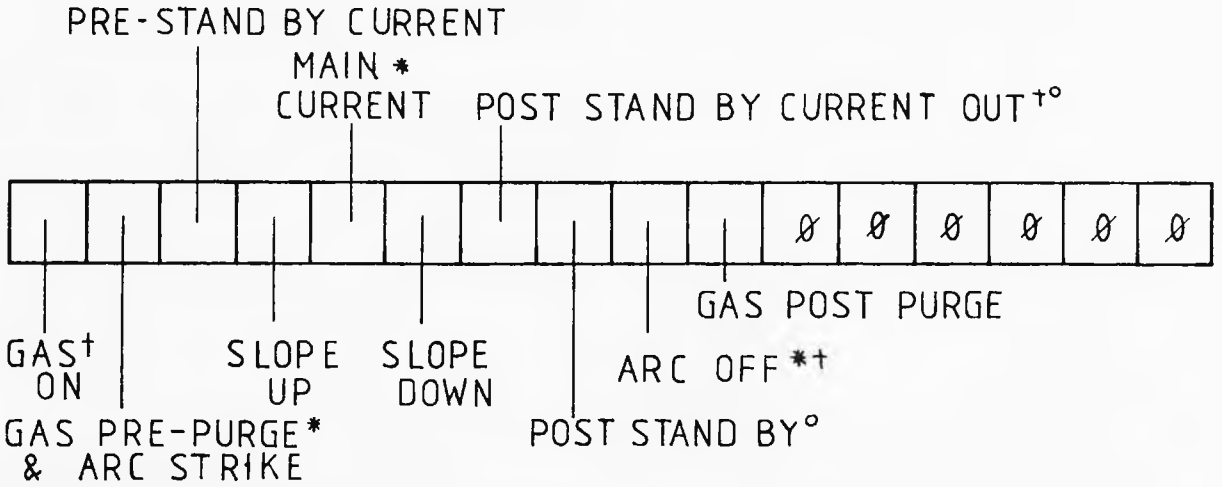


FIG 4.2 SYSTEM COMMANDS



* ALWAYS SET
† CALLED ONCE ONLY
Ø BIT MUST BE ZERO
° BOTH MUST BE SET

FIG.4.3 CURRENT ROUTINE FLAGS.

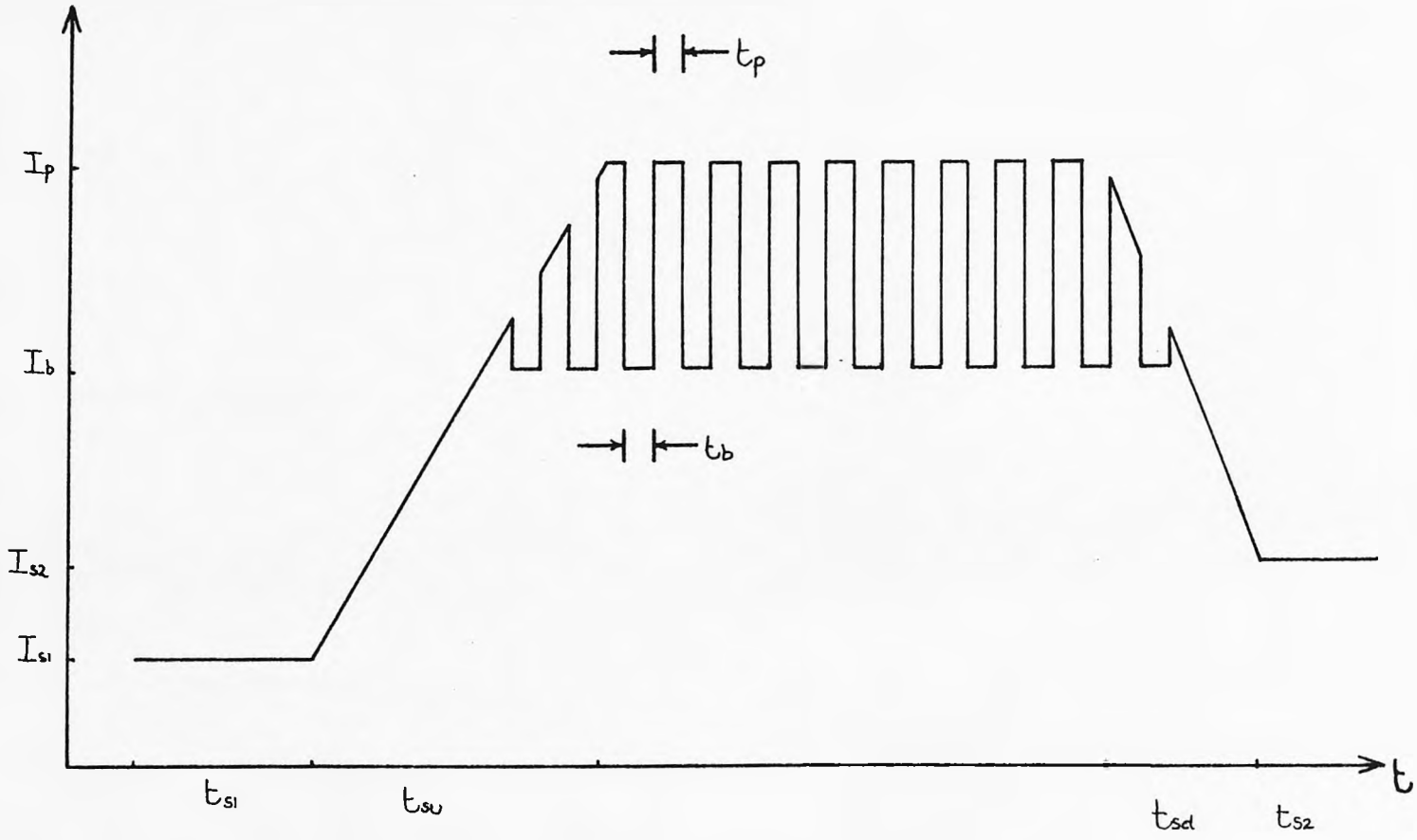


FIG. 4,4 FULL CURRENT WAVEFORM.

CHAPTER 5

SUPERWELDER OPERATING MANUAL

5.1. Starting Up.

On starting up the system the controller must be switched on first. Before the system is plugged in, the robot should be unplugged from the front of the unit and the robot drivers should be switched off. The system can then be plugged in and switched on. Once the terminal has warmed up a line of asterisks will be seen on the screen and an uppercase 'U' should be typed, the display should then look like this.

```
**************************U
iAPX 86, 88 MONITOR V1.0

-
```

The disk containing the main program should then be inserted into the lefthand drive and uppercase 'B <CR>' should then be typed. After a short time the disk drive unit will click on and start reading, this should last about 90 seconds. The terminal will then bleep three times indicating it is ready to be used, so the robot can then be plugged in.

The screen then consists of the SuperWeld heading at the top with a single command line at the bottom, this line should then read as follows.

```
LIVERPOOL UNIVERSITY SUPERWELDER
```

(empty screen)

DISK SUPERVISOR (Enter new disk).

The Disk Supervisor requires a correctly formatted data disk to be entered into the righthand drive (if a single drive unit is being used the systems disk should be removed and the data disk inserted). A single carriage return ('<CR>') should then be typed and the system will check for a correct disk. If it isn't correct the Disk Supervisor will request another disk.

5.2. SuperWelder

The system will then be ready to operate and be displaying the following line.

SUPERWELDER (DATA,PATH,WELD).

The SuperWelder is an overhead program which keeps the files of data and paths in order and checks all the hardware is in the correct state. The filing system is split into two, one side keeps the large path files in the main path directory, the other side is the main data directory which has a level of directories each of which contain one or more data files. This arrangement is done because the data files are usually great in number but are small, so they are sub-divided into directories (Fig.3.3). When dealing with PATH commands the system will give a list of file names and ask which one the command is to act upon or in some cases, ask for a new filename. Whereas the data commands firstly ask for a directory name to be

entered and then a file name.

The notation in this chapter is where a file name is required to be input the commands will be followed by <FN> (file name) or <FDN> (file and directory name), these files are expected to already exist. The symbols <NFN> and <NFDN> are new file names which may not already exist, otherwise an error will be reported and a new file name is again requested. The file or directory name may be up to nine alphanumeric characters long with an optional single decimal point or full stop. This point may not have more than three characters following it.

e.g.	Correct	Incorrect
	STEEL	STAINLESSSTEEL
	steel2.4mm	st&eel
	mildsteel	STEEL.12mm

On typing one of the choices given DATA, PATH or WELD, new choices are given as described in section 5.3-5.5.

5.3. Data Library Utilities

On requesting 'DATA' from the SuperWeld supervisor program the following line is displayed.

DATA LIBRARY UTILITIES(INPUT, EDIT, DIR, LIST, LOAD, DELETE)

Each of the commands will be dealt with in turn.

INPUT <NFDN>. As an example, the following would be typed in response to the systems prompts for the directory and file names.

Please enter new directory name. STEEL<CR>

Please enter new file name. 3.4mm<CR>

Once a new file has been opened, a blank set of data with a new set of commands is displayed as follows.

```
A:          PULSE CURRENT=    0.0 Amps
B:     BACKGROUND CURRENT=    0.0 Amps
C:          PULSE ON TIME=    0.0 Secs
D:         PULSE OFF TIME=    0.0 Secs
E:  PRE-STAND BY CURRENT=    20.0 Amps
F: POST-STAND BY CURRENT=    20.0 Amps
G:      PRE-STAND BY TIME=    0.0 Secs
H:     POST-STAND BY TIME=    0.0 Secs
I:          SLOPE UP TIME=    0.0 Secs
J:        SLOPE DOWN TIME=    0.0 Secs
K:     GAS PRE-PURGE TIME=    0.0 Secs
L:    GAS POST-PURGE TIME=    0.0 Secs
M:         TRAVERSE SPEED=    0.0 mm/Sec
N:    PENETRATION MONITOR=    0.0 on/off
                SET NUMBER=    1
```

DATA INPUT (A-N CHECK CLEAR END QUIT NEXT)

The data can then be adjusted, as required, by typing the letter of the parameter that needs to be changed followed by a space and the value required. **CHECK** does a simple error check on the data to test for simple errors such as the background current being larger than the pulse current etc. **CLEAR** will remove all the parameters input and return them to zero. **END** first does an error check on the data entered, if any errors are found then no futher action is taken. Otherwise the system stores the input file and returns to the data utilities question level. **QUIT** also returns to the data utilities question level but without storing the input data. **NEXT** allows the first set to be error checked and then a new clear set is displayed with the SET NUMBER set to two, so that several sets of different data may be stored in the same file. With each weld on a path a new set of data will be read from the file.

EDIT <FDN> <NFDN>. The data editor works in the same way as the data input routine except two files are requested,

one that already exists and one that does not. The existing file is then copied across to the new file and this new file is the one that can be edited. The prompt line reads as follows.

DATA EDITOR (A-N CHECK CLEAR END EXIT QUIT NEXT)

The functions are the same as for data input except EXIT. This function will copy across only the sets edited so far, the rest of the sets from the already existing file are discarded.

DIR <FDN>. This function allows the user to inspect the files that are contained in each directory, as usual a directory is selected and the files contained within are listed on the screen.

LIST <FDN>. This allows the dir function to be taken one step futher, such that a file is selected and its contents are listed upon the screen in the input data format. The data can then be inspected but not changed. The first set is automatically listed on the screen. The rest of the sets can be displayed by pressing 'space and return' and the next set will be listed. To exit this function press 'return' alone.

LOAD <FDN>. This allows the data files to be sent to the welding task. Before the file is sent the load command checks to see if any other files have already been sent, if so then these are returned so that only the last file loaded is used for welding. The load command assumes the data utilities have been finished with and so, returns the user to the SuperWeld level.

DELETE <FDN>. This command, as its name suggests, justs deletes the file named. If the directory the file was deleted from, is then empty, then the directory is also deleted. In this way empty directories cannot exist.

<CR>. This is the usual way to return to the SuperWeld level from the data utilities. A single carriage return is typed with no command.

5.4. The Path Library Utilities.

On requesting the path utilities from the SuperWeld level the following prompt is displayed.

PATH LIBRARY UTILITIES (INPUT SCALE DIR LIST LOAD DELETE EDIT MOVE TRACE)

The path library functions, to a large degree, use the hand held pendant, a diagram of which is shown in figure 5.1. The twelve motor keys drive the six motors, two direction keys for each motor. The eight function or special keys are not always enabled, depending on which command the key pendant is acting under. The speed keys, fast and slow, are enabled under the MOVE and INPUT commands and allow the user to change the speed that the robot moves. The exit key allows the user to exit immediately from the 'MOVE' and 'INPUT' commands if the circumstances are correct (see below). The other special keys are described in the commands they concern.

INPUT <FN>. After requesting a new unique filename, this command invites the user to input up to five lines of text. This is so the path may be described in full rather than just a filename, such things as date, job number and user may be entered. After this control is transfered to

the key pendant.

All the motor keys are enabled so the robot may then be driven over the path required and points may be placed on the path using the STEP key. Special points are placed to indicate where welds are to start and stop using the WELD key. Upon welding with the path, the robot will trace up to this point and continue welding until a second weld point is reached where welding ceases. In this way several welds can be placed on the path before the torch is returned to the origin. The robot may be driven around at speed or positioned very accurately using the speed keys to move up and down the speed scale. Once the path is complete the torch must be returned to the point where it started very accurately, to assist the operator in this operation, once the torch is within 2cm of where it started it will return itself when the EXIT key is pressed. The restriction of 2cm is enforced to reduce the chance of the key being pressed when the work piece is still between the torch and its origin. As soon as this is complete the system will file the path on the floppy disks and bleep to indicate the return of control to the terminal keyboard.

SCALE <FN> <NFN>. This function enables the user to copy paths that have already been input. At the same time the function asks for a scaling value in the form of a decimal number in the range 0.1 to 20.0 to be entered at the keyboard. If a scale of 1.0 is entered, then this will produce a perfect copy, else, each limb of the path is multiplied by the scaling value. Overflow is checked for

and if it occurs informs the user that path scaling is not possible with the scale value entered. Care should be taken in entering very small scale values as rounding down to integer step counts may result in the removal of short lines altogether.

DIR. This function gives the user a list of all the path files present on the disk in the righthand drive, no directory or file names are required.

LIST <FN>. List displays the text present in the file specified on the screen. Pressing any key will then result in a list of the line distances being displayed, each distance between the points is relative to the previous points (ie. they are all offsets) and is counted in motor steps.

	X	Y	Z	R0	R1	R2
	+08645	+00000	+00000	+00000	+00000	+00000
	+00000	+00000	+00000	-00400	+00000	+00000
WELD START	+00000	-08597	+00000	+00000	+00000	+00000
	+00100	+05670	+00000	+00000	+00000	+00000
	+00000	+00000	+00000	-00100	+00000	+00000
	+07689	-00129	+00000	+00000	+00000	+00000
WELD END	+00000	+05403	+00000	+00000	+00000	+00000
	-16434	+02347	+00000	+00000	+00000	+00000
	+00000	+00000	+00000	+00500	+00000	+00000
PATH END						

Path List Display

LOAD <FN>. This function is the same as the LOAD function in the data library functions. It removes any other path files already in the queue for welding and places the

specified file in the queue, so on the next WELD command this is the path file that will be welded with. As in the data utilities load this path load command returns the user to SuperWeld level assuming the path utilities are finished with.

DELETE <FN>. This removes the file named, from the floppy disk storage. Care should be taken with this command as once a file has been deleted it cannot be recovered.

EDIT <FN>. The edit function immediately transfers control to the key pendant whose keys are enabled in the following fashion. The STEP key causes the robot to move from the point it is presently stationary at, to the next point on the path. The key M1, if pressed while the torch is stationary at a point, enables the motor keys and allows the point to be moved. When the torch is over the position required, the key M2 should be pressed, disabling the motor keys. The robot can then be moved on to the next point by pressing the step key.

MOVE. This function enables all the motor keys and the speed keys and exit buttons to allow the robot to be repositioned wherever the user requires. The exit key stops the movement immediately. No storage or recording of the movement takes place.

TRACE <FN>. This command takes a path file and drives round the path with no user interaction at all, so the operator can inspect the accuracy of 'finished' paths.

<CR>. A simple carriage return with no command returns the user to the Superweld level.

5.5. The Welder.

The typing of 'WELD' will cause the system to immediately go and try and receive the two files required for welding. One data file and one path file must be present or the welder will immediately abort and return to the user. If both files are present then there should be no futher interaction and the system will weld as required.

5.6 An Example Superweld Session

The following is an exact copy of what could be entered at the terminal to create data and path files and then a weld may be done. It is assumed the system has been turned on and booted, the user should then see the bold type below and enter the lighter type exactly as shown.

DISK SUPERVISOR (Enter new disk).<CR>(Once the disk is in).

SUPERWELDER (DATA, PATH, WELD). DATA<CR>

DATA LIBRARY (INPUT,.....). INPUT<CR> STEEL<CR> 3.4mm<CR>

DATA INPUT (A-N CHECK .....). A 100<CR> M 2.2<CR> B 20<CR>
C .15<CR> D .15<CR> END<CR>

DATA LIBRARY (INPUT,.....). LOAD<CR> STEEL<CR> 3.4mm<CR>

SUPERWELDER (DATA, PATH, WELD). PATH<CR>

PATH LIBRARY (INPUT,.....). MOVE<CR> (Position the torch at the start of the weld with the key pendan

PATH LIBRARY (INPUT,.....). INPUT<CR> TESTPATH<CR> (Five lines of text may now be entered to describe the path and the torch is driven round the path, pressing the step key to place the points and using the weld key to specify where the weld starts and stops. Then return to the origin and press the exit key.)

PATH LIBRARY (INPUT,.....). SCALE<CR> TESTPATH<CR>
TESTPATH2<CR> 1.0<CR>

PATH LIBRARY (INPUT,.....). TRACE<CR> TESTPATH<CR>

PATH LIBRARY (INPUT,.....). LOAD<CR> TESTPATH<CR>

SUPERWELDER (DATA, PATH,WELD). WELD<CR>

This should then produce a weld as was described by the operator. It is suggested that operators should produce several data and path files using other functions such as the data editor before welding is attempted to familiarise themselves with the system.

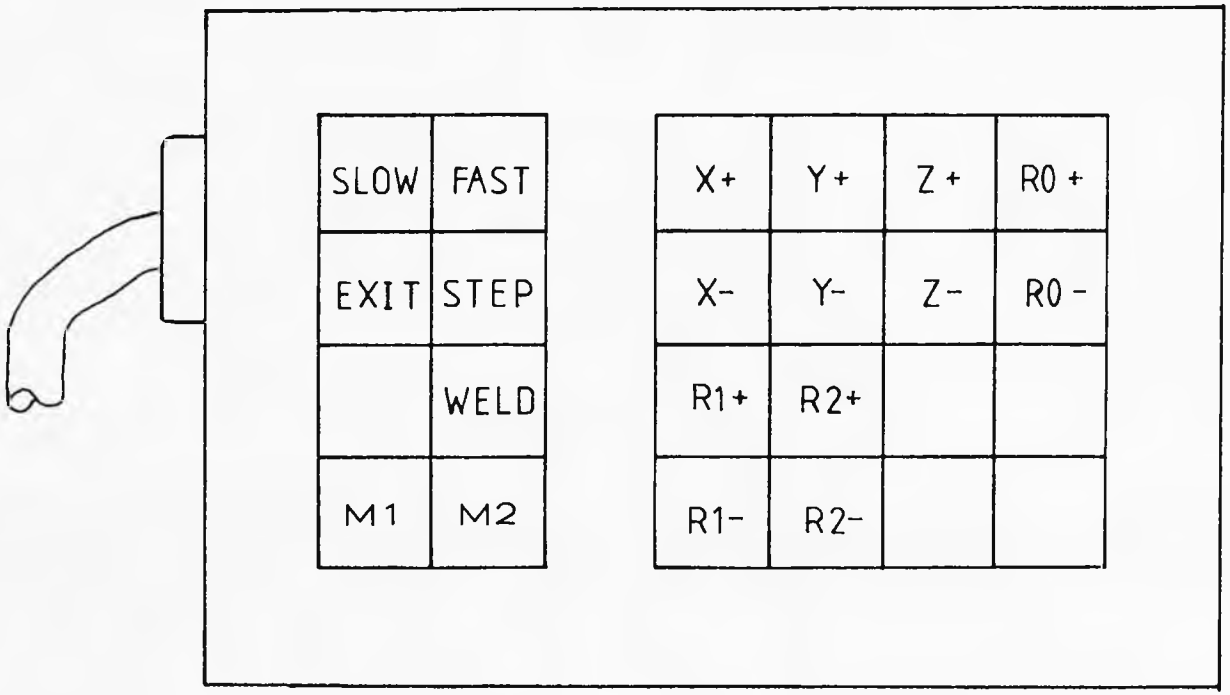


FIG. 5.1 KEY PENDANT.

CHAPTER 6.

CALIBRATION AND EXAMPLES.

6.1 Superwelder Calibration.

The diagrams following this chapter contain several current waveform traces and path traces from the calibration of the Superweld system.

6.1.1 Current Calibration.

The current traces, shown in figures 6.1-6.5, were taken with an XY plotter, with the X axis switched to the time base and the Y axis connected to the 0-5V output of the digital to analogue converter. They show the range of current waveforms available to the operator by varying the current parameters. With the plotter accurately calibrated we can use these plots to determine the accuracy of both the time and the current. Neither could be faulted though the speed of the current pulses had to be measured using a storage oscilloscope (Tektronix 7633) rather than using the plotter.

Figure 6.1 shows a full current trace. The current switches on at t=0 to 100 amps as specified and remains at this level for 1 second before rising at a constant rate to 500 amps. Pulsing starts as soon as the current passes the 200 amp mark which is correct as this is to be the background current. The current then pulses for the correct length of time (which is a function of the weld length). The current slope down takes the current back to precisely 100 amps at which it remains for 2 seconds until the arc is extinguished. Figure 6.2 shows a similar trace with the pulsing at a different rate and the pulse on time

at a different rate to that of the pulse off time. Figure 6.4 shows the same trace as figure 6.1 except the traverse velocity has been increased thereby reducing the overall weld time.

Figure 6.3 shows a more interesting trace with no slope up or down times. This produces an immediate rise for the stand-by current to the pulse current. The graph shows a slight slope in this time, but the error was checked using a storage scope and found to be because of the mechanical properties of the plotter rather than any error in the tracing. Figure 6.5 shows a current waveform with no pulsing. This enabled checking of the peak current value, as once again due to the mechanical latency of the plotter the peak was never reached by the pen with pulsing turned on.

6.1.2 Speed control and Path Calibration.

The speed control on the hand held pendant adjusts the counts sent to the 8253 PIT clocks which control the motors. These counts must be exact so as to produce an accurate number of motor steps per 10ms enabling them to be counted accurately. With incorrect counts either an extra short pulse is included in the 10ms or the last pulse is clipped, both producing an uneven and inaccurate travel of the robot. The number of pulses given for specific counts was therefore checked using an oscilloscope and where necessary the counts were tuned to give the exact number of pulses and produce smooth stepping by the motors.

The path traces, shown in figures 6.6-6.11, were taken with a recorder pen attached to the torch head against the paper on the workpiece bed. All the points listings are given in incremental offsets, calibrated in motor steps. On the path traces the solid lines indicate weld seams and the dashed lines are pre and post-weld tracking. The points are marked as follows ;-

O = path origin

WS = weld start

WE = weld end

In this way two dimensional paths could be plotted on to paper and repeatedly measured for accuracy against the original plot over which the torch was driven. The path matrix saved could also be checked using the list function in the path utilities, against the number of steps we expected to be executed for a given distance.

Figure 6.6 shows a square weld with overlapping weld ends. The square could then be replayed and the travel of the torch checked. Figure 6.7 shows a simple straight line weld. This can easily be related to the points graph showing the use of 5 points along the straight line. Figures 6.8 and 6.9 show the use of the scaling utility. The first octagon was plotted in and then scaled by a factor of two to a second path file. Both were then traced out and measured. The step coordinates can also be compared checking the scaling factor of two.

Figures 6.10 and 6.11 show two things. Firstly the way a circular path can be plotted using relatively few points (32 in this case) and secondly the down scaling.

The second circle is a scaled down copy of the first by a factor of two. It can be seen from the coordinates that the rounding down is working correctly.

In addition to these path tests the robot was driven over large distances in each direction. The distance was then checked against the step count, this enabled checking for motor slipping due to the work load on the motors.

6.2 Welding Trials.

The photographs at the end of the chapter show trials using the TIM robot and the resultant welds which can be attained.

The first photograph is of a simple bead on plate weld. In a bead on plate welding procedure the workpiece is simply melted by the welding arc and allowed to cool again forming the weld, no other metal such as filler wire was added. The plate is a decorative shield with the letters LU printed apon it. Three welds are used, one for the shield border, one for the 'L' and one for the 'U' but only one path is used. Therefore three sets of start and stop welding instructions are placed on the path. In this case only one set of data was entered forcing the weld proceedure to reuse this single set for all three welds. Three sets of different parameters could have been entered resulting in three different style seams. The plate was 1.4mm thick so a 50A pulse current was used with a 20 Amp background. The velocity of the torch was 2mm/sec and the pulses where 2 seconds in width.

Normal welding requires the use of simple shapes as

most structures are either tube or box like. These standard shapes have various sizes but may be stored in a standard shape file and scaled as required. Photograph two shows three standard shapes used by a industry for butt welding tubes. Each of these paths are stored in a standard file and scaled as required.

Two welds are shown in the next photograph, the bottom one has been done on the Superweld system and the top was welded by hand. Both are single pass welds but marked diffences can be seen in the two. The hand welded seam is rough and uneven because an expert welder is not able to move the torch at a constant velocity or keep the torch height constant. As a result the path is not straight and the bead sizes are uneven creating structural weaknesses in the seam. The weld surface is dull indicating the flow of argon over the plate was not even due to irregular torch movement resulting in oxidation of the molten metal. This produces a dull weld and will need milling over and polishing followed by a very detailed inspection for defects such as cracks, bubbles and lack of penetration all of which may render the weld as defective.

The bottom photograph however shows the same weld done by the Superweld system. The beads are regular, the weld is bright and will need little cleaning. Inspection is much easier with the weld being so uniform as defects stand out.

The above welds were all done using a 50 Amp Polypack power supply designed and built by the Welding Institute.

The welding parameters used were supplied by The Welding Institute. Many such linear welding trials were carried out, the parameters of these are shown in table 6.1. These are standard TIG welding parameters for three types of steel, Mild, Nimonic and Stainless.

Photograph four shows a flange plate being welded to a square tube. The pieces are shown assembled and prepared for welding in the top picture. The joint surfaces have been cleaned and an exact fit produced between the two components. In this way the components can be welded without any filler wire thus reducing costs. Using the TIM robot this weld can be repeated many times with no reprogramming of the system giving a fast turn round and good tolerance of reproduction. The weld done on the square tube and flange plate is a single pass, the weld ends are overlapping with current slope up and down giving a neat and clean finish. As can be seen from the bottom photograph the final product's weld area needs little if any milling and cleaning.

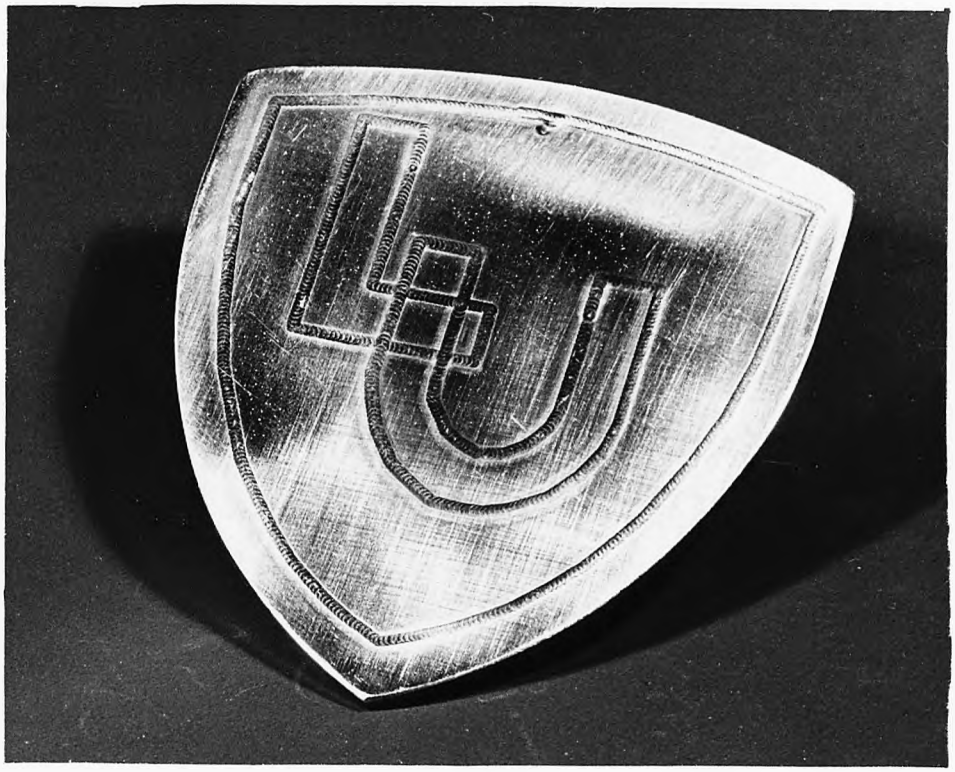
The final photograph shows the effect of using a penetration monitor. The top photograph shows the front and back faces of a welded component that is even in thickness. The penetration of the weld pools on the backface is even and matches the frontface. This is without a weld penetration monitor and works fine on even thickness workpieces and with little metal variation between similar components. The lower photographs however show what happens with a tapered component. The current

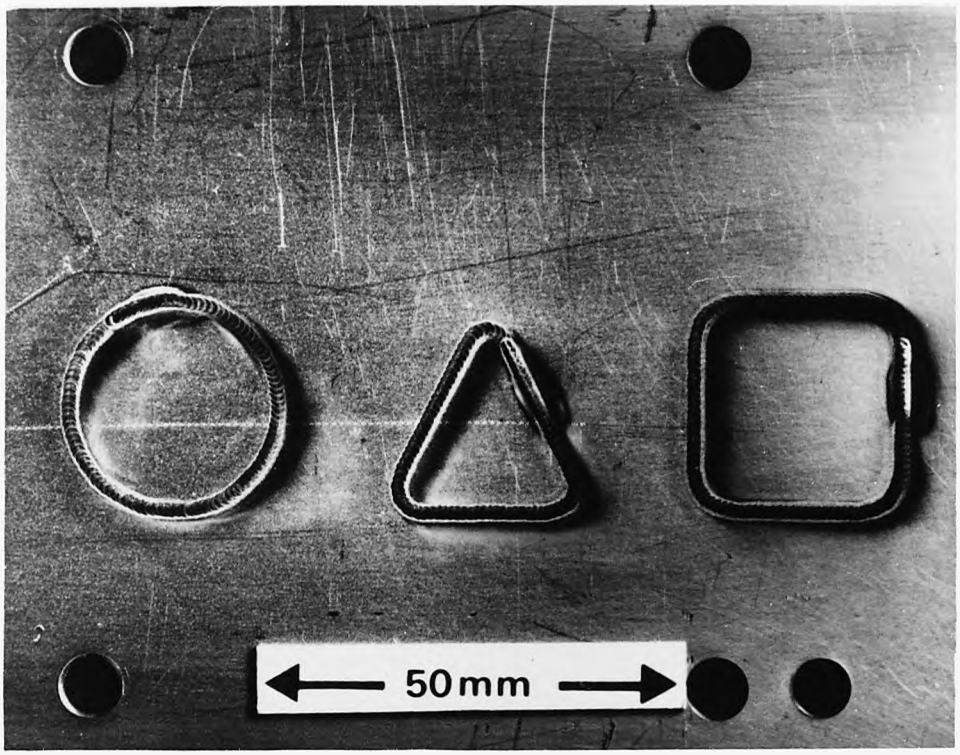
parameters were kept constant and the torch an even distance from the workpeice. However the component was 0.7mm thick at one end and 3.3mm at the other. Without the weld penetration monitor a peak current of 140 Amps was used from a AWP GEC 300E power supply and it can be seen that the weld penetration tapers as the metal thickness changes. Over penetration occurs at the thin end and normal penetration occurs at the thick end. With the weld penetration monitor attached though, even penetration is always attained. The seam width remains fixed and the beads can clearly be seen to be regular and constant in size. Cast to cast variation also affects the hardness and structure of the metal making it harder or easier to melt, usually a nominal current of 140 Amps was required but with some casts up to 200 Amps were required to achieve the same amount of penetration.

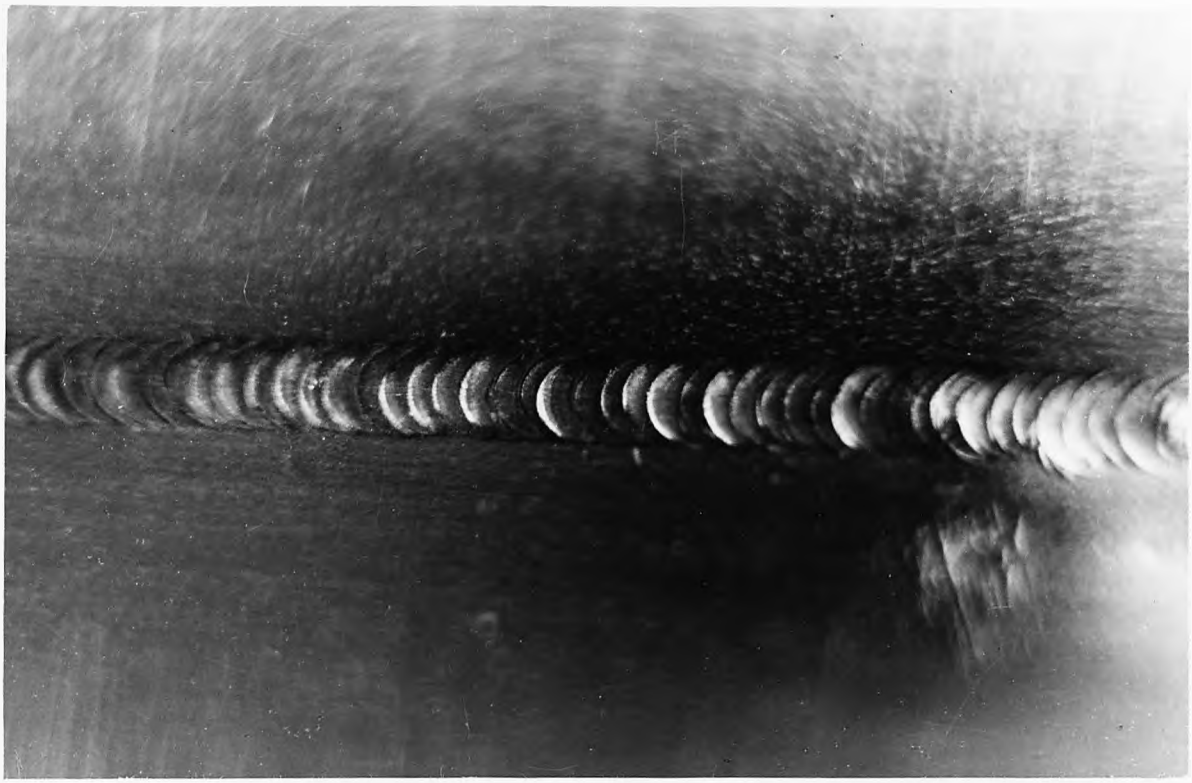
No	Material	Thickness	Ip	Ib	tp	tb	Velocity
99.9% Argon							
0	Stainless	0.8mm	25	10	0.4	0.4	0.05
1		1.0mm	35	10	0.4	0.4	0.05
2		1.2mm	50	10	0.4	0.4	0.05
3		1.4mm	50	10	2.0	2.0	0.018
4	Mild	1.0mm	45	10	0.4	0.4	0.05
5		1.4mm	50	10	2.7	2.7	0.008
6	Nimonic	0.8mm	28	10	0.8	0.8	0.025
7		1.2mm	50	10	0.4	0.4	0.05
Argon/5% Hydrogen							
8	Stainless	0.8mm	25	10	0.2	0.2	0.1
9		1.0mm	35	10	0.2	0.2	0.1
10		1.2mm	50	10	0.2	0.2	0.1
11		1.4mm	50	10	1.5	1.5	0.025
12	Mild	1.0mm	45	10	0.2	0.2	0.1
13		1.4mm	50	10	2.2	2.2	0.01
14	Nimonic	0.8mm	28	10	0.2	0.2	0.1
15		1.2mm	50	10	0.4	0.4	0.05

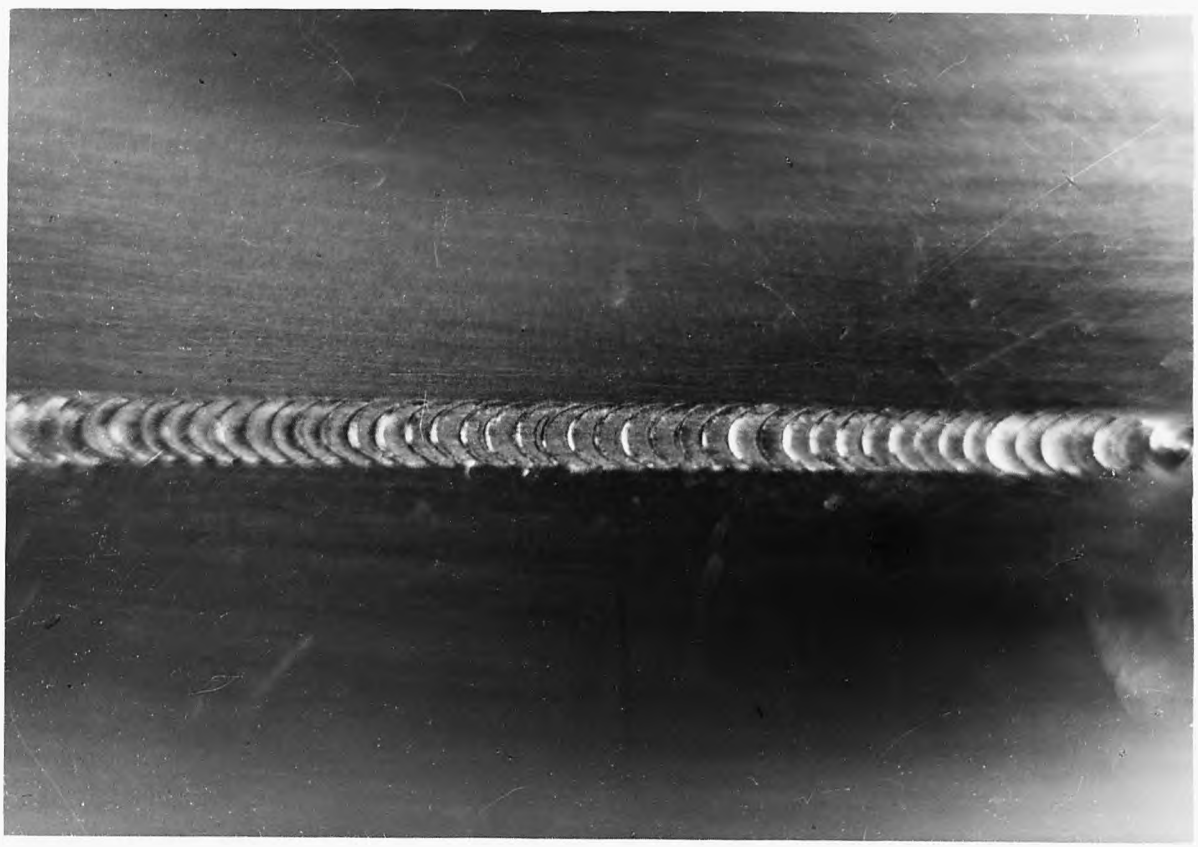
All currents given in Amps, times in seconds and velocity in metres per minute.

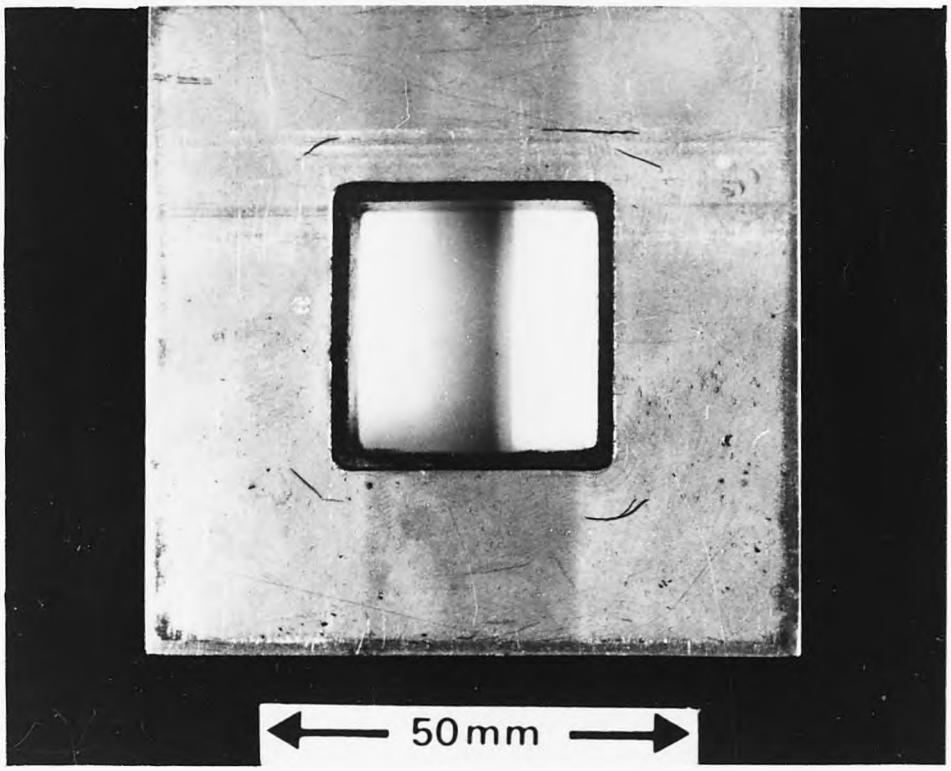
Table 6.1 Standard TIG Welding Parameter Sets

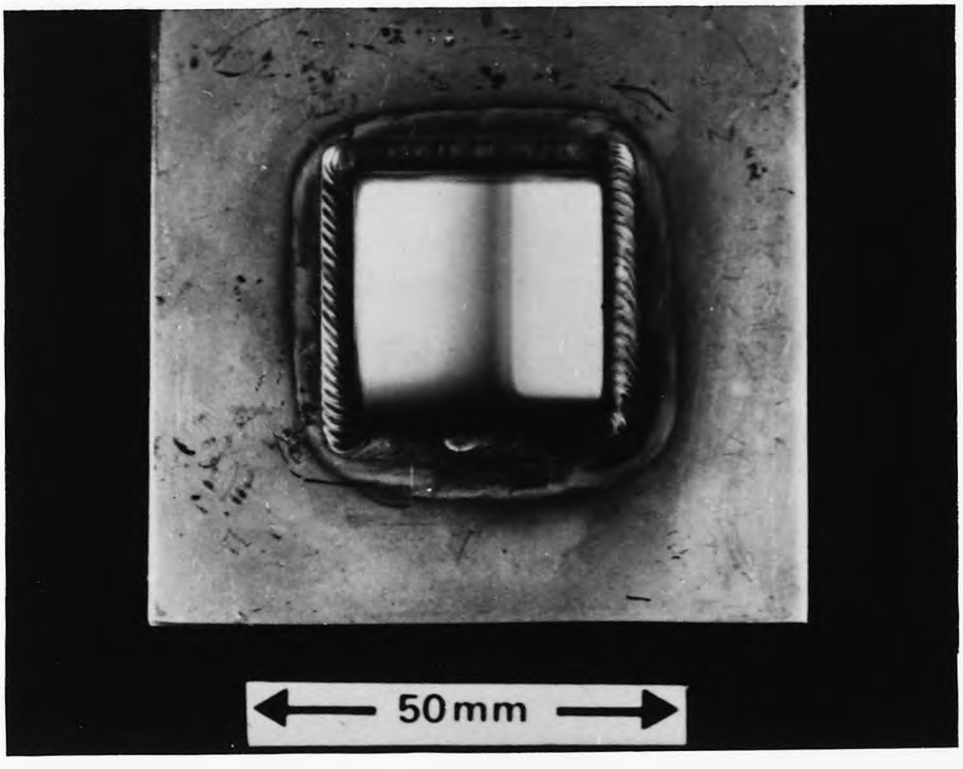


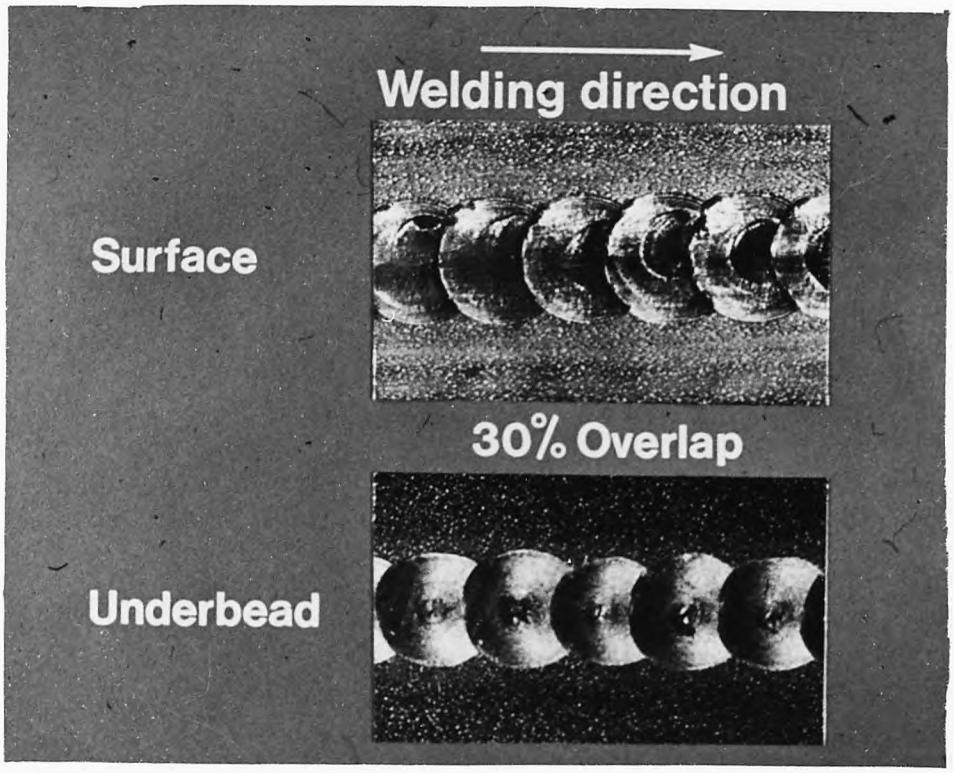

50mm

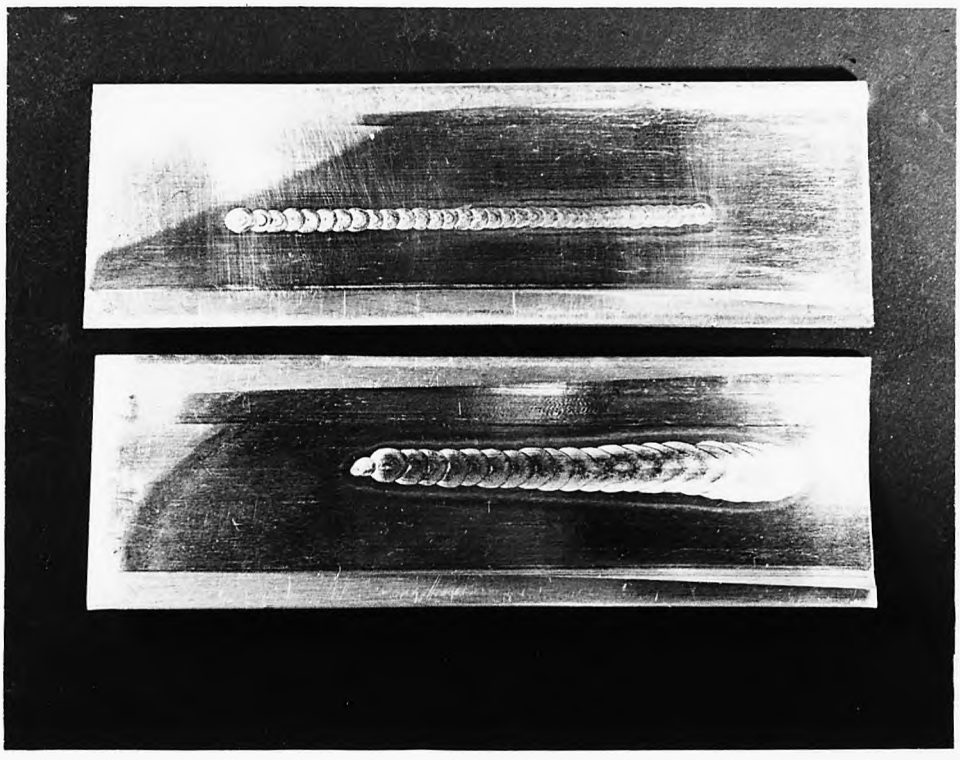


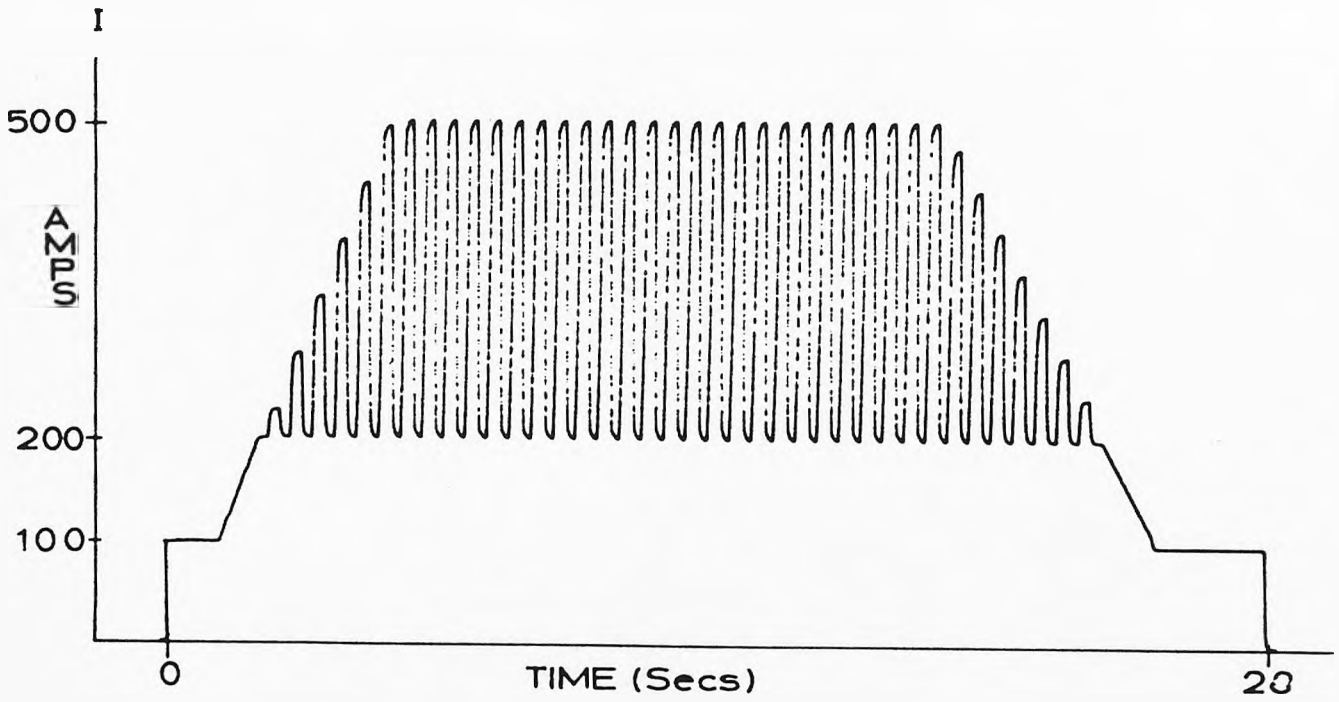



50mm


50mm

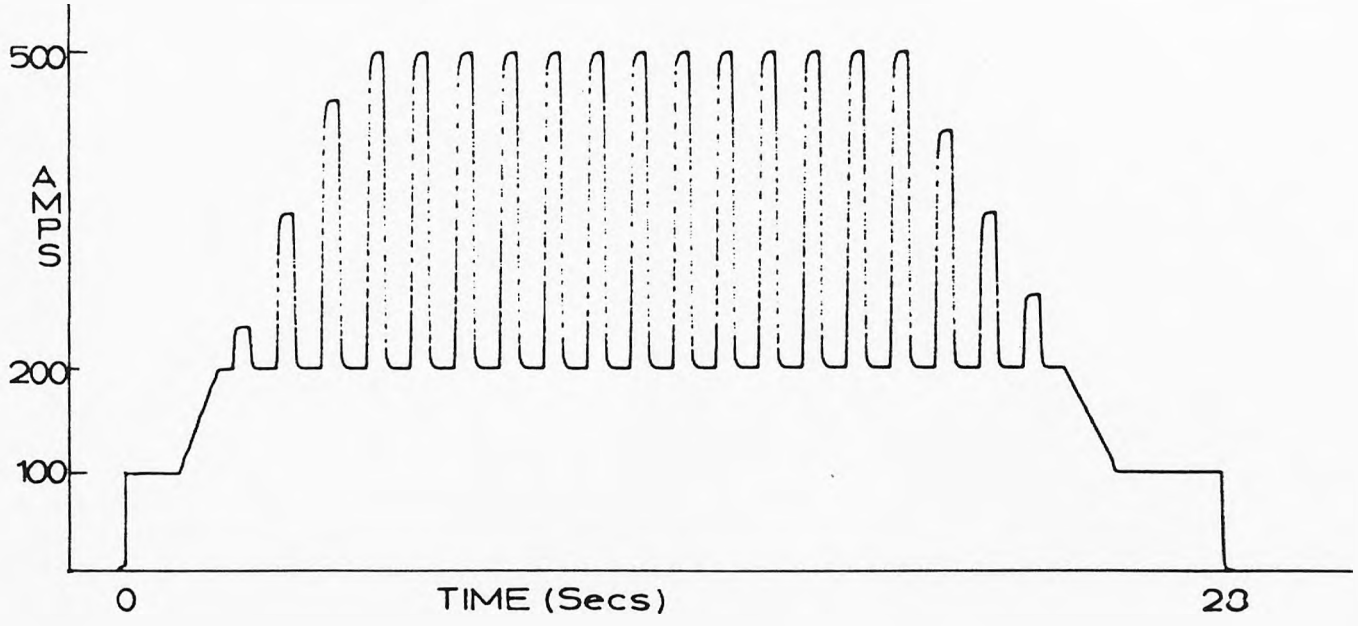

Welding direction
Surface
30% Overlap
Underbead





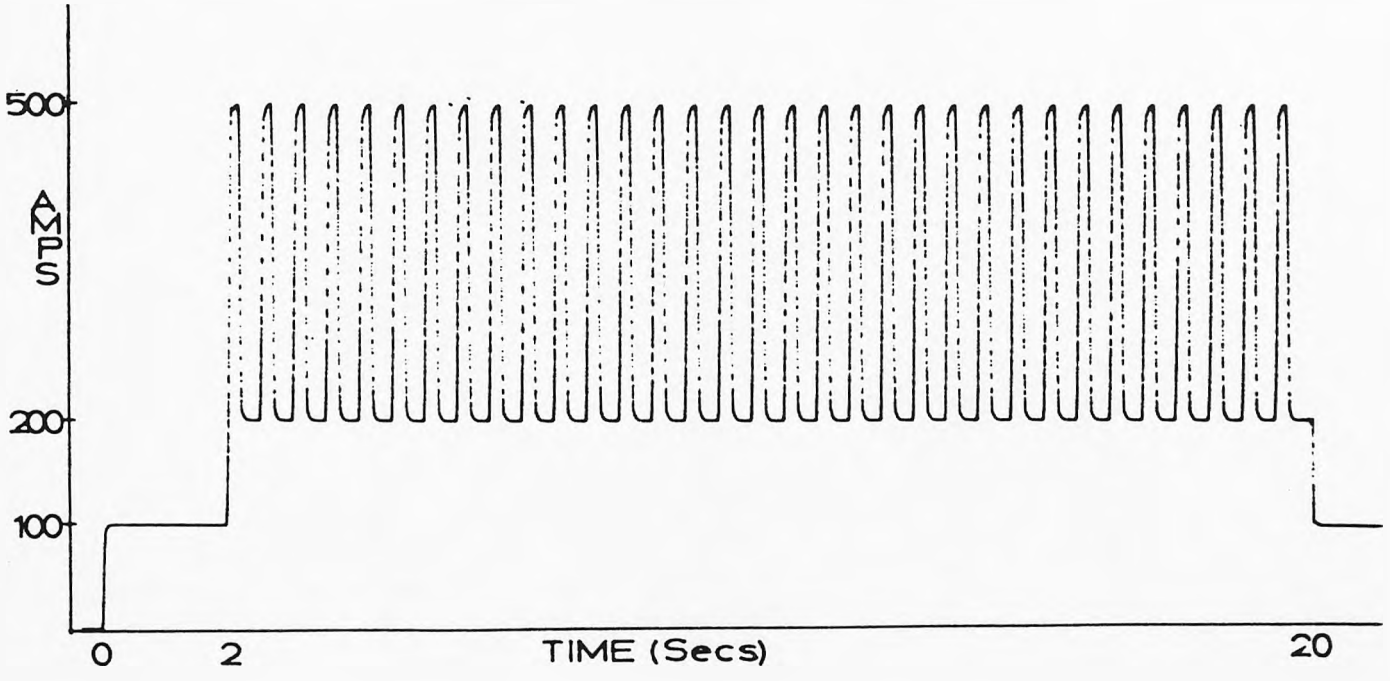
A:	PULSE CURRENT=+	500.0	Amps
B:	BACKGROUND CURRENT=+	200.0	Amps
C:	PULSE ON TIME=+	0.2	Secs
D:	PULSE OFF TIME=+	0.2	Secs
E:	PRE-STAND BY CURRENT=+	100.0	Amps
F:	POST-STAND BY CURRENT=+	100.0	Amps
G:	PRE-STAND BY TIME=+	1.0	Secs
H:	POST-STAND BY TIME=+	2.0	Secs
I:	SLOPE UP TIME=+	3.0	Secs
J:	SLOPE DOWN TIME=+	4.0	Secs
K:	GAS PRE-PURGE TIME=+	0.0	Secs
L:	GAS PORT-PURGE TIME=+	0.0	Secs
M:	TRAVERSE SPEED=+	1.6	mm/Sec
N:	PENETRATION MONITOR=+	0.0	on/off
	SET NUMBER=	1	

FIG. 6.1 FULL CURRENT WAVEFORM DESCIPTION AND TRACE.



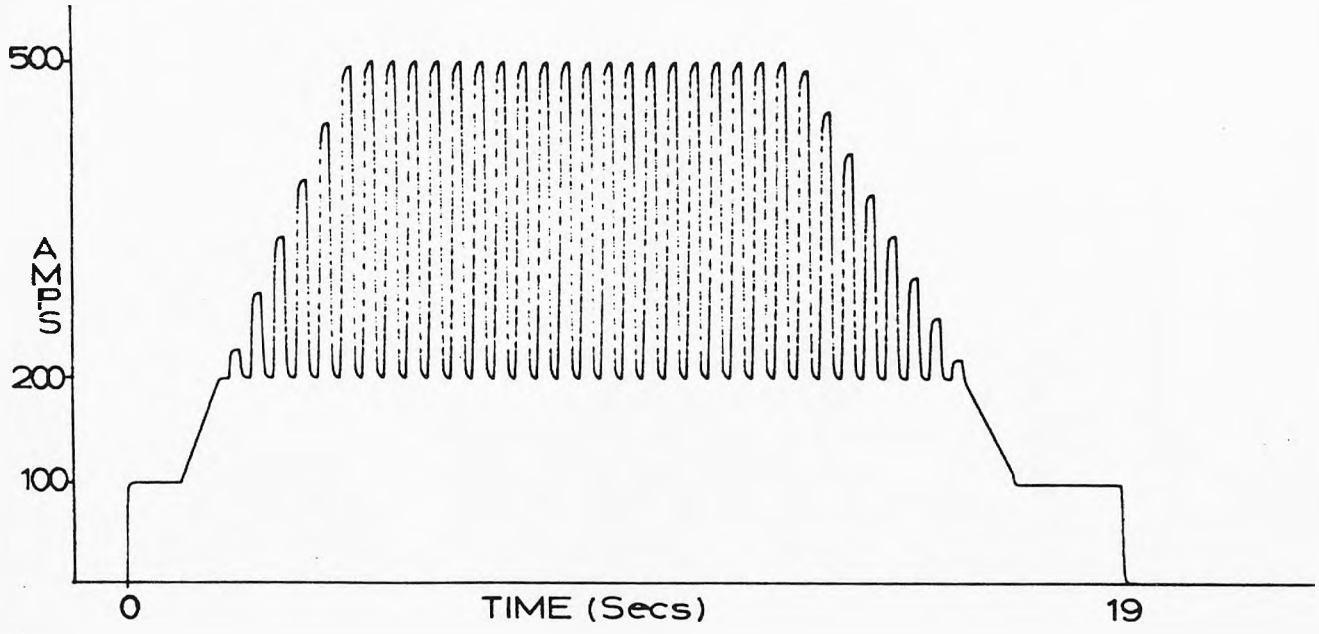

```
A:          PULSE CURRENT=+500.0 Amps
B:    BACKGROUND CURRENT=+200.0 Amps
C:          PULSE ON TIME=+  0.3 Secs
D:         PULSE OFF TIME=+  0.5 Secs
E: PRE-STAND BY CURRENT=+100.0 Amps
F:POST-STAND BY CURRENT=+100.0 Amps
G:     PRE-STAND BY TIME=+  1.0 Secs
H:    POST-STAND BY TIME=+  2.0 Secs
I:          SLOPE UP TIME=+  3.0 Secs
J:        SLOPE DOWN TIME=+  4.0 Secs
K:    GAS PRE-PURGE TIME=+  0.0 Secs
L:   GAS PORT-PURGE TIME=+  0.0 Secs
M:         TRAVERSE SPEED=+  1.6 mm/Sec
N:   PENETRATION MONITOR=+  0.0 on/off
                SET NUMBER=    1
```

FIG. 6.2 FULL CURRENT WAVEFORM WITH LARGER PULSE TIMES.




```
A:          PULSE CURRENT=+500.0 Amps
B:    BACKGROUND CURRENT=+200.0 Amps
C:          PULSE ON TIME=+  0.2 Secs
D:         PULSE OFF TIME=+  0.4 Secs
E: PRE-STAND BY CURRENT=+100.0 Amps
F:POST-STAND BY CURRENT=+100.0 Amps
G:     PRE-STAND BY TIME=+  2.0 Secs
H:    POST-STAND BY TIME=+  3.0 Secs
I:          SLOPE UP TIME=+  0.0 Secs
J:        SLOPE DOWN TIME=+  0.0 Secs
K:    GAS PRE-PURGE TIME=+  0.0 Secs
L:   GAS PORT-PURGE TIME=+  0.0 Secs
M:         TRAVERSE SPEED=+  1.6 mm/Sec
N:   PENETRATION MONITOR=+  0.0 on/off
                SET NUMBER=    1
```

FIG. 6.3 CURRENT WAVEFORM WITHOUT SLOPE PARAMETERS.



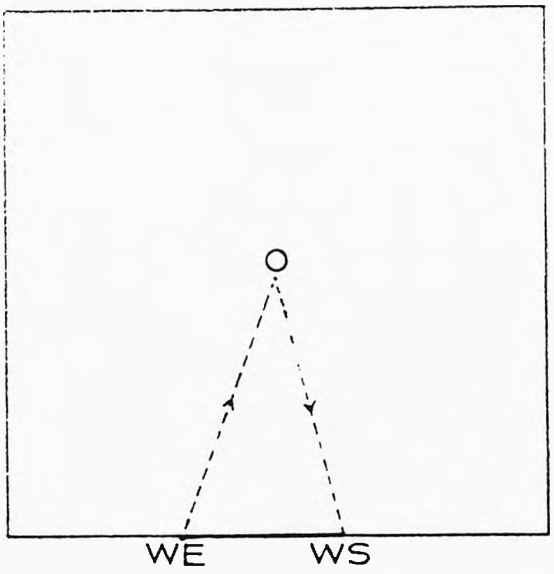
A:	PULSE CURRENT=+	500.0	Amps
B:	BACKGROUND CURRENT=+	200.0	Amps
C:	PULSE ON TIME=+	0.2	Secs
D:	PULSE OFF TIME=+	0.2	Secs
E:	PRE-STAND BY CURRENT=+	100.0	Amps
F:	POST-STAND BY CURRENT=+	100.0	Amps
G:	PRE-STAND BY TIME=+	1.0	Secs
H:	POST-STAND BY TIME=+	2.0	Secs
I:	SLOPE UP TIME=+	3.0	Secs
J:	SLOPE DOWN TIME=+	4.0	Secs
K:	GAS PRE-PURGE TIME=+	0.0	Secs
L:	GAS PORT-PURGE TIME=+	0.0	Secs
M:	TRAVERSE SPEED=+	2.0	mm/Sec
N:	PENETRATION MONITOR=+	0.0	on/off
	SET NUMBER=	1	

FIG 6.4 CURRENT WAVEFORM WITH INCREASED VELOCITY GIVING A SHORTER SEAM TIME.

A:	PULSE CURRENT=	+500.0	Amps
B:	BACKGROUND CURRENT=	+ 0.0	Amps
C:	PULSE ON TIME=	+ 0.0	Secs
D:	PULSE OFF TIME=	+ 0.0	Secs
E:	PRE-STAND BY CURRENT=	+100.0	Amps
F:	POST-STAND BY CURRENT=	+100.0	Amps
G:	PRE-STAND BY TIME=	+ 3.0	Secs
H:	POST-STAND BY TIME=	+ 3.0	Secs
I:	SLOPE UP TIME=	+ 3.0	Secs
J:	SLOPE DOWN TIME=	+ 4.0	Secs
K:	GAS PRE-PURGE TIME=	+ 0.0	Secs
L:	GAS PORT-PURGE TIME=	+ 0.0	Secs
M:	TRAVERSE SPEED=	+ 1.0	mm/Sec
N:	PENETRATION MONITOR=	+ 0.0	on/off
	SET NUMBER=	1	

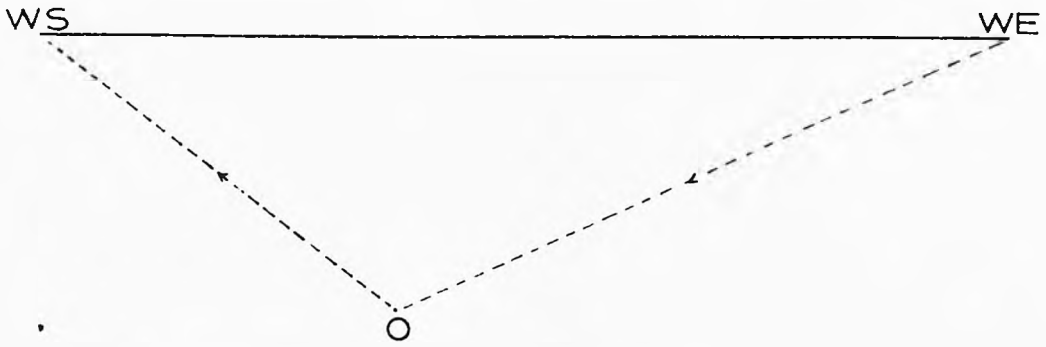
Note: Background Current is set to 0 not 500A

FIG 6.5 CURRENT WAVEFORM WITHOUT PULSE PARAMETERS GIVING A D.C. TIG WELD.



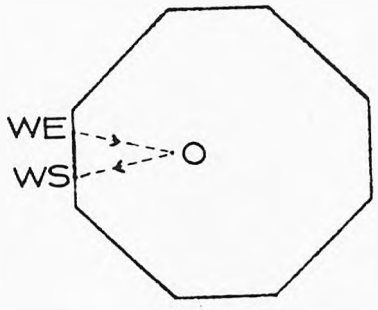
	X	Y	Z	R0	R1	R2
	+03514	+00959	+00000	+00000	+00000	+00000
	-00056	+00063	+00609	+00000	+00000	+00000
WELD START	+00000	-03829	+00000	+00000	+00000	+00000
	-07007	+00000	+00000	+00000	+00000	+00000
	+00000	+05642	+00000	+00000	+00000	+00000
	+06993	+00000	+00000	+00000	+00000	+00000
	+00000	-03521	+00000	+00000	+00000	+00000
WELD STOP	+00000	+00000	-00651	+00000	+00000	+00000
	-03500	+00854	+00000	+00000	+00000	+00000
	+00056	-00168	+00042	+00000	+00000	+00000
PATH END						

FIG. 6.6 SQUARE PATH TRACE WITH OVERLAPPING WELD ENDS.



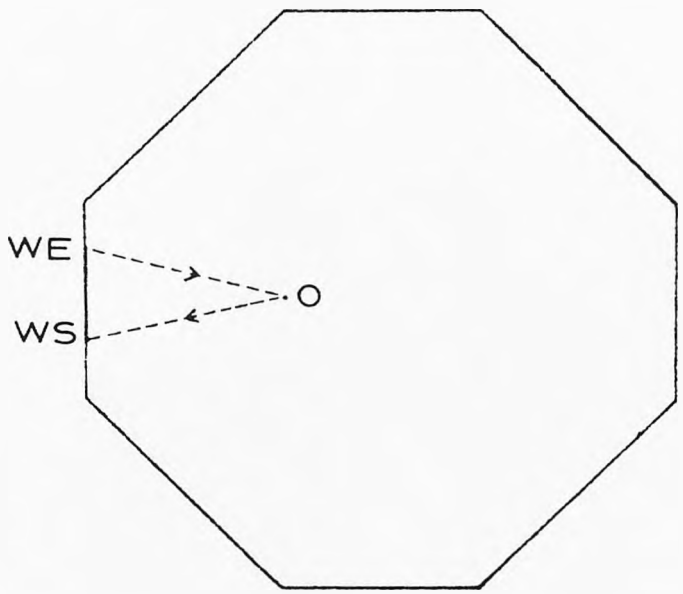
	X	Y	Z	R0	R1	R2
	-03654	+03691	+00000	+00000	+00000	+00000
	+00000	+00105	+00546	+00000	+00000	+00000
WELD START	+00000	-02870	+00000	+00000	+00000	+00000
	+00000	-01512	+00000	+00000	+00000	+00000
	+00000	-02814	+00000	+00000	+00000	+00000
	+00000	-02912	+00000	+00000	+00000	+00000
WELD STOP	+00000	+00000	-00525	+00000	+00000	+00000
	+03773	+06377	+00000	+00000	+00000	+00000
	-00119	+00007	-00021	+00000	+00000	+00000
PATH END						

FIG. 6.7 STRIAGHT LINE WELD.



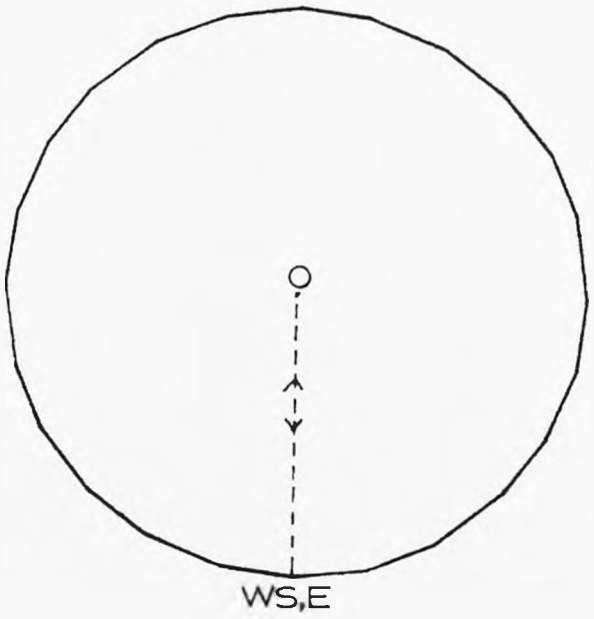
	X	Y	Z	R0	R1	R2
	+00287	+01001	+00000	+00000	+00000	+00000
	+00000	+00026	+00322	+00000	+00000	+00000
WELD START	-00906	+00006	+00000	+00000	+00000	+00000
	-01284	-01038	+00000	+00000	+00000	+00000
	+00000	-01026	+00000	+00000	+00000	+00000
	+01296	-01002	+00000	+00000	+00000	+00000
	+01296	+00000	+00000	+00000	+00000	+00000
	+01254	+01014	+00000	+00000	+00000	+00000
	+00000	+01032	+00000	+00000	+00000	+00000
	-01278	+01014	+00000	+00000	+00000	+00000
	-00996	+00000	+00000	+00000	+00000	+00000
WELD STOP	+00000	+00000	-00294	+00000	+00000	+00000
	+00336	-00798	+00000	+00000	+00000	+00000
	-00005	-00229	-00028	+00000	+00000	+00000
PATH END						

FIG. 6.8 OCTAGONAL WELD WITH OVERLAPPING WELD ENDS.



	X	Y	Z	R0	R1	R2
	+00574	+02002	+00000	+00000	+00000	+00000
	+00000	+00052	+00644	+00000	+00000	+00000
WELD START	-01812	+00012	+00000	+00000	+00000	+00000
	-02568	-02076	+00000	+00000	+00000	+00000
	+00000	-02056	+00000	+00000	+00000	+00000
	+02592	-02004	+00000	+00000	+00000	+00000
	+02592	+00000	+00000	+00000	+00000	+00000
	+02508	+02028	+00000	+00000	+00000	+00000
	+00000	+02064	+00000	+00000	+00000	+00000
	-02556	+02028	+00000	+00000	+00000	+00000
	-01992	+00000	+00000	+00000	+00000	+00000
WELD STOP	+00000	+00000	-00588	+00000	+00000	+00000
	+00672	-01596	+00000	+00000	+00000	+00000
	-00010	-00458	-00056	+00000	+00000	+00000
PATH END						

FIG. 6.9 OCTAGONAL WELD SCALED UP BY A FACTOR OF 2.



	X	Y	Z	R0	R1	R2
	+00000	+03003	+00000	+00000	+00000	+00000
	-00014	+00000	+00161	+00000	+00000	+00000
WELD START	+01068	-00130	+00000	+00000	+00000	+00000
	+00805	-00273	+00000	+00000	+00000	+00000
	+00802	-00497	+00000	+00000	+00000	+00000
	+00595	-00612	+00000	+00000	+00000	+00000
	+00388	-00763	+00000	+00000	+00000	+00000
	+00126	-00763	+00000	+00000	+00000	+00000
	-00136	-00798	+00000	+00000	+00000	+00000
	-00357	-00700	+00000	+00000	+00000	+00000
	-00704	-00704	+00000	+00000	+00000	+00000
	-00704	-00416	+00000	+00000	+00000	+00000
	-00934	-00308	+00000	+00000	+00000	+00000
	-00934	-00084	+00000	+00000	+00000	+00000
	-01106	+00126	+00000	+00000	+00000	+00000
	-00808	+00270	+00000	+00000	+00000	+00000
	-00774	+00500	+00000	+00000	+00000	+00000
	-00598	+00623	+00000	+00000	+00000	+00000
	-00406	+00802	+00000	+00000	+00000	+00000
	-00112	+00704	+00000	+00000	+00000	+00000
	+00150	+00805	+00000	+00000	+00000	+00000
	+00368	+00718	+00000	+00000	+00000	+00000
	+00679	+00682	+00000	+00000	+00000	+00000
	+00696	+00416	+00000	+00000	+00000	+00000
	+00942	+00308	+00000	+00000	+00000	+00000
	+00934	+00102	+00000	+00000	+00000	+00000
WELD STOP	+00000	+00000	-00308	+00000	+00000	+00000
	+00000	-03017	+00000	+00000	+00000	+00000
	+00035	+00007	+00147	+00000	+00000	+00000
PATH END						

FIG. 6.10 32 POINT CIRCULAR WELD.

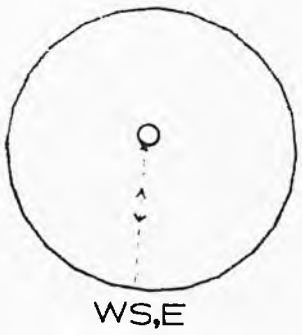


FIG. 6.11 CIRCULAR WELD SCALED DOWN BY A FACTOR OF 2.

CHAPTER 7.

FUTURE RESEARCH AND CONCLUSIONS.

This research project has demonstrated the practicability of controlling the TIG welding process to an industrial standard. The accuracy of both power supply control and robot positioning is above that required. This has been proven by the calibration tests in Chapter 6.

The application is able to control and 'teach' the TIM robot an unlimited variety of paths using the linearly interpolated three dimensional technique.

The system has proven the technique of using a single computer to control both parameter storage and the welding operation. The basic system is at present open loop during welding. Here the welding is being carried out according to a predefined path and with preset welding parameters. Present and future research is investigating the possibility of closing the loop with various monitors running online.

We have produced a system which is completely expandable in every way. The system is non-dedicated to the TIM robot, any linear system could be driven, with one, two or three main axis and up to three minor axes. The unit was produced using the Intel multibus and iSBC system, enabling more dedicated processor boards to be added if necessary. Already a high speed vision system is being developed using more powerful 80286 processor boards that will eventually be placed in the same rack as the controller when completed. Other peripherals may also now be developed, such as seam penetration monitors, on the

multibus system thereby producing an expandable kit system for all welding requirements.

This system was predominantly designed for TIG welding but with MIG welding in mind. The system however is also flexible enough to cope easily with pulsed MIG, a new welding technique where droplets of filler wire are dripped fast onto the workpiece to form overlapping pools like those seen with TIG welding. This show the immediate versatility of the system.

The 8086 processor we have used copes with the work load of welding quite adequately with some room for expansion (processing power left for future peripherals), though if need be a larger 80286 or 80386 processor board may be used with no changes to the coding as it is fully transportable across the Intel processor range.

Figure 7.1 shows the research present and future. This project has taken the place of both computers A &B and produced a single unit. The ultimate welding system the group is aiming to produce contains a hybrid of both open and closed loop units. Here the controller operates with the path and data predefined, but this is updated during the weld time by taking into account the information from external sensors. The system would then also have to revert back to open loop when it was considered that the situation encountered might cause the sensors to be unreliable.

The multitasking system developed now needs to be expanded to cater for the peripherals being attached.

These included seam trackers [6], penetration monitors [7,8,9] and power supply systems all being developed at present within the Industrial Electronics Group at Liverpool University.

The coding to communicate to these peripherals may either be inserted in the present code or be written as separate tasks. Communication techniques and protocols are also being looked at to provide an external welding system bus. This will enable intelligent sensors and peripherals to communicate without talking through the main controller. Present ideas included both parallel buses and local area networks [31].

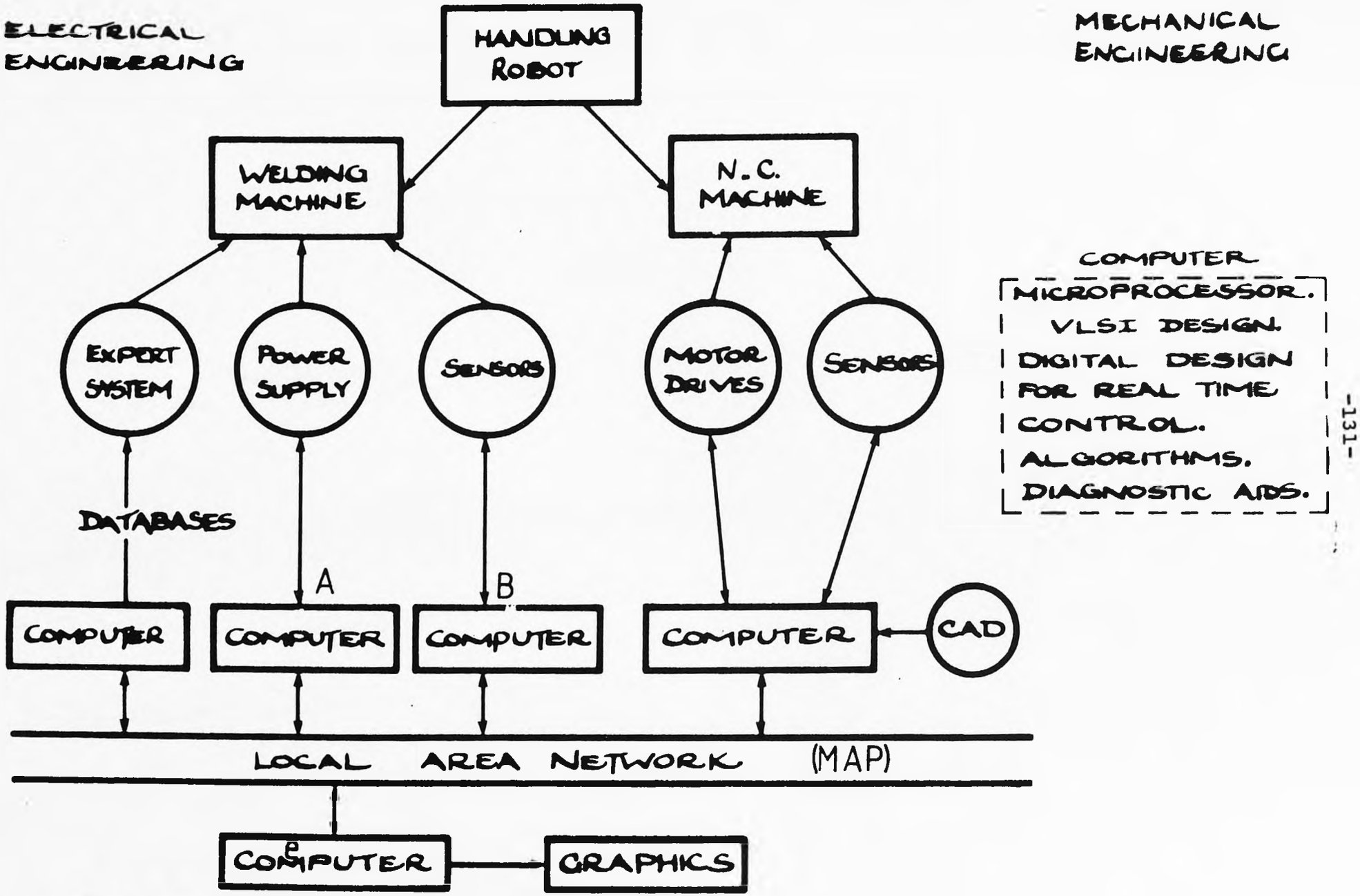


FIGURE. 7.1 The Automated Welding System

1. MORRIS, E., "An Investigation into the Automation of TIG (Tungsten Inert Gas) Welding", PhD Thesis, Liverpool University.
2. LANCASTER, J.F., Metallurgy of Welding, Allen and Unwin, 1980.
3. LUCAS, W., "TIG and Plasma Welding in the 80's", Metal Construction, 1982, Vols. 9,10,11,12., The Journal of the Welding Institute, Abington Hall, Cambridge, CB1 6AL.
4. SLOAN, K., and LUCAS, J., "Microprocessor Control of TIG Welding System", Proc. I.E.E., 1981, 129, Pt.E., pp. 1-8.
5. Digiplan Drives, Type 1054. Unimatic Engineers Ltd., Cricklewood, London, NW2 2LN.
6. CLARK, S., and LUCAS, J., "Seam Trackers for TIG welding", Proc. I.E.E., Vol 132, Pt.D,No. 4, July 1985.
7. TAN, C., and LUCAS, J., "Low Cost Sensors for Seam Tracking in Arc Welding"., First International Conference on Computer Technology in Welding., 1986.
8. SMITH, J.S., PARKER, A.B., and LUCAS, J., "A Vision based seam tracker for TIG welding"., First International Conference on Computer Technology in Welding., London., 1986.
9. AINSCOUGH, D.A., and LUCAS, J., "Penetration Monitors for TIG welding" ., ACMA Directorate Grantholders Conference., Royal Holloway College., 1984.
10. iCS 80 Industrial Chassis Hardware Reference Manual.,

Order No. 9800799-02. Intel Corporation (UK) Ltd., Swindon, Wilts., SN1 1BJ.

11. iSBC 640 Power Supply Hardware Reference Manual., Order No. 9800803-02. Intel.
12. iSBC 604/614 Cardcage Hardware Reference Manual., Order No. 9800708A. Intel.
13. iSBC 86/14 and iSBC 86/30 Single Board Computer Hardware Reference Manual., Order No. 144044-002. Intel.
14. iSBC 337 Multimodule Numeric Data Processor Hardware Reference Manual., Order No. 142887-001., Intel.
15. OEM Systems Handbook., Order No. 210941-003., Intel. pp 5-11.
16. iSBX 328 Analogue Output Multimodule Board Hardware Reference Manual., Order No. 142914-002., Intel.
17. iCS 910 Analogue Signal Conditioning/Termination Panel Hardware Reference Manual., Order No. 9800800A., Intel Corporation (UK) Ltd., Swindon, Wilts, SN1 1BJ.
18. iCS 920 Digital Signal Conditioning/Termination Panel Hardware Reference Manual., Order No. 9800801-02. Intel.
19. iSBC 208 Floppy Disk Controller Board Hardware Reference Manual., Order No. 98-143078., Intel.
20. iSBC 012B Memory Board Technical Manual., Order No. 112748., Intel.
21. Microsystem Components Handbook, Vol. 2., Chpt. 5., pp. 229-240., Order No. 230843-002., Intel.

22. System 86/330A Installation and Maintenance Manual., Order No. 144777-001., Intel

23. Introduction to the iRMX 86 Operating System., Order No. 9803124-04., Intel.

24. iRMX 86 Nucleus Reference Manual., Order No. 98031122-04., Intel.

25. iRMX 86 Release 5 Basic I/O System Reference Manual., Order No. 172766-001., Intel.

26. iRMX 86 Release 5 Extended I/O System Reference Manual., Order No. 172767-001., Intel.

27. iRMX 86 Human Interface Reference Manual., Order No. 9803202-03., Intel.

28. Getting Started With Release 5 iRMX 86 System., Order No. 145073-001., Intel.

29. iRMX 86 Release 5 System Debug Monitor Reference Manual., Order No. 172763-001., Intel.

30. iRMX 86 Release 5 Configuration Guide., Order No. 172765-001., Intel.

31. WILSON, J. N.,"Local Area Networks for Retail Applications.", PhD Thesis, University of Liverpool, Electronic and Electrical Engineering Dept., 1984.

APPENDIX

SUPERWELD SOFTWARE.

1. SUPERVISOR ROOT TASK.

```
;RMX 86 PL/M-86 V2.3 COMPILATION OF MODULE SUPERVISOR
OBJECT MODULE PLACED IN MAIN/CLI.OBJ
COMPILER INVOKED BY:  :LANG:PLM86 MAIN/CLI.P86


              $TITLE ('SUPERWELD SUPERVISOR TASK VER 86/2.0 DATE 24/05/83')

              $DEBUG
              $LARGE
              $OPTIMIZE(0)

   1          SUPERVISOR: DO;

              $INCLUDE (INC/LITERALS.P86)
              $SAVE NOLIST

              $INCLUDE(/INC/RMX86/NCRMBX.EXT)
              $SAVE NOLIST
              $INCLUDE(/INC/RMX86/NGTTOK.EXT)
              $SAVE NOLIST
              $INCLUDE(/INC/RMX86/NSNMES.EXT)
              $SAVE NOLIST
              $INCLUDE(/INC/RMX86/NCRSEG.EXT)
              $SAVE NOLIST
              $INCLUDE(/INC/RMX86/NLUOBJ.EXT)
              $SAVE NOLIST
              $INCLUDE(/INC/RMX86/NSTEXH.EXT)
              $SAVE NOLIST
              $INCLUDE(/INC/RMX86/NRCMES.EXT)
              $SAVE NOLIST
              $INCLUDE(/INC/RMX86/NCRSEM.EXT)
              $SAVE NOLIST
              $INCLUDE(/INC/RMX86/NSNUNI.EXT)
              $SAVE NOLIST
              $INCLUDE(/INC/RMX86/NCRTSK.EXT)
              $SAVE NOLIST
              $INCLUDE(/INC/RMX86/NEINIT.EXT)
              $SAVE NOLIST
              $INCLUDE(/INC/RMX86/NCTOBJ.EXT)
              $SAVE NOLIST
              $INCLUDE(/INC/RMX86/NDLSEG.EXT)
              $SAVE NOLIST
              $INCLUDE(/INC/RMX86/ICRUSR.EXT)
              $SAVE NOLIST
              $INCLUDE(/INC/RMX86/IAOPEN.EXT)
              $SAVE NOLIST
              $INCLUDE(/INC/RMX86/IWTIO.EXT)
              $SAVE NOLIST

              $EJECT
```

```
                /************************** LOCALS ******************************************/

144   1         declare (status,gen_word,type_code)                  word;
145   1         declare (dir_mailbox,file_mailbox,disk_connection)          token;
146   1         declare (data_mailbox,path_mailbox,gen_mailbox,main_task_token)     token;
147   1         declare (write_task_token,read_task_token,path_sem,data_sem)        token;
148   1         declare (th_in_mbx,th_out_mbx,read_in,read_out,weld_sem,disk_sem)   token;
149   1         declare com_seg_ptr                              pointer;
150   1         declare com_seg_vals structure (offset word,base word) AT (@com_seg_ptr);
151   1         declare (wrt_buf based com_seg_ptr)(80)                byte;
152   1         declare (read_buf based com_seg_ptr) structure (count byte,chars(80) byte);

153   1         declare title1(*) byte data (0,0,36,ESC,'#','3','    Liverpool Welding Controller.');
154   1         declare title2(*) byte data (20,0,29,'SUPERWELDER (DATA,PATH,WELD).');

                $EJECT
```

```
                /****************************** EXTERNALS ************************************/

155   1         DATA_TASK: procedure external;
156   2         END DATA_TASK;

157   1         PATH_TASK: procedure external;
158   2         END PATH_TASK;

159   1         WELD_TASK: procedure external;
160   2         END WELD_TASK;

161   1         WRITE: procedure external;
162   2         END WRITE;

163   1         READ: procedure external;
164   2         END READ;

165   1         LIST: procedure external;
166   2         END LIST;

167   1         GET_NAME: procedure external;
168   2         END GET_NAME;

169   1         CREATE_A_FILE: procedure external;
170   2         END CREATE_A_FILE;

171   1         CONNECT_FILE: procedure external;
172   2         END CONNECT_FILE;

173   1         FLOPPY_SECURITY_TASK: procedure external;
174   2         END FLOPPY_SECURITY_TASK;

                $EJECT
```

```
175   1       CREATE_MBX_SEMS: PROCEDURE;

              /* Catalogue this tasks token so others may look it up. */
176   2       main_task_token=RQ$GET$TASK$TOKENS(this_task,@status);
177   2       call RQ$CATALOG$OBJECT(this_job,main_task_token,@(4,'MTSK'),@status);

              /* The path mailbox is for passing path file tokens to the welding task. */
178   2       path_mailbox=RQ$CREATE$MAILBOX(fifo,@status);
179   2       call RQ$CATALOG$OBJECT(this_job,path_mailbox,@(4,'PMBX'),@status);

              /* The data mailbox is for passing data file tokens to the welding task. */
180   2       data_mailbox=RQ$CREATE$MAILBOX(fifo,@status);
181   2       call RQ$CATALOG$OBJECT(this_job,data_mailbox,@(4,'DMBX'),@status);

              /* The general mailbox is used mainly as a response mailbox by all the tasks
                 main uses are collecting segments as a result of any set information call.*/
182   2       gen_mailbox=RQ$CREATE$MAILBOX(fifo,@status);
183   2       call RQ$CATALOG$OBJECT(this_job,gen_mailbox,@(4,'GMBX'),@status);

              /* The following semaphores will all be waited at by their respective tasks.
                 To start the task running therefore all we need to do is to send a unit
                 to the correct semaphore.                                    */
184   2       data_sem=RQ$CREATE$SEMAPHORE(null,one,fifo,@status);
185   2       call RQ$CATALOG$OBJECT(this_job,data_sem,@(4,'DSEM'),@status);
186   2       path_sem=RQ$CREATE$SEMAPHORE(null,one,fifo,@status);
187   2       call RQ$CATALOG$OBJECT(this_job,path_sem,@(4,'PSEM'),@status);
188   2       weld_sem=RQ$CREATE$SEMAPHORE(null,one,fifo,@status);
189   2       call RQ$CATALOG$OBJECT(this_job,weld_sem,@(4,'WSEM'),@status);
190   2       disk_sem=RQ$CREATE$SEMAPHORE(null,one,fifo,@status);
191   2       call RQ$CATALOG$OBJECT(this_job,disk_sem,@(7,'DISKSEM'),@status);

192   2       END CREATE_MBX_SEMS;

              $EJECT
```

```
193   1       CREATE_TASKS: PROCEDURE;

              /* This procedure creates the data,path and weld tasks and then catalogues
                 them in the Jobs object directory. All the tasks have nucleus allocated
                 stack and include the 8087 processor option. */

194   2       declare (data_task_token,path_task_token,weld_task_token) token;

195   2       data_task_token=RQ$CREATE$TASK(data_priority,@DATA_TASK,no_data_seg,
                                 nuc_allocated_stk,stk_600_bytes,inc_8087,
                                 @status);
196   2       call RQ$CATALOG$OBJECT(this_Job,data_task_token,@(4,'DTSK'),@status);

197   2       path_task_token=RQ$CREATE$TASK(path_priority,@PATH_TASK,no_data_seg,
                                 nuc_allocated_stk,stk_600_bytes,inc_8087,
                                 @status);
198   2       call RQ$CATALOG$OBJECT(this_Job,path_task_token,@(4,'PTSK'),@status);

199   2       weld_task_token=RQ$CREATE$TASK(weld_priority,@WELD_TASK,no_data_seg,
                                 nuc_allocated_stk,stk_600_bytes,inc_8087,
                                 @status);
200   2       call RQ$CATALOG$OBJECT(this_Job,weld_task_token,@(4,'WTSK'),@status);

201   2       END CREATE_TASKS;

              $EJECT
```

```
202   1       CREATE_TH: PROCEDURE;

              /* This procedure creates the terminal read and write tasks and then
                 looks up the mailboxes they have created to allow the passage of
                 memory segment between them and this task. The required memory segments
                 are then created and the screen header is written and the screen cleared.
              */

              /*   Create both the read & write tasks    */
203   2       write_task_token=RQ$CREATE$TASK(write_priority,@WRITE,no_data_seg,
                              nuc_allocated_stk,stk_600_bytes,no_8087,
                              @status);
204   2       call RQ$CATALOG$OBJECT(this_job,write_task_token,@(6,'THWTSK'),@status);
205   2       read_task_token=RQ$CREATE$TASK(read_priority,@READ,no_data_seg,
                              nuc_allocated_stk,stk_600_bytes,no_8087,
                              @status);
206   2       call RQ$CATALOG$OBJECT(this_job,read_task_token,@(6,'THRTSK'),@status);
              /*   Look up their mailboxes     */
207   2       th_in_mbx=RQ$LOOK$UP$OBJECT(this_job,@(7,'THINMBX'),wait_forever,@status);
208   2       th_out_mbx=RQ$LOOK$UP$OBJECT(this_job,@(8,'THOUTMBX'),wait_forever,@status);
209   2       read_in=RQ$LOOK$UP$OBJECT(this_job,@(6,'READIN'),wait_forever,@status);
210   2       read_out=RQ$LOOK$UP$OBJECT(this_job,@(7,'READOUT'),wait_forever,@status);
              /*   Create this tasks write & read buffers   */
211   2       com_seg_vals.base=RQ$CREATE$SEGMENT(80,@status);
212   2       com_seg_vals.offset=null;
              /*   The following writes the screen header   */
213   2       call MOVB(clear_all_screen,@wrt_buf,6);
214   2       call RQ$SEND$MESSAGE(th_in_mbx,com_seg_vals.base,th_out_mbx,@status);
215   2       com_seg_vals.base=RQ$RECEIVE$MESSAGE(th_out_mbx,wait_forever,no_response,@status);
216   2       call MOVB(@title1,@wrt_buf,39);
217   2       call RQ$SEND$MESSAGE(th_in_mbx,com_seg_vals.base,th_out_mbx,@status);
218   2       com_seg_vals.base=RQ$RECEIVE$MESSAGE(th_out_mbx,wait_forever,no_response,@status);
219   2       wrt_buf(0)=2;
220   2       wrt_buf(5)='4';
221   2       call RQ$SEND$MESSAGE(th_in_mbx,com_seg_vals.base,th_out_mbx,@status);
222   2       com_seg_vals.base=RQ$RECEIVE$MESSAGE(th_out_mbx,wait_forever,no_response,@status);

223   2       END CREATE_TH;

              $EJECT
```

```
224   1       CREATE_UTILS: PROCEDURE;

              /* This procedure creates all the utility tasks which in order of
                 creation lists directories, gets valid file names from the console,
                 create files and attaches files to the task.
              */

225   2       declare (list_dir_task,cre_file_task,gname_task,att_file_task)   token;

226   2       list_dir_task=RQ$CREATE$TASK(utils_priority,@LIST,no_data_seg,
                             nuc_allocated_stk,stk_600_bytes,no_8087,
                             @status);
227   2       call RQ$CATALOG$OBJECT(this_job,list_dir_task,@(5,'LDTSK'),@status);

228   2       gname_task=RQ$CREATE$TASK(gn_priority,@GET_NAME,no_data_seg,
                             nuc_allocated_stk,stk_600_bytes,no_8087,
                             @status);
229   2       call RQ$CATALOG$OBJECT(this_job,gname_task,@(5,'GNTSK'),@status);

230   2       cre_file_task=RQ$CREATE$TASK(utils_priority,@CREATE_A_FILE,no_data_seg,
                             nuc_allocated_stk,stk_600_bytes,no_8087,
                             @status);
231   2       call RQ$CATALOG$OBJECT(this_job,cre_file_task,@(5,'CRTSK'),@status);

232   2       att_file_task=RQ$CREATE$TASK(utils_priority,@CONNECT_FILE,no_data_seg,
                             nuc_allocated_stk,stk_600_bytes,no_8087,
                             @status);
233   2       call RQ$CATALOG$OBJECT(this_job,att_file_task,@(5,'ATTSK'),@status);

234   2       END CREATE_UTILS;

              $EJECT
```

```
235   1     SET_UP_FLOPPY: PROCEDURE;

            /* This procedure creates the task which looks after the floppy disk which
               contains the welding data. The task is entered in the object directory
               purely for debugging.*/

236   2     declare floppy_task_token                    token;

237   2     floppy_task_token=RQ$CREATE$TASK(sec_disk_priority,@FLOPPY_SECURITY_TASK,no_data_segs,
                        nuc_allocated_stk,stk_600_bytes,no_8087,
                        @status);
238   2     call RQ$CATALOG$OBJECT(this_job,floppy_task_token,@(7,'FLOPTSK'),@status);

239   2     END SET_UP_FLOPPY;


240   1     SET_ERROR_H: PROCEDURE;

241   2     declare e_h structure (offset word, base word, mode byte);

242   2     e_h.offset,e_h.base,e_h.mode=null;
243   2     call RQ$SET$EXCEPTION$HANDLER(@e_h,@status);

244   2     END SET_ERROR_H;

            $EJECT
```

```
245   1       HARDWARE_RESET: PROCEDURE;

              /* Hardware reset as the name sujests puts all the specialised hardware into
                 a known state. The three I/O ports are programmed to be input or output as
                 required. The digital to analogue board is then reset and the motors are
                 turned off and the six timers for clocking the motors are put in the
                 correct mode.*/

246   2       OUTPUT(0CEH)=90H;
              /* PORT A (0C8H) 8 BIT I/P */
              /* PORT B (0CAH) 8 BIT O/P */
              /* PORT C (0CCH) 8 BIT O/P */
247   2       OUTPUT(0CAH)=0FFH;
248   2       OUTPUT(0CCH)=0FFH;
              /* BOTH THE ABOVE ARE INVERTED AFTER iSC920 DRIVERS */
249   2       OUTPUT(0A2H)=00H;
              /* ANALOG RESET */
250   2       OUTPUT(0408H)=00H;
              /* MOTORS OFF */
251   2       OUTPUT(0416H)=034H;
252   2       OUTPUT(0416H)=074H;
253   2       OUTPUT(0416H)=0B4H;
254   2       OUTPUT(041EH)=034H;
255   2       OUTPUT(041EH)=074H;
256   2       OUTPUT(041EH)=0B4H;
              /* ALL SIX TIMERS IN MODE 2 */
              /* ( RATE GENERATION MODE )   */

257   2       END HARDWARE_RESET;

              $EJECT
```

```
                /******************************** MAIN PROG ************************************

                   This is the main  program  consisting of an initial task which  creates a
                   user number ( which will be  modified to  passcode  checking  later) then
                   it creates connections to the terminal and the floppy disks. The internal
                   software communication  lines are then  created in the form of mailboxes,
                   the rest of the child  tasks are  then created and  allowed to initialise
                   and the root task is informed that the initailisation is finished.
                   The rest of the module is the executive command line interpreter accepting
                   only the following four commands ;-
                   1/ DATA
                   2/ PATH
                   3/ WELD
                   4/ LOGOFF

                ********************************************************************************/

258   1         START$TASK:PROCEDURE PUBLIC;

259   2         call RQ$END$INIT$TASK;
260   2         call INIT$REAL$MATH$UNIT;
261   2         CALL SET_ERROR_H;
                /* Reset all hardware to a known state */
262   2         call hardware_reset;
                /* Create the utility tasks */
263   2         call create_mbx_sems;
264   2         call create_th;
265   2         call create_utils;
                /* Ring bell three times */
266   2         call MOVB(bell3,@wrt_buf,6);
267   2         call RQ$SEND$MESSAGE(th_in_mbx,com_seg_vals.base,th_out_mbx,@status);
268   2         com_seg_vals.base=RQ$RECEIVE$MESSAGE(th_out_mbx,wait_forever,no_response,@status);
                /* Enter correct path & data disk */
269   2         call set_up_floppy;
270   2         call RQ$SEND$UNITS(disk_sem,one,@status);
                /* Create data, path, disk & weld tasks */
271   2         call create_tasks;
                /* Get command from the terminal and parse it looking for a valid command */
                /* then send a unit to the correct semaphore to start the required task.  */
272   2         do forever;
273   3            call MOVB(@title2,@wrt_buf,32);
274   3            call RQ$SEND$MESSAGE(th_in_mbx,com_seg_vals.base,th_out_mbx,@status);
275   3            com_seg_vals.base=RQ$RECEIVE$MESSAGE(th_out_mbx,wait_forever,no_response,@status);
276   3            call RQ$SEND$MESSAGE(read_in,com_seg_vals.base,read_out,@status);
277   3            com_seg_vals.base=RQ$RECEIVE$MESSAGE(read_out,wait_forever,no_response,@status);
278   3            if CMPB(@('LOGOFF'),@read_buf.chars,4)=match then call RQ$SEND$UNITS(disk_sem,one,@status);
280   3            if CMPB(@('DATA'),@read_buf.chars,4)=match then call RQ$SEND$UNITS(data_sem,one,@status);
282   3            if CMPB(@('PATH'),@read_buf.chars,4)=match then call RQ$SEND$UNITS(path_sem,one,@status);
284   3            if CMPB(@('WELD'),@read_buf.chars,4)=match then call RQ$SEND$UNITS(weld_sem,one,@status);
286   3         end;
```

```
 287   2      END;

 288   1      END SUPERVISOR;


MODULE INFORMATION:

     CODE AREA SIZE     = 07ADH   1965D
     CONSTANT AREA SIZE = 00EDH    237D
     VARIABLE AREA SIZE = 0041H     65D
     MAXIMUM STACK SIZE = 0020H     32D
     672 LINES READ
     0 PROGRAM WARNINGS
     0 PROGRAM ERRORS

DICTIONARY SUMMARY:

     116KB MEMORY AVAILABLE
     13KB MEMORY USED   (11%)
     0KB DISK SPACE USED

END OF PL/M-86 COMPILATION
```

2. DATA UTILITIES TASK.

```
iRMX 86 PL/M-86 V2.3 COMPILATION OF MODULE DATA_LIBRARY
OBJECT MODULE PLACED IN DATA/LIB.OBJ
COMPILER INVOKED BY:  :LANG:PLM86 DATA/LIB.P86


              $TITLE ('SUPERWELD  DATA LIBRARY TASK VER 86/1.0  DATE 10/03/83')

              $DEBUG
              $LARGE
              $OPTIMIZE(3)

  1           DATA_LIBRARY: DO;

              $INCLUDE (INC/LITERALS.P86)
              $SAVE NOLIST
              $INCLUDE(/INC/RMX86/NCRMBX.EXT)
              $SAVE NOLIST
              $INCLUDE(/INC/RMX86/NDLMBX.EXT)
              $SAVE NOLIST
              $INCLUDE(/INC/RMX86/NRSTSK.EXT)
              $SAVE NOLIST
              $INCLUDE(/INC/RMX86/NSUTSK.EXT)
              $SAVE NOLIST
              $INCLUDE(/INC/RMX86/NRCMES.EXT)
              $SAVE NOLIST
              $INCLUDE(/INC/RMX86/NSNMES.EXT)
              $SAVE NOLIST
              $INCLUDE(/INC/RMX86/NDLSEG.EXT)
              $SAVE NOLIST
              $INCLUDE(/INC/RMX86/NCRSEG.EXT)
              $SAVE NOLIST
              $INCLUDE(/INC/RMX86/NSTEXH.EXT)
              $SAVE NOLIST
              $INCLUDE(/INC/RMX86/NLUOBJ.EXT)
              $SAVE NOLIST
              $INCLUDE(/INC/RMX86/NRCUNI.EXT)
              $SAVE NOLIST
              $INCLUDE(/INC/RMX86/NGTTYP.EXT)
              $SAVE NOLIST
              $INCLUDE(/INC/RMX86/IAATFL.EXT)
              $SAVE NOLIST
              $INCLUDE(/INC/RMX86/IACRDR.EXT)
              $SAVE NOLIST
              $INCLUDE(/INC/RMX86/IAGTDE.EXT)
              $SAVE NOLIST

              $EJECT
```

```
$INCLUDE(/INC/RMX86/IADLCN.EXT)
$SAVE NOLIST
$INCLUDE(/INC/RMX86/IAOPEN.EXT)
$SAVE NOLIST
$INCLUDE(/INC/RMX86/IAREAD.EXT)
$SAVE NOLIST
$INCLUDE(/INC/RMX86/IAWRIT.EXT)
$SAVE NOLIST
$INCLUDE(/INC/RMX86/IASEEK.EXT)
$SAVE NOLIST
$INCLUDE(/INC/RMX86/IACLOS.EXT)
$SAVE NOLIST
$INCLUDE(/INC/RMX86/IADLFL.EXT)
$SAVE NOLIST
$INCLUDE(/INC/RMX86/IWTIO.EXT)
$SAVE NOLIST

$EJECT
```

```
/***************************** LOCALS ******************************************/
/*                          THE DATA SET STRUCTURE                             */

166  1    declare data_set      structure      (I_P          real,
                                               I_b          real,
                                               t_P          real,
                                               t_b          real,
                                               I_s1         real,
                                               I_s2         real,
                                               t_s1         real,
                                               t_s2         real,
                                               t_su         real,
                                               t_sd         real,
                                               t_g1         real,
                                               t_g2         real,
                                               V_t          real,
                                               P_m          real,
                                               spare0       real,
                                               spare1       real,
                                               spare2       real,
                                               spare3       real,
                                               spare4       real,
                                               spare5       byte,
                                               spare6       byte,
                                               set          byte,
                                               stop         byte);

/*          NAME                                    COUNTER       TOTAL      O/P
 *          ------                                  ----------    -------    -----
 *
 *   I_P   = pulse current                            -             -        I_Pw
 *   I_b   = background current                       -             -        I_bw
 *   t_P   = high current pulse width                 t_Pc          t_Pt       -
 *   t_b   = low current pulse width                  t_bc          t_bt       -
 *   I_s1  = stand by current ( for slope up )        -             -        I_s1w
 *   I_s2  = stand by current ( for slope down )      -             -        I_s2w
 *   t_s1  = time at stand by current I_s1            t_s1c         t_s1t
 *   t_s2  = time at stand by current I_s2            t_s2c         t_s2t      -
 *   t_su  = slope up time ( I_s1 to I_P )            t_suc         t_sut      -
 *   t_sd  = slope down time (I_P to I_s2 )           t_sdc         t_sdt      -
 *   t_g1  = gas pre-purge time                       t_g1c         t_g1t      -
 *   t_g2  = gas post-purge time                      t_g2c         t_g2t      -
 *   V_t   = velocity of traverse                     -             -          -
 *   P_m   = penetration monitor                      -             -          -
 *    set  = set number (0-255 possible therefore)
 *    stop = FFH  end of list marker
 */

*EJECT
```

```
                /*                  THIS TASKS GLOBAL VARIABLES                                  */

167   1         declare (done,loop,error_result,value)                                    byte;
168   1         declare temp                                                              real;
169   1         declare (status,gen_word,type_code,bytes_read)                            word;
170   1         declare data_ptr                                                        pointer;
171   1         declare (data_array based data_ptr)(14)                                   real;
172   1         declare (read_in,read_out,att_file_mbx,cre_file_mbx)                     token;
173   1         declare (def_connection,data_sem,main_task_token)                        token;
174   1         declare (dir_mailbox,data_mailbox,gen_mailbox,disk_token,list_dir_mbx)   token;
175   1         declare (file_connection,dir_connection,th_out_mbx,th_in_mbx,sname_mbx)  token;
176   1         declare wrt_buf_ptr                                                     pointer;
177   1         declare wrt_buf_vals structure (offset word,base word)    AT   (@wrt_buf_ptr);
178   1         declare data_real_save_area(94)                                           byte;
179   1         declare (wrt_buf based wrt_buf_ptr)(80)                                   byte;
180   1         declare (read_buf based wrt_buf_ptr)(80)                                  byte;
181   1         declare st_null                                          byte        data(0);

                /*                  QUESTIONS AND ERRORS                                         */

182   1         declare question1(*) byte data (20,0,50,'Data listing (Return=Exit Space+Return=next page).');
183   1         declare question2(*) byte data (20,0,49,'DATA EDITOR (A-N CHECK CLEAR END EXIT QUIT NEXT).');
184   1         declare question3(*) byte data (20,0,43,'DATA INPUT (A-N CHECK CLEAR END QUIT NEXT).');
185   1         declare question4(*) byte data (20,0,64,'DATA LIBRARY UTILITIES (INPUT EDIT DIR LIST LOAD DELETE Return).'
                    -);
186   1         declare question5(*) byte data (20,0,40,'Please enter name of directory required.');
187   1         declare question6(*) byte data (20,0,46,'Please enter new directory name (9 Chars Max).');
188   1         declare error1(*) byte data (22,0,38,'Command Aborted (file already exists).');

                $EJECT
```

```
                /************************** PROCEDURES ****************************************/

189   1         INPUT_REAL_NUMBER: PROCEDURE(buffer_ptr,count,num_ptr) WORD;

                /* This procedure is used to convert ASCII real numbers entered at the */
                /* to internal real number format                                      */

190   2         declare (negative,loop,done,count)                          byte;
191   2         declare (num_ptr,buffer_ptr)                              pointer;
192   2         declare (result based num_ptr)                               real;
193   2         declare (whole,fraction)                                     real;
194   2         declare (buffer based buffer_ptr)(15)                        byte;

195   2         done=false;
196   2         loop=1;
197   2         whole=0.0;
198   2         fraction=0.0;
199   2         count=count-1;
200   2         negative=false;
201   2         do while buffer(loop)=space;
202   3            loop=loop+1;
203   3         end;
204   2         do while buffer(count)=space;
205   3            count=count-1;
206   3         end;
207   2         if ((buffer(loop)='+')OR(buffer(loop)='-'))then
208   2           do;
209   3              if buffer(loop)='-'then negative=true;
211   3              loop=loop+1;
212   3           end;
213   2         do while ((buffer(loop)>='0')AND(buffer(loop)<='9')AND(done=false));
214   3            whole=whole*10.0;
215   3            whole=whole+(FLOAT(INT(buffer(loop)-'0')));
216   3            loop=loop+1;
217   3            if loop>count then done=true;
219   3         end;
220   2         if ((buffer(loop)='.')AND(done=false)) then
221   2           do;
222   3              do while ((buffer(count)>='0')AND(buffer(count)<='9'));
223   4                 fraction=fraction+(FLOAT(INT(buffer(count)-'0')));
224   4                 fraction=fraction/10.0;
225   4                 count=count-1;
226   4              end;
227   3              if loop=count then done=true;
229   3           end;
230   2         result=whole+fraction;
231   2         if negative then result=-result;
233   2         if ((whole>=1000.0)OR(done=false)OR((result<0.001)AND(result<>0.0))) then return real_error;
235   2         return E$OK;

244   2         END INPUT_REAL_NUMBER;
```

```
237   1       CREATE_A_DIR: PROCEDURE(dir_token,new_dir_token_ptr) WORD;

              /* This  procedure is to  create a  directory, it first  lists all the that
                 already exist on the  screen to help the user to  avoid name duplication
                 duplication. The procedure then tries to attach the directory in case it
                 already exists, if it does then this is okay, else a new  directory will
                 be created. */

238   2       declare (iors_ptr,new_dir_token_ptr)                              pointer;
239   2       declare iors_seg structure (offset word,base word)      AT     (@iors_ptr);
240   2       declare iors based iors_ptr structure (status word,ustat word,actual word);
241   2       declare dir_token                                                  token;
242   2       declare ret_status                                                 word;
243   2       declare new_dir_token based new_dir_token_ptr                      token;

244   2       ret_status=E$OK;
245   2       call RQ$SEND$MESSAGE(list_dir_mbx,dir_token,gen_mailbox,@status);
246   2       dir_token=RQ$RECEIVE$MESSAGE(gen_mailbox,wait_forever,no_response,@status);
247   2       dir_mailbox=RQ$CREATE$MAILBOX(fifo,@status);
248   2       call MOVB(@question6,@wrt_buf,49);
249   2       call write_message;
250   2       call RQ$SEND$MESSAGE(@name_mbx,wrt_buf_vals.base,gen_mailbox,@status);
251   2       wrt_buf_vals.base=RQ$RECEIVE$MESSAGE(gen_mailbox,wait_forever,no_response,@status);
252   2       call RQ$A$ATTACH$FILE(id,dir_token,@read_buf(0),dir_mailbox,@status);
253   2       new_dir_token=RQ$RECEIVE$MESSAGE(dir_mailbox,wait_forever,no_response,@status);
254   2       type_code=RQ$GET$TYPE(new_dir_token,@status);
255   2       if type_code=segment_code then
256   2         do;
257   3           iors_seg.base=new_dir_token;
258   3           iors_seg.offset=null;
259   3           type_code=iors.status;
260   3           call RQ$DELETE$SEGMENT(iors_seg.base,@status);
261   3           if type_code=E$FNEXIST then
262   3             do;
263   4               call RQ$A$CREATE$DIRECTORY(id,dir_token,@read_buf(0),DDAC,dir_mailbox,@status);
264   4               new_dir_token=RQ$RECEIVE$MESSAGE(dir_mailbox,wait_forever,no_response,@status);
265   4               type_code=RQ$GET$TYPE(new_dir_token,@status);
266   4               if type_code=segment_code then
267   4                 do;
268   5                   call RQ$DELETE$SEGMENT(new_dir_token,@status);
269   5                   ret_status=error_flag;
270   5                 end;
271   4             end;
272   3           else ret_status=error_flag;
273   3         end;
274   2       call RQ$DELETE$MAILBOX(dir_mailbox,@status);
275   2       return ret_status;

276   2       END CREATE_A_DIR;
```

```
277  1      CONNECT_DIR: PROCEDURE(dir_token,new_dir_token_ptr);

            /* This procedures job is to connect a directory that already exists, if
               it fails it tries again by asking for another name until it succeeds. */

278  2      declare new_dir_token_ptr                                    pointer;
279  2      declare dir_token                                              token;
280  2      declare new_dir_token based new_dir_token_ptr                  token;
281  2      declare done                                                    byte;

282  2      call RQ$SEND$MESSAGE(list_dir_mbx,dir_token,gen_mailbox,@status);
283  2      dir_token=RQ$RECEIVE$MESSAGE(gen_mailbox,wait_forever,no_response,@status);
284  2      dir_mailbox=RQ$CREATE$MAILBOX(fifo,@status);
285  2      done=bad;
286  2      do while not done;
287  3         call MOVB(@question5,@wrt_buf,43);
288  3         call write_message;
289  3         call RQ$SEND$MESSAGE(sname_mbx,wrt_buf_vals.base,gen_mailbox,@status);
290  3         wrt_buf_vals.base=RQ$RECEIVE$MESSAGE(gen_mailbox,wait_forever,no_response,@status);
291  3         call RQ$A$ATTACH$FILE(id,dir_token,@read_buf(0),dir_mailbox,@status);
292  3         if status=E$OK then
293  3           do;
294  4              new_dir_token=RQ$RECEIVE$MESSAGE(dir_mailbox,wait_forever,no_response,@status);
295  4              type_code=RQ$GET$TYPE(new_dir_token,@status);
296  4              if type_code=connection_code then done=good;
298  4              else call RQ$DELETE$SEGMENT(new_dir_token,@status);
299  4           end;
300  3      end;
301  2      call RQ$DELETE$MAILBOX(dir_mailbox,@status);

302  2      END CONNECT_DIR;

            $EJECT
```

```
303   1       INSERT_VALUE: PROCEDURE(number);

              /* This routine takes a real number as it's parameter and converts it
                 into ASCII digits and places them in the 1st 8 bytes of the wrt_buf.
                 The first digit is always a '+' or '-' and the fourth is always a
                 decimal point. Three digits are produced both before and after the
                 decimal point with all leading and lagging zeros supressed with
                 blanks. */

304   2       declare (number,temp)                                          real;
305   2       declare (blanks,buffer)                                        byte;
306   2       declare (whole,fraction)                                       word;

307   2       call SET$REAL$MODE(set_fix_towards_0);
308   2       wrt_buf(3)='+';
309   2       if number<>ABS(number)then
310   2         do;
311   3            number=ABS(number);
312   3            wrt_buf(3)='-';
313   3         end;
314   2       temp=FLOAT(FIX(number));
315   2       number=1000.0*(number-temp);
316   2       whole=UNSIGN(FIX(temp));
317   2       call SET$REAL$MODE(set_fix_nearest_int);
318   2       fraction=UNSIGN(FIX(number));
319   2       if fraction>=1000 then
320   2         do;
321   3            fraction=fraction-1000;
322   3            whole=whole+1;
323   3         end;
324   2       call SET$REAL$MODE(set_fix_towards_0);
325   2       do loop = 1 to 3;
326   3          buffer=(whole MOD 10)+'0';
327   3          whole=whole/10;
328   3          if ((whole=0)AND(buffer='0')AND(loop<>1)) then buffer=space;
330   3          wrt_buf(7-loop)=buffer;
331   3       end;
332   2       wrt_buf(7)='.';
333   2       blanks=true;
334   2       do loop=1 to 3;
335   3          buffer=(fraction MOD 10)+'0';
336   3          fraction=fraction/10;
337   3          if ((buffer<>'0') OR (loop=3)) then blanks=false;
339   3          if blanks=true then buffer=space;
341   3          wrt_buf(11-loop)=buffer;
342   3       end;
343   2       call SET$REAL$MODE(set_fix_nearest_int);
344   2       wrt_buf(2)=8;

345   1       END INSERT_VALUE;
```

```
346   1       DATA_SCREEN: PROCEDURE;

              /* This  routine  simple puts all the text  associated with the data screen
                 on to the terminal leaving a gap for the actual value to be inserted in. */

347   2       declare line(15) structure (chars(41) byte) data
                                (40,' A:          PULSE CURRENT=        Amps  ',
                                40,' B:    BACKGROUND CURRENT=        Amps  ',
                                40,' C:          PULSE ON TIME=       Secs  ',
                                40,' D:         PULSE OFF TIME=       Secs  ',
                                40,' E:  PRE-STAND BY CURRENT=        Amps  ',
                                40,' F: POST-STAND BY CURRENT=        Amps  ',
                                40,' G:     PRE-STAND BY TIME=        Secs  ',
                                40,' H:    POST-STAND BY TIME=        Secs  ',
                                40,' I:          SLOPE UP TIME=       Secs  ',
                                40,' J:        SLOPE DOWN TIME=       Secs  ',
                                40,' K:    GAS PRE-PURGE TIME=        Secs  ',
                                40,' L:   GAS POST-PURGE TIME=        Secs  ',
                                40,' M:         TRAVERSE SPEED=       mm/Sec',
                                40,' N:   PENETRATION MONITOR=        on/off',
                                40,'                SET NUMBER=             ');

              /*   N.B.     26 Characters before value slot                    */

348   2       call MOVB(clear_screen,@wrt_buf,6);
349   2       call write_message;
350   2       do loop = 0 to 14;
351   3           call MOVB(@line(loop).chars,@wrt_buf(2),41);
352   3           wrt_buf(0)=loop+4;
353   3           call write_message;
354   3       end;

355   2       END DATA_SCREEN;

              $EJECT
```

```
356   1       DATA_DISPLAY: PROCEDURE;

              /* This procedure assumes the data screen has already been written and takes
                 the complete data set which is present in the data structure and displays
                 it using the insert_value routine. */

357   2       declare  loop       byte;
358   2       declare  set_real   real;

359   2       wrt_buf(1)=27;
360   2       do loop=0 to 13;
361   3          call insert_value(data_array(loop));
362   3          wrt_buf(0)=loop+4;
363   3          call write_message;
364   3       end;
365   2       set_real=FLOAT(INT(data_set.set));
366   2       call insert_value(set_real);
367   2       wrt_buf(0)=18;
368   2       call write_message;

369   2       END DATA_DISPLAY;

              $EJECT
```

```
370   1       DATA_LIST: PROCEDURE;

              /* This routine combines several of the previous routines to make up the list
                 function. Once LIST has been entered the routine asks for a  directory and
                 lists this directories contents on the screen. A file is the choosen using
                 the  attachfile task. The data screen is then displayed and  data from the
                 file is read and  displayed. To  display the next set  space and return is
                 entered, to exit just return is typed. */

371   2       declare  (set_number,sets,finished)        byte;

372   2       call connect_dir(def_connection,@dir_connection);
373   2       call RQ$SEND$MESSAGE(att_file_mbx,dir_connection,gen_mailbox,@status);
374   2       file_connection=RQ$RECEIVE$MESSAGE(gen_mailbox,wait_forever,no_response,@status);
375   2       call RQ$A$OPEN(file_connection,file_read,share_mode,no_response,@status);
376   2       call RQ$A$READ(file_connection,@sets,one,gen_mailbox,@status);
377   2       set_number=0;
378   2       bytes_read=RQ$WAIT$IO(file_connection,gen_mailbox,wait_forever,@status);
379   2       call data_screen;
380   2       finished=false;
381   2       read_buf(1)=space;
382   2       do while not finished;
383   3          if (read_buf(1)=space) then
384   3            do;
385   4               set_number=set_number+1;
386   4               if set_number>sets then
387   4                 do;
388   5                    set_number=1;
389   5                    call RQ$A$SEEK(file_connection,address_mode,one,gen_mailbox,@status);
390   5                    bytes_read=RQ$WAIT$IO(file_connection,gen_mailbox,wait_forever,@status);
391   5                 end;
392   4               call RQ$A$READ(file_connection,@data_set,one_set,gen_mailbox,@status);
393   4               bytes_read=RQ$WAIT$IO(file_connection,gen_mailbox,wait_forever,@status);
394   4               call data_display;
395   4            end;
396   3          call MOVB(@question1,@wrt_buf,53);
397   3          call write_message;
398   3          call RQ$SEND$MESSAGE(read_in,wrt_buf_vals.base,read_out,@status);
399   3          wrt_buf_vals.base=RQ$RECEIVE$MESSAGE(read_out,wait_forever,no_response,@status);
400   3          if read_buf(0)=null then finished=true;
402   3       end;
403   2       call RQ$A$CLOSE(file_connection,no_response,@status);
404   2       call MOVB(clear_screen,@wrt_buf,6);
405   2       call write_message;
106   2       call RQ$A$DELETE$CONNECTION(file_connection,no_response,@status);
407   2       call RQ$A$DELETE$CONNECTION(dir_connection,no_response,@status);

408   2       END DATA_LIST;

              $EJECT
```

```
409   1       DATA_DIR: PROCEDURE;

              /* This function merely lists the requested directories contents on the
                 screen using the listdirectory task. */

410   2       call connect_dir(def_connection,@dir_connection);
411   2       call RQ$SEND$MESSAGE(list_dir_mbx,dir_connection,gen_mailbox,@status);
412   2       dir_connection=RQ$RECEIVE$MESSAGE(gen_mailbox,wait_forever,no_response,@status);
413   2       call RQ$A$DELETE$CONNECTION(dir_connection,no_response,@status);

414   2       END DATA_DIR;


415   1       DATA_DELETE: PROCEDURE;

              /* This procedure first of all deletes the file required to be deleted
                 and  deletes its  connection. Then it  attempts to delete the files
                 parent directory. We do not care if it fails as this only indicates
                 that the directory is not empty. */

416   2       call connect_dir(def_connection,@dir_connection);
417   2       call RQ$SEND$MESSAGE(att_file_mbx,dir_connection,gen_mailbox,@status);
418   2       file_connection=RQ$RECEIVE$MESSAGE(gen_mailbox,wait_forever,no_response,@status);
419   2       call RQ$A$DELETE$FILE(id,file_connection,@st_null,no_response,@status);
420   2       call RQ$A$DELETE$CONNECTION(file_connection,no_response,@status);
421   2       call RQ$A$DELETE$FILE(id,dir_connection,@st_null,no_response,@status);
422   2       call RQ$A$DELETE$CONNECTION(dir_connection,no_response,@status);

423   2       END DATA_DELETE;

              $EJECT
```

```
424   1       DATA_LOAD: PROCEDURE;

              /* This routine connects a  directory and then a file in the usual way. The
                 file  connection is then sent to the data  mailbox where when instigated
                 the weld task may  receive it. Note  the  data  file  connection  is not
                 detached at the  end of the  routine. The mailbox is  flushed before the
                 token is sent to make sure that the last file token sent is the one that
                 will be welded with. */

425   2       declare (ex,connection)       word;

426   2       call connect_dir(def_connection,@dir_connection);
427   2       call RQ$SEND$MESSAGE(att_file_mbx,dir_connection,gen_mailbox,@status);
428   2       file_connection=RQ$RECEIVE$MESSAGE(gen_mailbox,wait_forever,no_response,@status);
429   2       connection=RQ$RECEIVE$MESSAGE(data_mailbox,no_waiting,no_response,@ex);
430   2       do while ex=E$OK;
431   3          call RQ$A$DELETE$CONNECTION(connection,no_response,@status);
432   3          connection=RQ$RECEIVE$MESSAGE(data_mailbox,no_waiting,no_response,@ex);
433   3       end;
434   2       call RQ$SEND$MESSAGE(data_mailbox,file_connection,no_response,@status);
435   2       call RQ$A$DELETE$CONNECTION(dir_connection,no_response,@status);

436   2       END DATA_LOAD;



437   1       SET_DEFAULTS: PROCEDURE;

              /* This routine sets the data structure to its default values so a new set */
              /* may be entered. It is used in the input routine at the start of a new   */
              /* set and when the clear function is requested.                           */

438   2       call SETB(null,@data_set,one_set);
439   2       data_set.stop=0FFH;
440   2       data_set.I_s1=20.0;
441   2       data_set.I_s2=20.0;

442   2       END SET_DEFAULTS;

              $EJECT
```

```
443   1       DATA_ERROR: PROCEDURE(error_number);

              /* This procedure is called after data verify has checked the data. If any
                 errors have occured they are listed on the screen for one second each. */

444   2       declare error_number                                              byte;
445   2       declare fail(15) structure (chars(36) byte) data
                                  (35,'NO PRE STAND BY CURRENT!           ',
                                  35,'NO POST STAND BY CURRENT!          ',
                                  35,'PULSE CURRENT OUT OF RANGE!        ',
                                  35,'BACKGROUND CURRENT OUT OF RANGE!   ',
                                  35,'PULSE ON TIME OUT OF RANGE!        ',
                                  35,'PULSE OFF TIME OUT OF RANGE!       ',
                                  35,'PRE STAND BY CURRENT OUT OF RANGE! ',
                                  35,'POST STAND BY CURRENT OUT OF RANGE!',
                                  35,'PRE STAND BY TIME OUT OF RANGE!    ',
                                  35,'POST STAND BY TIME OUT OF RANGE!   ',
                                  35,'SLOPE UP TIME OUT OF RANGE!        ',
                                  35,'SLOPE DOWN TIME OUT OF RANGE!      ',
                                  35,'GAS TIMES OUT OF RANGE!            ',
                                  35,'TRAVERSE SPEED OUT OF RANGE!       ',
                                  35,'MULTIPLE ERRORS IN THE DATA SET!   ');

446   2       call MOVB(@(22,0),@wrt_buf,2);
447   2       call MOVB(@fail(error_number).chars,@wrt_buf(2),36);
448   2       call write_message;
449   2       if error_number<>14 then call TIME(one_sec);
451   2       error_result=error_result+1;

452   2       END DATA_ERROR;

              $EJECT
```

```
453   1       DATA_VERIFY: PROCEDURE BYTE;

              /* This  procedure simply checks all the data to see if it is valid. If not
                 valid then the data error procedure is called and the error is displayed.*/

454   2       error_result=null;
455   2       if (((data_set.t_s1<>0.0) OR (data_set.t_su<>0.0)) AND (data_set.I_s1<=0.0)) then call data_error(0);
457   2       if (((data_set.t_s2<>0.0) OR (data_set.t_sd<>0.0)) AND (data_set.I_s2<=0.0)) then call data_error(1);
459   2       if (data_set.I_p<1.0) OR (data_set.I_p>500.0) then call data_error(2);
461   2       if (data_set.I_b<0.0) OR (data_set.I_b>=data_set.I_p) then call data_error(3);
463   2       if (data_set.t_p<0.0) OR (data_set.t_p>10.0) then call data_error(4);
465   2       if (data_set.t_b<0.0) OR (data_set.t_b>10.0) then call data_error(5);
467   2       if (data_set.I_s1<0.0) OR (data_set.I_s1>=data_set.I_p) then call data_error(6);
469   2       if (data_set.I_s2<0.0) OR (data_set.I_s2>=data_set.I_p) then call data_error(7);
471   2       if (data_set.t_s1<0.0) OR (data_set.t_s1>120.0) then call data_error(8);
473   2       if (data_set.t_s2<0.0) OR (data_set.t_s2>120.0) then call data_error(9);
475   2       if (data_set.t_su<0.0) OR (data_set.t_su>10.0) then call data_error(10);
477   2       if (data_set.t_sd<0.0) OR (data_set.t_sd>10.0) then call data_error(11);
479   2       if (data_set.t_s1<0.0) OR (data_set.t_s1>60.0) then call data_error(12);
481   2       if (data_set.t_s2<0.0) OR (data_set.t_s2>60.0) then call data_error(12);
483   2       if (data_set.V_t<0.1) OR (data_set.V_t>20.0) then call data_error(13);
485   2       if error_result>1 then call data_error(14);
487   2       if error_result<>null then
488   2         do;
489   3            error_result=bad;
490   3            return error_result;
491   3         end;
492   2       else
            2         do;
493   3            error_result=good;
494   3            if (data_set.t_b=0.0 OR data_set.t_p=0.0) then
495   3              do;
496   4                 data_set.t_b=0.0;
497   4                 data_set.t_p=0.0;
498   4                 data_set.I_b=0.0;
499   4              end;
500   3            if data_set.t_s1=0.0 then data_set.t_s2=0.0;
502   3         end;
503   2       return error_result;

504   2       END DATA_VERIFY;

              $EJECT
```

```
505   1     DATA_EDIT: PROCEDURE;

            /* The data editor first attaches two files, one that already exists and a
               new file. The data form the first file is listed on the screen one set at
               a time where it may be edited using the same commands as input data. The
               command which is an exception is the end command which truncates the file
               at the data set presently on the screen. Once finished the old file is
               left unchanged and the new file is stored on the disk. */

506   2     declare   old_file_connection                                    token;
507   2     declare (sets,new_sets,data_number,done)                       byte;

508   2     call connect_dir(def_connection,@dir_connection);
509   2     call RQ$SEND$MESSAGE(att_file_mbx,dir_connection,gen_mailbox,@status);
510   2     old_file_connection=RQ$RECEIVE$MESSAGE(gen_mailbox,wait_forever,no_response,@status);
511   2     call RQ$SEND$MESSAGE(cre_file_mbx,dir_connection,gen_mailbox,@status);
512   2     file_connection=RQ$RECEIVE$MESSAGE(gen_mailbox,wait_forever,no_response,@status);
513   2     type_code=RQ$GET$TYPE(file_connection,@status);
514   2     if type_code<>connection_code then
515   2       do;
516   3          call RQ$A$DELETE$CONNECTION(old_file_connection,no_response,@status);
517   3          call RQ$A$DELETE$CONNECTION(dir_connection,no_response,@status);
518   3          call RQ$DELETE$SEGMENT(file_connection,@status);
519   3          call MOVB(@error1,@wrt_buf,41);
520   3          call write_message;
521   3          return;
522   3       end;
523   2     call RQ$A$OPEN(old_file_connection,file_read,share_mode,no_response,@status);
524   2     call RQ$A$OPEN(file_connection,file_write,share_mode,no_response,@status);
525   2     call RQ$A$READ(old_file_connection,@sets,one,no_response,@status);
526   2     call RQ$A$SEEK(file_connection,address_mode,one,no_response,@status);
527   2     call data_screen;
528   2     new_sets=0;
529   2     call RQ$A$READ(old_file_connection,@data_set,one_set,gen_mailbox,@status);
530   2     bytes_read=RQ$WAIT$IO(old_file_connection,gen_mailbox,wait_forever,@status);
531   2     call data_display;
532   2     done=bad;
533   2     do while not done;
534   3        call MOVB(@question2,@wrt_buf,52);
535   3        call write_message;
536   3        call RQ$SEND$MESSAGE(read_in,wrt_buf_vals.base,read_out,@status);
537   3        wrt_buf_vals.base=RQ$RECEIVE$MESSAGE(read_out,wait_forever,no_response,@status);
538   3        if ((read_buf(1)>='A')AND(read_buf(1)<='N')AND(read_buf(2)=space)) then
539   3          do;
540   4             status=input_real_number(@read_buf(1),read_buf(0),@temp);
            $EJECT
```

```
 541   4              if status<>real_error then
 542   4                do;
 543   5                  data_number=read_buf(1)-'A';
 544   5                  data_array(data_number)=temp;
 545   5                  call insert_value(data_array(data_number));
 546   5                  wrt_buf(0)=data_number+4;
 547   5                  wrt_buf(1)=27;
 548   5                  call write_message;
 549   5                end;
 550   4            end;
 551   3          if CMPB(@('NEXT'),@read_buf(1),4)=match then
 552   3            if data_verify then
 553   3            do;
 554   4               call RQ$A$WRITE(file_connection,@data_set,one_set,gen_mailbox,@status);
 555   4               bytes_read=RQ$WAIT$IO(file_connection,gen_mailbox,wait_forever,@status);
 556   4               new_sets=new_sets+1;
 557   4               if data_set.set<>sets then
 558   4                 do;
 559   5                    call RQ$A$READ(old_file_connection,@data_set,one_set,gen_mailbox,@status);
 560   5                    bytes_read=RQ$WAIT$IO(old_file_connection,gen_mailbox,wait_forever,@status);
 561   5                    call data_display;
 562   5                 end;
 563   4               else done=good;
 564   4            end;
 565   3          if CMPB(@read_buf(1),@('CHECK'),5)=match then gen_word=data_verify;
 567   3          if CMPB(@read_buf(1),@('CLEAR'),5)=match then
 568   3            do;
 569   4               call set_defaults;
 570   4               data_set.set=new_sets+1;
 571   4               call data_display;
 572   4            end;
 573   3          if CMPB(@read_buf(1),@('END'),3)=match then
 574   3            if data_verify then
 575   3            do;
 576   4               call RQ$A$WRITE(file_connection,@data_set,one_set,gen_mailbox,@status);
 577   4               bytes_read=RQ$WAIT$IO(file_connection,gen_mailbox,wait_forever,@status);
 578   4               new_sets=new_sets+1;
 579   4               do while data_set.set<>sets;
 580   5                  call RQ$A$READ(old_file_connection,@data_set,one_set,gen_mailbox,@status);
 581   5                  bytes_read=RQ$WAIT$IO(old_file_connection,gen_mailbox,wait_forever,@status);
 582   5                  call RQ$A$WRITE(file_connection,@data_set,one_set,gen_mailbox,@status);
 583   5                  bytes_read=RQ$WAIT$IO(file_connection,gen_mailbox,wait_forever,@status);
 584   5                  new_sets=new_sets+1;
 585   5               end;
 586   4               call RQ$A$SEEK(file_connection,address_mode,null,no_response,@status);
 587   4               call RQ$A$WRITE(file_connection,@new_sets,one,no_response,@status);
 588   4               done=good;
 589   1            end;
          $EJECT
```

```
590   3          if CMPB(@read_buf(1),@('EXIT'),4)=match then
591   3            if data_verify then
592   3            do;
593   4               call RQ$A$WRITE(file_connection,@data_set,one_set,no_response,@status);
594   4               new_sets=new_sets+1;
595   4               call RQ$A$SEEK(file_connection,address_mode,null,no_response,@status);
596   4               call RQ$A$WRITE(file_connection,@new_sets,one,no_response,@status);
597   4               done=good;
598   4            end;
599   3          if CMPB(@read_buf(1),@('QUIT'),4)=match then done=good;
601   3       end;
602   2       call RQ$A$CLOSE(file_connection,no_response,@status);
603   2       call RQ$A$CLOSE(old_file_connection,no_response,@status);
604   2       call RQ$A$DELETE$CONNECTION(old_file_connection,no_response,@status);
605   2       call RQ$A$DELETE$CONNECTION(dir_connection,no_response,@status);
606   2       if read_buf(1)='Q' then call RQ$A$DELETE$FILE(id,file_connection,@st_null,no_response,@status);
608   2       call RQ$A$DELETE$CONNECTION(file_connection,no_response,@status);
609   2       call MOVB(clear_screen,@wrt_buf,6);
610   2       call write_message;

611   2       END DATA_EDIT;

               $EJECT
```

```
612   1     DATA_INPUT: PROCEDURE;

            /* The data input command is the most complex of all the data library commands.
               Firstly a new  file is  created ( using the  create  file  task)  with a new
               directory if  necessary. The  data  screen is  then  displayed  along with a
               default set of data. The data is  changed by typing the corresponding letter
               and the real number value. The data which is being changed can be calculated
               from the letter and the input_real_number routine decodes the value. The new
               value is then  displayed in the  correct  position on the  screen. The other
               commands  are detected using the PL/M compare  string routines and cause the
               following  action.  NEXT  causes the  presently  displayed set of data to be
               checked for  errors and if  error  free it is  written to the file and a new
               default set is displayed for alteration. CHECK causes the data to be checked
               for errors but the data remains on the screen and is not stored in the file.
               CLEAR enables  the operator to remove the present data screen and replace it
               with the  default set. This is  usually used when a large number of erroneus
               values have  been  entered. The  command END  causes the last data set to be
               checked and  stored. The file is then  closed and command is returned to the
               data  librarian level  where as QUIT will  delete the file and return to the
               librarian. */

613   2     declare (data_number,set_number,done)                          byte;

614   2     status=create_a_dir(def_connection,@dir_connection);
615   2     if status<>E$OK then return;
617   2     call RQ$SEND$MESSAGE(cre_file_mbx,dir_connection,gen_mailbox,@status);
618   2     file_connection=RQ$RECEIVE$MESSAGE(gen_mailbox,wait_forever,no_response,@status);
619   2     type_code=RQ$GET$TYPE(file_connection,@status);
620   2     if type_code<>connection_code then
621   2       do;
622   3          call RQ$A$DELETE$CONNECTION(dir_connection,no_response,@status);
623   3          call RQ$DELETE$SEGMENT(file_connection,@status);
624   3          call MOVB(@error1,@wrt_buf,41);
625   3          call write_message;
624   3          return;
627   3       end;
628   2     call RQ$A$OPEN(file_connection,file_write,share_mode,no_response,@status);
629   2     call RQ$A$SEEK(file_connection,address_mode,one,no_response,@status);
630   2     set_number=1;
631   2     call data_screen;
632   2     call set_defaults;
633   2     data_set.set=set_number;
634   2     call data_display;
635   2     done=bad;
636   2     call MOVB(@question3,@wrt_buf,46);
637   2     call write_message;
            $EJECT
```

```
638   2       do while not done;
639   3          call RQ$SEND$MESSAGE(read_in,wrt_buf_vals.base,read_out,@status);
640   3          wrt_buf_vals.base=RQ$RECEIVE$MESSAGE(read_out,wait_forever,no_response,@status);
641   3          if ((CMPB(@('NEXT'),@read_buf(1),4)=match)AND(set_number=255)) then call MOVB(@('END'),@read_buf(1),3);
643   3          if ((read_buf(1)>='A')AND(read_buf(1)<='N')AND(read_buf(2)=space)) then
644   3            do;
645   4               status=input_real_number(@read_buf(1),read_buf(0),@temp);
646   4               if status<>real_error then
647   4                 do;
648   5                   data_number=read_buf(1)-'A';
649   5                   data_array(data_number)=temp;
650   5                   call insert_value(data_array(data_number));
651   5                   wrt_buf(0)=data_number+4;
652   5                   wrt_buf(1)=27;
653   5                   call write_message;
654   5                 end;
655   4             end;
656   3          if CMPB(@('NEXT'),@read_buf(1),4)=match then
657   3            if data_verify then
658   3              do;
659   4                 call RQ$A$WRITE(file_connection,@data_set,one_set,gen_mailbox,@status);
660   4                 bytes_read=RQ$WAIT$IO(file_connection,gen_mailbox,wait_forever,@status);
661   4                 set_number=set_number+one;
662   4                 call set_defaults;
663   4                 data_set.set=set_number;
664   4                 call data_display;
665   4               end;
666   3          if CMPB(@read_buf(1),@('CHECK'),5)=match then gen_word=data_verify;
668   3          if CMPB(@read_buf(1),@('CLEAR'),5)=match then
669   3            do;
670   4               call set_defaults;
671   4               data_set.set=set_number;
672   4               call data_display;
673   4               call MOVB(clear_error_line,@wrt_buf,6);
674   4               call write_message;
675   4             end;
          $EJECT
```

```
676   3          if CMPB(@read_buf(1),@('END'),3)=match then
677   3            if data_verify then
678   3              do;
679   4                 call RQ$A$WRITE(file_connection,@data_set,one_set,no_response,@status);
680   4                 call RQ$A$SEEK(file_connection,address_mode,null,no_response,@status);
681   4                 call RQ$A$WRITE(file_connection,@set_number,one,no_response,@status);
682   4                 call RQ$A$CLOSE(file_connection,no_response,@status);
683   4                 call RQ$A$DELETE$CONNECTION(file_connection,no_response,@status);
684   4                 call RQ$A$DELETE$CONNECTION(dir_connection,no_response,@status);
685   4                 done=good;
686   4              end;
687   3          if CMPB(@read_buf(1),@('QUIT'),4)=match then
688   3            do;
689   4               call RQ$A$CLOSE(file_connection,no_response,@status);
690   4               call RQ$A$DELETE$FILE(id,file_connection,@st_null,no_response,@status);
691   4               call RQ$A$DELETE$FILE(id,dir_connection,@st_null,no_response,@status);
692   4               call RQ$A$DELETE$CONNECTION(file_connection,no_response,@status);
693   4               call RQ$A$DELETE$CONNECTION(dir_connection,no_response,@status);
694   4               done=good;
695   4            end;
696   3        end;
697   2        call MOVB(clear_screen,@wrt_buf,6);
698   2        call write_message;

699   2        END DATA_INPUT;

                $EJECT
```

```
700   1       WRITE_MESSAGE: PROCEDURE;

701   2       call RQ$SEND$MESSAGE(th_in_mbx,wrt_buf_vals.base,th_out_mbx,@status);
702   2       wrt_buf_vals.base=RQ$RECEIVE$MESSAGE(th_out_mbx,wait_forever,no_response,@status);

703   2       END WRITE_MESSAGE;


704   1       SET_ERROR_HANDLER: PROCEDURE;

705   2       declare e_h structure (offset word,base word,mode byte);

706   2       e_h.offset,e_h.base=null;
707   2       e_h.mode=1;
708   2       call RQ$SET$EXCEPTION$HANDLER(@e_h,@status);

709   2       END SET_ERROR_HANDLER;

                $EJECT
```

```
                /***************************** MAIN LOOP ***************************************/

 710    1       DATA_TASK: PROCEDURE PUBLIC;

                /* The main routine of the data  librarian task when  created looks up all the
                   tokens it  requires and then sleeps at its  semaphore  waiting for the main
                   task to send it a unit. Once this unit is  received the data task  suspends
                   the main task to stop it  continuing. The data  librarian then looks up the
                   disk connection as it may have changed since it last used it and then tries
                   to attach it's root storage  directory. If it fails then the disk has been
                   changed  without using the logoff  command and so resumes the main task and
                   again sleeps at its own semaphore. If the data directory is  found then the
                   command line is  displayed and the  librarian waits for  keyboard input and
                   calls the correct  function as the input dictates. Return to the Supervisor
                   is achieved by  using the LOAD command (in which case the librarian assumes
                   that the data is finished with) or by typing a single  carriage return. The
                   default data  directory is then  detached and the  communication  buffer is
                   deleted, the  Supervisor task is then resumed and the data librarian sleeps
                   waiting for its next unit. */

 711    2       th_in_mbx=RQ$LOOKUP$OBJECT(this_job,@(7,'THINMBX'),wait_forever,@status);
 712    2       th_out_mbx=RQ$LOOKUP$OBJECT(this_job,@(8,'THOUTMBX'),wait_forever,@status);
 713    2       read_in=RQ$LOOKUP$OBJECT(this_job,@(6,'READIN'),wait_forever,@status);
 714    2       read_out=RQ$LOOKUP$OBJECT(this_job,@(7,'READOUT'),wait_forever,@status);
 715    2       gname_mbx=RQ$LOOKUP$OBJECT(this_job,@(5,'GNMBX'),wait_forever,@status);
 716    2       att_file_mbx=RQ$LOOKUP$OBJECT(this_job,@(6,'ATTMBX'),wait_forever,@status);
 717    2       cre_file_mbx=RQ$LOOKUP$OBJECT(this_job,@(6,'CREMBX'),wait_forever,@status);
 718    2       list_dir_mbx=RQ$LOOKUP$OBJECT(this_job,@(5,'LDMBX'),wait_forever,@status);
 719    2       gen_mailbox=RQ$LOOKUP$OBJECT(this_job,@(4,'GMBX'),wait_forever,@status);
 720    2       data_mailbox=RQ$LOOKUP$OBJECT(this_job,@(4,'DMBX'),wait_forever,@status);
 721    2       main_task_token=RQ$LOOKUP$OBJECT(this_job,@(4,'MTSK'),wait_forever,@status);
 722    2       data_sem=RQ$LOOKUP$OBJECT(this_job,@(4,'DSEM'),wait_forever,@status);
 723    2       call set_error_handler;
 724    2       data_ptr=@data_set;
                $EJECT
```

```
723   2       do forever;
726   3         value=RQ$RECEIVE$UNITS(data_sem,one,wait_forever,@status);
727   3         call RQ$SUSPEND$TASK(main_task_token,@status);
728   3         call SAVE$REAL$STATUS(@data_real_save_area);
729   3         disk_token=RQ$LOOKUP$OBJECT(this_job,@(4,'DCON'),wait_forever,@status);
730   3         wrt_buf_vals.base=RQ$CREATE$SEGMENT(80,@status);
731   3         wrt_buf_vals.offset=null;
732   3         call RQ$A$ATTACH$FILE(id,disk_token,@(7,'DATADEF'),gen_mailbox,@status);
733   3         def_connection=RQ$RECEIVE$MESSAGE(gen_mailbox,wait_forever,no_response,@status);
734   3         type_code=RQ$GET$TYPE(def_connection,@status);
735   3         if type_code<>connection_code then CALL RQ$DELETE$SEGMENT(def_connection,@status);
737   3         else
                    do;
738   4              done=bad;
739   4              do while not done;
740   5                 call MOVB(@question4,@wrt_buf,67);
741   5                 call write_message;
742   5                 call RQ$SEND$MESSAGE(read_in,wrt_buf_vals.base,read_out,@status);
743   5                 wrt_buf_vals.base=RQ$RECEIVE$MESSAGE(read_out,wait_forever,no_response,@status);
744   5                 if read_buf(0)=null then done=good;
746   5                 else
                            do;
747   6                     if CMPB(@('INPUT'),@read_buf(1),5)=match then call data_input;
749   6                     if CMPB(@('DELETE'),@read_buf(1),6)=match then call data_delete;
751   6                     if CMPB(@('LIST'),@read_buf(1),4)=match then call data_list;
753   6                     if CMPB(@('DIR'),@read_buf(1),3)=match then call data_dir;
755   6                     if CMPB(@('EDIT'),@read_buf(1),4)=match then call data_edit;
757   6                     if CMPB(@('LOAD'),@read_buf(1),4)=match then
                                do;
759   7                        done=good;
760   7                        call data_load;
761   7                       end;
762   6                 end;
763   5              end;
764   4           end;
765   3         call MOVB(clear_all_lines,@wrt_buf,6);
766   3         call write_message;
767   3         call RQ$A$DELETE$CONNECTION(def_connection,no_response,@status);
768   3         call RQ$DELETE$SEGMENT(wrt_buf_vals.base,@status);
769   3         call RESTORE$REAL$STATUS(@data_real_save_area);
770   3         call RQ$RESUME$TASK(main_task_token,@status);
771   3       end;

772   2       END DATA_TASK;

773   1       END DATA_LIBRARY;

                $EJECT
```

MODULE INFORMATION:

```
     CODE AREA SIZE      = 1B46H   6982D
     CONSTANT AREA SIZE  = 06DBH   1755D
     VARIABLE AREA SIZE  = 011EH    286D
     MAXIMUM STACK SIZE  = 0026H     38D
     1363 LINES READ
     0 PROGRAM WARNINGS
     0 PROGRAM ERRORS
```

DICTIONARY SUMMARY:

```
     116KB MEMORY AVAILABLE
     20KB MEMORY USED   (17%)
     0KB DISK SPACE USED
```

END OF PL/M-86 COMPILATION

3. PATH UTILITIES TASK.

```
iRMX 86 PL/M-86 V2.3 COMPILATION OF MODULE PATH_LIBRARY
OBJECT MODULE PLACED IN PATH/LIB.OBJ
COMPILER INVOKED BY:  :LANG:plm86 PATH/LIB.P86


                $TITLE ('SUPERWELD  PATH LIBRARY TASK VER 86/2.0  DATE 23/11/84')

                $DEBUG
                $LARGE
                $OPTIMIZE(3)

   1            PATH_LIBRARY: DO;

                $INCLUDE(INC/LITERALS.P86)
          =     $SAVE NOLIST
                $INCLUDE(/INC/RMX86/NCRMBX.EXT)
          =     $SAVE NOLIST
                $INCLUDE(/INC/RMX86/NDLMBX.EXT)
          =     $SAVE NOLIST
                $INCLUDE(/INC/RMX86/NSNMES.EXT)
          =     $SAVE NOLIST
                $INCLUDE(/INC/RMX86/NRCMES.EXT)
          =     $SAVE NOLIST
                $INCLUDE(/INC/RMX86/NRSTSK.EXT)
          =     $SAVE NOLIST
                $INCLUDE(/INC/RMX86/NSUTSK.EXT)
          =     $SAVE NOLIST
                $INCLUDE(/INC/RMX86/NSTPRI.EXT)
          =     $SAVE NOLIST
                $INCLUDE(/INC/RMX86/NSTEXH.EXT)
          =     $SAVE NOLIST
                $INCLUDE(/INC/RMX86/NCRSEG.EXT)
          =     $SAVE NOLIST
                $INCLUDE(/INC/RMX86/NDLSEG.EXT)
          =     $SAVE NOLIST
                $INCLUDE(/INC/RMX86/NGTTYP.EXT)
          =     $SAVE NOLIST
                $INCLUDE(/INC/RMX86/NLUOBJ.EXT)
          =     $SAVE NOLIST
                $INCLUDE(/INC/RMX86/NRCUNI.EXT)
          =     $SAVE NOLIST
                $INCLUDE(/INC/RMX86/NSLEEP.EXT)
          =     $SAVE NOLIST
                $INCLUDE(/INC/RMX86/IAATFL.EXT)
          =     $SAVE NOLIST
                $EJECT
```

```
      $INCLUDE(/INC/RMX86/IATRUN.EXT)
=     $SAVE NOLIST
      $INCLUDE(/INC/RMX86/IAREAD.EXT)
=     $SAVE NOLIST
      $INCLUDE(/INC/RMX86/IAWRIT.EXT)
=     $SAVE NOLIST
      $INCLUDE(/INC/RMX86/IASEEK.EXT)
=     $SAVE NOLIST
      $INCLUDE(/INC/RMX86/IAOPEN.EXT)
=     $SAVE NOLIST
      $INCLUDE(/INC/RMX86/IACLOS.EXT)
=     $SAVE NOLIST
      $INCLUDE(/INC/RMX86/IADLCN.EXT)
=     $SAVE NOLIST
      $INCLUDE(/INC/RMX86/IADLFL.EXT)
=     $SAVE NOLIST
      $INCLUDE(/INC/RMX86/IAGTPC.EXT)
=     $SAVE NOLIST
      $INCLUDE(/INC/RMX86/IWTIO.EXT)
=     $SAVE NOLIST

      /* PATH FILE FORMAT:
         -----------------
         The path file will be formatted as follows :-

      1/ 16 Bytes reserved for storage of number of seams etc.
         a. number of bytes in one dimension of filled matrix. WORD
         b. number of words containing offsets. WORD
         c. number of splining routines in path. BYTE
         d. number of welds on the path. BYTE

      2/ 400 Bytes reserved for the storage of five lines of text each of
         80 characters.This will store a textual desciption of the path.

      3/ The rest of the file will store the path in point format.
         a. word integer offsets X(number of points).
         b. word integer offsets Y(number of points).
         c. word integer offsets Z(number of points).
         d. word integer offsets R0(number of points).
         f. word integer offsets R1(number of points).
         g. word integer offsets R2(number of points).

      */

      $EJECT
```

```
                /******************************* LOCALS ****************************************/

172   1         declare (done,loop,number_seams)                                    byte;
173   1         declare st_null                                                  byte data(0);
174   1         declare wrt_buf_ptr                                              pointer;
175   1         declare wrt_buf_vals structure(offset word,base word)  AT   (@wrt_buf_ptr);
176   1         declare path_info structure  (mat_dim word,num_points word,num_spln byte,
                                                                              num_weld byte);
177   1         declare (read_buf based wrt_buf_ptr)(80)                            byte;
178   1         declare (wrt_buf based wrt_buf_ptr)(80)                             byte;
179   1         declare (status,gen_word,bytes_read,type_code)                      word;
180   1         declare path_real_save_area(94)                                     byte;
181   1         declare (cre_file_mbx,list_dir_mbx)                                 token;
182   1         declare (path_mailbox,gen_mailbox,th_in_mbx)                        token;
183   1         declare (def_connection,main_task_token,th_out_mbx,read_in,read_out) token;
184   1         declare (disk_token,path_sem,file_connection,att_file_mbx,co)       token;

185   1         declare error1(*) byte data (22,0,38,'Command Aborted (file already exists).');
186   1         declare error2(*) byte data (22,0,19,'File reading error.');
187   1         declare error3(*) byte data (22,0,19,'Scale number error.');
188   1         declare error4(*) byte data (22,0,21,'Scale overflow error.');
189   1         declare question1(*) byte data (20,0,31,'Please enter five lines of text');
190   1         declare question2(*) byte data (20,0,26,'Path listing (Return=Exit).');
191   1         declare question3(*) byte data (20,0,74,'PATH LIBRARY UTILITIES (INPUT SCALE DIR LIST LOAD DELETE EDIT MOV
                -E TRACE).');
192   1         declare question4(*) byte data (20,0,24,'Please enter scale value');
193   1         declare message1(*) byte data (20,0,27,'*** USE THE KEY PENDANT ***');
194   1         declare message2(*) byte data (20,0,17,'**** TRACING ****');

                $EJECT
```

```
                /********************************* EXTERNALS ***********************************/

195   1         declare (ptbytes,ptcount)         word external;
196   1         declare (pinput,splncnt,weldcnt) byte external;
197   1         declare xarray(200)               word external;
198   1         declare yarray(200)               word external;
199   1         declare zarray(200)               word external;
200   1         declare r0array(200)              word external;
201   1         declare r1array(200)              word external;
202   1         declare r2array(200)              word external;

203   1         KEY_DRIVER: PROCEDURE EXTERNAL;
204   2         END KEY_DRIVER;

205   1         TRACK: PROCEDURE EXTERNAL;
206   2         END TRACK;

                $EJECT
```

```
                /********************************** PROCEDURES **********************************/

207   1         SCALE_NUMBER: PROCEDURE REAL;

                /* This routine reads the input buffer from the read task and tries to form
                   a real  number from the  ASCII. The range is  checked for the scale path
                   routine and then the real number is returned. */

208   2         declare (negative,loop,done,count,done2)                    byte;
209   2         declare (tempfrac,divisor,whole,fraction,result)            real;

210   2         done2=false;
211   2         call clear_the_screen;
212   2         do while not done2;
213   3            done2=true;
214   3            call MOVB(@question4,@wrt_buf,27);
215   3            call write_message;
216   3            call RQ$SEND$MESSAGE(read_in,wrt_buf_vals.base,read_out,@status);
217   3            wrt_buf_vals.base=RQ$RECEIVE$MESSAGE(read_out,wait_forever,no_response,@status);
218   3            loop=1;
219   3            whole=0.0;
220   3            done=false;
221   3            fraction=0.0;
222   3            tempfrac=0.0;
223   3            count=wrt_buf(0);
224   3            do while wrt_buf(count)=space;
225   4              count=count-1;
226   4            end;
227   3            do while wrt_buf(loop)=space;
228   4               loop=loop+1;
229   4            end;
230   3            if ((wrt_buf(loop)='+')OR(wrt_buf(loop)='-'))then
231   3              do;
232   4                 if wrt_buf(loop)='-'then done2=false;
234   4                 loop=loop+1;
235   4              end;
236   3            do while ((wrt_buf(loop)>='0')AND(wrt_buf(loop)<='9')AND(done=false));
237   4               whole=whole*10.0;
238   4               whole=whole+(FLOAT(INT(wrt_buf(loop)-'0')));
239   4               loop=loop+1;
240   4               if loop>count then done=true;
242   4            end;
              $EJECT
```

```
243   3          if ((wrt_buf(loop)='.')AND(done=false)) then
244   3            do;
245   4              loop=loop+1;
246   4              divisor=10.0;
247   4              do while ((wrt_buf(loop)>='0')AND(wrt_buf(loop)<='9'));
248   5                 tempfrac=tempfrac+(FLOAT(INT(wrt_buf(loop)-'0')));
249   5                 tempfrac=tempfrac/divisor;
250   5                 divisor=divisor*10.0;
251   5                 fraction=fraction+tempfrac;
252   5                 loop=loop+1;
253   5                 if loop>count then done=true;
255   5              end;
256   4            end;
257   3          result=whole+fraction;
258   3          if ((whole>20.0)OR(done=false)OR(result<0.1)OR(result=0.0)) then
259   3             do;
260   4               done2=false;
261   4               call MOVB(@error3,@wrt_buf,22);
262   4               call write_message;
263   4             end;
264   3        end;
265   2        return result;

266   2        END SCALE_NUMBER;

                $EJECT
```

```
267   1     PATH_DELETE: PROCEDURE;

            /* This routine simly connects a path file using the attachfile task and then
               deletes the file and the  connection. Unlike the data DELETE command there
               is no directory level to attempt to delete. */

268   2     call RQ$SEND$MESSAGE(att_file_mbx,def_connection,gen_mailbox,@status);
269   2     file_connection=RQ$RECEIVE$MESSAGE(gen_mailbox,wait_forever,no_response,@status);
270   2     call RQ$A$DELETE$FILE(id,file_connection,@st_null,no_response,@status);
271   2     call RQ$A$DELETE$CONNECTION(file_connection,no_response,@status);

272   2     END PATH_DELETE;


273   1     PATH_LOAD: PROCEDURE;

            /* The command LOAD acts  exactely like the data  librarians command LOAD. A
               file is attached using the  attachfile task as usual and then the mailbox
               is purged to  check that the last file 'posted' is the one that is welded
               with. The files token is then sent. Note the connection is not deleted as
               the token must remain valid for the welding task to pick it up.*/

274   2     declare (ex,connection)                                    word;

275   2     call RQ$SEND$MESSAGE(att_file_mbx,def_connection,gen_mailbox,@status);
276   2     file_connection=RQ$RECEIVE$MESSAGE(gen_mailbox,wait_forever,no_response,@status);
277   2     connection=RQ$RECEIVE$MESSAGE(path_mailbox,no_waiting,no_response,@ex);
278   2     do while ex=E$OK;
279   3        call RQ$A$DELETE$CONNECTION(connection,no_response,@status);
280   3        connection=RQ$RECEIVE$MESSAGE(path_mailbox,no_waiting,no_response,@ex);
281   3     end;
282   2     call RQ$SEND$MESSAGE(path_mailbox,file_connection,no_response,@status);

283   2     END PATH_LOAD;

            $EJECT
```

```
284   1       PATH_SCALE: PROCEDURE;

              /* This routine connects an already existing file from the main path directory
                 and  then using the  createfile  task we  open a new  file. The old file is
                 completely  read and then closed and detached. A real  number is then input
                 by the user and the  scale_number  routine decodes it. All X,Y and Z points
                 are multiplied by this scaling number and overflow is detected. If overflow
                 occurs then the new file is  deleted else all the new path data is  written
                 to this file and it is closed and stored as usual. */

285   2       declare old_file_connection                                          token;
286   2       declare quit                                                          byte;
287   2       declare (scale,xtemp,ytemp,ztemp)                                     real;
288   2       declare (bytes_read,scloop)                                           word;

289   2       call RQ$SEND$MESSAGE(att_file_mbx,def_connection,gen_mailbox,@status);
290   2       old_file_connection=RQ$RECEIVE$MESSAGE(gen_mailbox,wait_forever,no_response,@status);
291   2       call RQ$SEND$MESSAGE(cre_file_mbx,def_connection,gen_mailbox,@status);
292   2       file_connection=RQ$RECEIVE$MESSAGE(gen_mailbox,wait_forever,no_response,@status);
293   2       type_code=RQ$GET$TYPE(file_connection,@status);
294   2       if type_code<>connection_code then
295   2         do;
296   3            call RQ$DELETE$SEGMENT(file_connection,@status);
297   3            call MOVB(@error1,@wrt_buf,41);
298   3            call RQ$SEND$MESSAGE(th_in_mbx,wrt_buf_vals.base,th_out_mbx,@status);
299   3            wrt_buf_vals.base=RQ$RECEIVE$MESSAGE(th_out_mbx,wait_forever,no_response,@status);
300   3            return;
301   3         end;
302   2       call RQ$A$OPEN(file_connection,file_write,share_mode,no_response,@status);
303   2       call RQ$A$OPEN(old_file_connection,file_read,share_mode,no_response,@status);
304   2       call RQ$A$READ(old_file_connection,@xarray,416,gen_mailbox,@status);
305   2       bytes_read=RQ$WAIT$IO(old_file_connection,gen_mailbox,wait_forever,@status);
306   2       call RQ$A$WRITE(file_connection,@xarray,416,no_response,@status);
307   2       call MOVB(@xarray,@path_info,6);
308   2       call RQ$A$READ(old_file_connection,@xarray,path_info.mat_dim,no_response,@status);
309   2       call RQ$A$READ(old_file_connection,@yarray,path_info.mat_dim,no_response,@status);
310   2       call RQ$A$READ(old_file_connection,@zarray,path_info.mat_dim,no_response,@status);
311   2       call RQ$A$READ(old_file_connection,@r0array,path_info.mat_dim,no_response,@status);
312   2       call RQ$A$READ(old_file_connection,@r1array,path_info.mat_dim,no_response,@status);
313   2       call RQ$A$READ(old_file_connection,@r2array,path_info.mat_dim,gen_mailbox,@status);
              $EJECT
```

```
314   2       bytes_read=RQ$WAIT$IO(old_file_connection,gen_mailbox,wait_forever,@status);
315   2       scale=scale_number;
316   2       quit=false;
317   2       do scloop=0 to path_info.num_points-1;
318   3          if zarray(scloop)<>control_point then
319   3            if (zarray(scloop)<>0 AND yarray(scloop)<>0 AND xarray(scloop)<>0) then
320   3              do;
321   4                 xtemp=scale*FLOAT(SIGNED(xarray(scloop)));
322   4                 ytemp=scale*FLOAT(SIGNED(yarray(scloop)));
323   4                 ztemp=scale*FLOAT(SIGNED(zarray(scloop)));
324   4                 if ((xtemp>32767.0) OR (ytemp>32767.0) OR (ztemp>32767.0) OR
                            (xtemp<-32768.0) OR (ytemp<-32768.0) OR (ztemp<-32768.0)) then
325   4                   do;
326   5                     quit=true;
327   5                     call MOVB(@error4,@wrt_buf,24);
328   5                     call write_message;
329   5                   end;
330   4                 xarray(scloop)=UNSIGN(FIX(xtemp));
331   4                 yarray(scloop)=UNSIGN(FIX(ytemp));
332   4                 zarray(scloop)=UNSIGN(FIX(ztemp));
333   4               end;
334   3       end;
335   2       call RQ$A$CLOSE(old_file_connection,no_response,@status);
336   2       call RQ$A$DELETE$CONNECTION(old_file_connection,no_response,@status);
337   2       call path_write(path_info.mat_dim);
338   2       call RQ$A$CLOSE(file_connection,no_response,@status);
339   2       if quit=true then call RQ$A$DELETE$FILE(id,file_connection,@st_null,no_response,@status);
341   2       call RQ$A$DELETE$CONNECTION(file_connection,no_response,@status);
342   2       call clear_the_screen;

343   2       END PATH_SCALE;

               $EJECT
```

```
344   1       READ_PATH: PROCEDURE;

              /* This routine reads all limbs of the offset matrix assuming that the path
                 info matrix has been filled in correctly. */

345   2       call RQ$A$READ(file_connection,@xarray,path_info.mat_dim,no_response,@status);
346   2       call RQ$A$READ(file_connection,@yarray,path_info.mat_dim,no_response,@status);
347   2       call RQ$A$READ(file_connection,@zarray,path_info.mat_dim,no_response,@status);
348   2       call RQ$A$READ(file_connection,@r0array,path_info.mat_dim,no_response,@status);
349   2       call RQ$A$READ(file_connection,@r1array,path_info.mat_dim,no_response,@status);
350   2       call RQ$A$READ(file_connection,@r2array,path_info.mat_dim,gen_mailbox,@status);
351   2       bytes_read=RQ$WAIT$IO(file_connection,gen_mailbox,wait_forever,@status);

352   2       END READ_PATH;



353   1       TEXT_INPUT: PROCEDURE;

              /* This routine reads in the ASCII strings from the read task and places
                 them both in the files text storage area and displays them on the screen.
                 Once finished the screen is cleared. */

354   2       declare loop                                                    byte;

355   2       call RQ$A$SEEK(file_connection,address_mode,16,no_response,@status);
356   2       call clear_the_screen;
357   2       call MOVB(@question1,@wrt_buf,34);
358   2       call write_message;
359   2       do loop=0 to 4;
360   3          call SETB(null,@read_buf,81);
361   3          call RQ$SEND$MESSAGE(read_in,wrt_buf_vals.base,read_out,@status);
362   3          wrt_buf_vals.base=RQ$RECEIVE$MESSAGE(read_out,wait_forever,no_response,@status);
363   3          call RQ$A$WRITE(file_connection,@read_buf(1),one_set,gen_mailbox,@status);
364   3          bytes_read=RQ$WAIT$IO(file_connection,gen_mailbox,wait_forever,@status);
365   3          call MOVRB(@read_buf,@wrt_buf(char_count),81);
366   3          wrt_buf(col)=0;
367   3          wrt_buf(row)=10+loop;
368   3          call write_message;
369   3       end;
370   2       call clear_the_screen;

371   2       END TEXT_INPUT;

              $EJECT
```

```
372   1       PATH_INPUT: PROCEDURE;

              /* This routine is mainly done in ASM86, but the file handling is still done
                 here. A file is created using the createfile task and then the above text
                 input routine is called to fill the files text storage area. The prompt
                 to use the key pendant is then displayed on the screen and control is passed
                 to the assembly language routine drivers which allows the user to input the
                 path and fill in the offset matrix. When the system returns to the PL/M the
                 path store routine is called and the file is then closed and stored. */

373   2       call RQ$SEND$MESSAGE(cre_file_mbx,def_connection,gen_mailbox,@status);
374   2       file_connection=RQ$RECEIVE$MESSAGE(gen_mailbox,wait_forever,no_response,@status);
375   2       type_code=RQ$GET$TYPE(file_connection,@status);
376   2       if type_code<>connection_code then
377   2         do;
378   3            call RQ$DELETE$SEGMENT(file_connection,@status);
379   3            call MOVB(@error1,@wrt_buf,41);
380   3            call write_message;
381   3            return;
382   3         end;
383   2       call RQ$A$OPEN(file_connection,file_write,share_mode,no_response,@status);
384   2       call text_input;
385   2       call clear_the_screen;
386   2       call MOVB(@message1,@wrt_buf,30);
387   2       call write_message;
388   2       call RQ$SLEEP(100,@status);
389   2       pinput=input_routines;
390   2       call RQ$SET$PRIORITY(this_task,while_move_priority,@status);
391   2       call key_driver;
392   2       call RQ$SET$PRIORITY(this_task,path_priority,@status);
393   2       call MOVB(bell,@wrt_buf,4);
394   2       call write_message;
395   2       call path_store;
396   2       call RQ$A$CLOSE(file_connection,no_response,@status);
397   2       call RQ$A$DELETE$CONNECTION(file_connection,no_response,@status);

398   2       END PATH_INPUT;

              $EJECT
```

```
399   1       PATH_WRITE: PROCEDURE(number_points);

              /* The path write routine stores the path in the required file and in the
                 correct format. The number of points to be stored in each leg of the
                 matrix is the only parameter input. */

400   2       declare number_points word;

401   2       call RQ$A$WRITE(file_connection,@xarray,number_points,no_response,@status);
402   2       call RQ$A$WRITE(file_connection,@yarray,number_points,no_response,@status);
403   2       call RQ$A$WRITE(file_connection,@zarray,number_points,no_response,@status);
404   2       call RQ$A$WRITE(file_connection,@r0array,number_points,no_response,@status);
405   2       call RQ$A$WRITE(file_connection,@r1array,number_points,no_response,@status);
406   2       call RQ$A$WRITE(file_connection,@r2array,number_points,gen_mailbox,@status);
407   2       bytes_read=RQ$WAIT$IO(file_connection,gen_mailbox,wait_forever,@status);

408   2       END PATH_WRITE;


409   1       PATH_STORE: PROCEDURE;

              /* Path  store is one  level up from  path  write in that it  fills in the
                 information bytes at the start of the file as well as the offset matrix
                 and also does not assume the file  pointer is in the  correct  place on
                 entry. */

410   2       call RQ$A$SEEK(file_connection,address_mode,0,no_response,@status);
411   2       call RQ$A$WRITE(file_connection,@ptbytes,2,no_response,@status);
412   2       call RQ$A$WRITE(file_connection,@ptcount,2,no_response,@status);
413   2       call RQ$A$WRITE(file_connection,@splncnt,1,no_response,@status);
414   2       call RQ$A$WRITE(file_connection,@weldcnt,1,gen_mailbox,@status);
415   2       bytes_read=RQ$WAIT$IO(file_connection,gen_mailbox,wait_forever,@status);
416   2       call RQ$A$SEEK(file_connection,address_mode,416,gen_mailbox,@status);
417   2       bytes_read=RQ$WAIT$IO(file_connection,gen_mailbox,wait_forever,@status);
418   2       call path_write(ptbytes);
419   2       call RQ$A$TRUNCATE(file_connection,no_response,@status);

420   2       END PATH_STORE;

              $EJECT
```

```
421   1       WAIT_FOR_CR: PROCEDURE;

422   2       call MOVB(@question2,@wrt_buf,29);
423   2       call write_message;
424   2       call RQ$SEND$MESSAGE(read_in,wrt_buf_vals.base,read_out,@status);
425   2       wrt_buf_vals.base=RQ$RECEIVE$MESSAGE(read_out,wait_forever,no_response,@status);

426   2       END WAIT_FOR_CR;


427   1       CLEAR_THE_SCREEN: PROCEDURE;

428   2       call MOVB(clear_screen,@wrt_buf,6);
429   2       call write_message;

430   2       END CLEAR_THE_SCREEN;

              $EJECT
```

```
431   1     PATH_LIST: PROCEDURE;

            /* The path LIST routine connects a file in the usual way and displays its
               name on the top of the screen followed by any text present in the text
               storage area in the file underneath. Once a single carriage return has
               been input the offsets in the matrix are converted into ASCII and are
               displayed in six colomns, one for each leg. Again a carriage return is
               waited for before the screen is cleared. */

432   2     declare fname_ptr                                          pointer;
433   2     declare file_seg structure (offset word,base word) AT  (@fname_ptr);
434   2     declare fname based fname_ptr structure (status word,name(14) byte);
435   2     declare rloop                                                byte;
436   2     declare (count based fname_ptr)(2)                           byte;

437   2     call RQ$SEND$MESSAGE(att_file_mbx,def_connection,gen_mailbox,@status);
438   2     file_connection=RQ$RECEIVE$MESSAGE(gen_mailbox,wait_forever,no_response,@status);
439   2     call RQ$A$OPEN(file_connection,file_read,share_mode,no_response,@status);
440   2     call RQ$A$READ(file_connection,@path_info,6,gen_mailbox,@status);
441   2     bytes_read=RQ$WAIT$IO(file_connection,gen_mailbox,wait_forever,@status);
442   2     call RQ$A$SEEK(file_connection,address_mode,16,no_response,@status);
443   2     call RQ$A$GET$PATH$COMPONENT(file_connection,gen_mailbox,@status);
444   2     file_seg.offset=null;
445   2     file_seg.base=RQ$RECEIVE$MESSAGE(gen_mailbox,wait_forever,no_response,@status);
446   2     call MOVB(@(6,0,17,ESC,'*','3'),@wrt_buf,6);
447   2     call MOVB(@fname.name,@wrt_buf(6),14);
448   2     call write_message;
449   2     wrt_buf(row)=7;
450   2     wrt_buf(5)='4';
451   2     call write_message;
452   2     call RQ$DELETE$SEGMENT(file_seg.base,@status);
453   2     wrt_buf(col)=null;
454   2     do loop=0 to 4;
455   3        call RQ$A$READ(file_connection,@read_buf(3),one_set,gen_mailbox,@status);
456   3        bytes_read=RQ$WAIT$IO(file_connection,gen_mailbox,wait_forever,@status);
457   3        wrt_buf(char_count)=LOW(bytes_read);
458   3        wrt_buf(row)=13+loop;
459   3        call write_message;
460   3     end;
461   2     call MOVB(clear_all_lines,@wrt_buf,6);
462   2     call write_message;
463   2     call wait_for_cr;
464   2     call clear_the_screen;
465   2     call RQ$A$SEEK(file_connection,address_mode,416,no_response,@status);
466   2     call read_path;
            $EJECT
```

```
467   2      if bytes_read<>path_info.mat_dim then
468   2        do;
469   3           call MOVB(@error2,@wrt_buf,22);
470   3           call write_message;
471   3        end;
472   2      else
                 do;
473   3           rloop=0;
474   3           do loop=0 to path_info.num_points-1;
475   4              wrt_buf(char_count)=6;
476   4              wrt_buf(row)=(rloop MOD 18)+4;
477   4              wrt_buf(col)=12;
478   4              if zarray(loop)<>control_point then
479   4                do;
480   5                   call offset_ascii(xarray(loop));
481   5                   wrt_buf(col)=22;
482   5                   call offset_ascii(yarray(loop));
483   5                   wrt_buf(col)=32;
484   5                   call offset_ascii(zarray(loop));
485   5                   wrt_buf(col)=42;
486   5                   call offset_ascii(r0array(loop));
487   5                   wrt_buf(col)=52;
488   5                   call offset_ascii(r1array(loop));
489   5                   wrt_buf(col)=62;
490   5                   call offset_ascii(r2array(loop));
491   5                   rloop=rloop+1;
492   5                end;
493   4                else
                          do;
494   5                   if xarray(loop)<>control_point then
495   5                     do;
496   6                        do case xarray(loop);
497   7                           call MOVB(@('WELD STOP '),@wrt_buf(3),10);
498   7                           call MOVB(@('WELD START'),@wrt_buf(3),10);
499   7                           call MOVB(@('SPLINE OFF'),@wrt_buf(3),10);
500   7                           call MOVB(@('SPLINE ON '),@wrt_buf(3),10);
501   7                           call MOVB(@('PATH END  '),@wrt_buf(3),10);
502   7                        end;
503   6                        wrt_buf(col)=0;
504   6                        wrt_buf(char_count)=10;
505   6                        call write_message;
506   6                     end;
507   5                end;
508   4           end;
509   3        end;
510   2      call wait_for_cr;
511   2      call clear_the_screen;
512   2      call RQ$A$CLOSE(file_connection,no_response,@status);
513   2      call RQ$A$DELETE$CONNECTION(file_connection,no_response,@status);

514   2      END PATH_LIST;
```

```
515   1       PATH_EDIT: PROCEDURE;

516   2       call RQ$SEND$MESSAGE(att_file_mbx,def_connection,gen_mailbox,@status);
517   2       file_connection=RQ$RECEIVE$MESSAGE(gen_mailbox,wait_forever,no_response,@status);
518   2       call RQ$A$OPEN(file_connection,update,share_mode,no_response,@status);
519   2       call RQ$A$READ(file_connection,@path_info,6,gen_mailbox,@status);
520   2       bytes_read=RQ$WAIT$IO(file_connection,gen_mailbox,wait_forever,@status);
521   2       call RQ$A$SEEK(file_connection,address_mode,416,no_response,@status);
522   2       ptbytes=path_info.mat_dim;
523   2       ptcount=path_info.num_points;
524   2       weldcnt=path_info.num_weld;
525   2       splncnt=path_info.num_spln;
526   2       call read_path;
527   2       if bytes_read<>path_info.mat_dim then
528   2         do;
529   3            call MOVB(@error2,@wrt_buf,22);
530   3            call write_message;
531   3         end;
532   2       else
            ;  do;
533   3            pinput=edit_keys;
534   3            call MOVB(clear_question_line,@wrt_buf,6);
535   3            call write_message;
536   3            call MOVB(@message1,@wrt_buf,30);
537   3            call write_message;
538   3            call RQ$SLEEP(100,@status);
539   3            call RQ$SET$PRIORITY(this_task,while_move_priority,@status);
540   3            call track;
541   3            call RQ$SET$PRIORITY(this_task,path_priority,@status);
542   3            call MOVB(bell,@wrt_buf,4);
543   3            call write_message;
544   3            call RQ$A$SEEK(file_connection,address_mode,0,gen_mailbox,@status);
545   3            bytes_read=RQ$WAIT$IO(file_connection,gen_mailbox,wait_forever,@status);
546   3            call path_store;
547   3         end;
548   2       call RQ$A$CLOSE(file_connection,no_response,@status);
549   2       call RQ$A$DELETE$CONNECTION(file_connection,no_response,@status);

550   2       END PATH_EDIT;

              $EJECT
```

```
551   1       PATH_TRACE: PROCEDURE;

              /* The Path TRACE routine attaches a file in the usual manner and reads all
                 the information block and all the matrix. The trace message is then
                 displayed and the flags are set so that the ASM86 driver routines know
                 to move the robot round the path with no interruption. Once the program
                 returns the file is closed and detached. */

552   2       call RQ$SEND$MESSAGE(att_file_mbx,def_connection,gen_mailbox,@status);
553   2       file_connection=RQ$RECEIVE$MESSAGE(gen_mailbox,wait_forever,no_response,@status);
554   2       call RQ$A$OPEN(file_connection,update,share_mode,no_response,@status);
555   2       call RQ$A$READ(file_connection,@path_info,6,gen_mailbox,@status);
556   2       bytes_read=RQ$WAIT$IO(file_connection,gen_mailbox,wait_forever,@status);
557   2       call RQ$A$SEEK(file_connection,address_mode,416,no_response,@status);
558   2       call read_path;
559   2       if bytes_read<>path_info.mat_dim then
560   2         do;
561   3            call MOVB(@error2,@wrt_buf,22);
562   3            call write_message;
563   3         end;
564   2       else
              /*   do;
565   3            pinput=trace_keys;
566   3            call MOVB(clear_question_line,@wrt_buf,6);
567   3            call write_message;
568   3            call MOVB(@message2,@wrt_buf,20);
569   3            call write_message;
570   3            call RQ$SLEEP(100,@status);
571   3            call RQ$SET$PRIORITY(this_task,while_move_priority,@status);
572   3            call track;
573   3            call RQ$SET$PRIORITY(this_task,path_priority,@status);
574   3            call MOVB(bell,@wrt_buf,4);
575   3            call write_message;
576   3         end;
577   2       call RQ$A$CLOSE(file_connection,no_response,@status);
578   2       call RQ$A$DELETE$CONNECTION(file_connection,no_response,@status);

579   2       END PATH_TRACE;

              $EJECT
```

```
580   1       PATH_DIR: PROCEDURE;

581   2       call RQ$SEND$MESSAGE(list_dir_mbx,def_connection,gen_mailbox,@status);
582   2       def_connection=RQ$RECEIVE$MESSAGE(gen_mailbox,wait_forever,no_response,@status);

583   2       END PATH_DIR;


584   1       POSITION: PROCEDURE;

                /* The position routine simply passes control to the ASM86 drivers which
                   allows the  user to move the robot to any  position and  then returns
                   control to the PL/M. */

585   2       call MOVB(clear_question_line,@wrt_buf,6);
586   2       call write_message;
587   2       call MOVB(@message1,@wrt_buf,30);
588   2       call write_message;
589   2       pinput=speed_exit_only;
590   2       call RQ$SLEEP(100,@status);
591   2       call RQ$SET$PRIORITY(this_task,while_move_priority,@status);
592   2       call key_driver;
593   2       call RQ$SET$PRIORITY(this_task,path_priority,@status);
594   2       call MOVB(bell,@wrt_buf,4);
595   2       call write_message;

596   2       END POSITION;

                $EJECT
```

```
597   1       OFFSET_ASCII: PROCEDURE(offset_num);

              /* The offset ASCII routine converts any offset passed as a parameter into
                 the ascii equivalent which is then written to the screen. */

598   2       declare oaloop                                          byte;
599   2       declare offset_num                                      word;
600   2       declare int_offset                                    integer;

601   2       int_offset=SIGNED(offset_num);
602   2       if int_offset<0 then wrt_buf(3)='-';
604   2       else wrt_buf(3)='+';
605   2       int_offset=IABS(int_offset);
606   2       do oaloop=0 to 4;
607   3          wrt_buf(8-oaloop)=LOW(UNSIGN(int_offset MOD 10))+'0';
608   3          int_offset=int_offset/10;
609   3       end;
610   2       call write_message;

611   2       END OFFSET_ASCII;

612   1       WRITE_MESSAGE: PROCEDURE;

613   2       call RQ$SEND$MESSAGE(th_in_mbx,wrt_buf_vals.base,th_out_mbx,@status);
614   2       wrt_buf_vals.base=RQ$RECEIVE$MESSAGE(th_out_mbx,wait_forever,no_response,@status);

615   2       END WRITE_MESSAGE;



616   1       SET_ERROR_HANDLER: PROCEDURE;

617   2       declare e_h structure (offset word,base word,mode byte);

618   2       e_h.offset,e_h.base,e_h.mode=null;
619   2       call RQ$SET$EXCEPTION$HANDLER(@e_h,@status);

620   2       END SET_ERROR_HANDLER;

              $EJECT
```

```
                /******************************* MAIN MODULE ************************************/

                /* The path task comprises of two source files, one written in PL/M 86 and one
                   written in ASM86. The main  body of the path  library is just file handling
                   as in the data task and this is all  present in this  PL/M 86  source file.
                   The ASM86  source file  contains two external routines  which are linked to
                   this file and are the  drivers for the  robot and key  pendent in input and
                   playback modes. This task when  initially created looks up all the required
                   tokens and creates its own  communications buffer. Then as in the data task
                   it waits at its own  semaphore for a single unit from the  Supervisor task,
                   when it  receives this unit it immediately suspends the  Supervisor task to
                   prevent it  from  continuing to  run. The  disk  connection  token  is then
                   looked up in case it has changed since the last time the path librarian ran
                   and then the  main path  directory is  attached. Once this is done the task
                   displays the path librarians prompt line and waits at the read task mailbox
                   for a  command. The  command is identified and the correct utility routines
                   are called. When LOAD has been used or a single  carriage  return is  typed
                   the path  librarian  deletes  its  connection with the root path directory,
                   resumes the Supervisor task and then goes back to waiting for the next unit
                   at the path semaphore. */

621   1         PATH_TASK: PROCEDURE PUBLIC;

622   2         gen_mailbox=RQ$LOOKUP$OBJECT(this_job,@(4,'GMBX'),wait_forever,@status);
623   2         path_mailbox=RQ$LOOKUP$OBJECT(this_job,@(4,'PMBX'),wait_forever,@status);
624   2         main_task_token=RQ$LOOKUP$OBJECT(this_job,@(4,'MTSK'),wait_forever,@status);
625   2         th_in_mbx=RQ$LOOKUP$OBJECT(this_job,@(7,'THINMBX'),wait_forever,@status);
626   2         th_out_mbx=RQ$LOOKUP$OBJECT(this_job,@(8,'THOUTMBX'),wait_forever,@status);
627   2         att_file_mbx=RQ$LOOKUP$OBJECT(this_job,@(6,'ATTMBX'),wait_forever,@status);
628   2         list_dir_mbx=RQ$LOOKUP$OBJECT(this_job,@(5,'LDMBX'),wait_forever,@status);
629   2         cre_file_mbx=RQ$LOOKUP$OBJECT(this_job,@(6,'CREMBX'),wait_forever,@status);
630   2         read_in=RQ$LOOKUP$OBJECT(this_job,@(6,'READIN'),wait_forever,@status);
631   2         read_out=RQ$LOOKUP$OBJECT(this_job,@(7,'READOUT'),wait_forever,@status);
632   2         co=RQ$LOOKUP$OBJECT(this_job,@(2,'CO'),wait_forever,@status);
633   2         path_sem=RQ$LOOKUP$OBJECT(this_job,@(4,'PSEM'),wait_forever,@status);
634   2         wrt_buf_vals.base=RQ$CREATE$SEGMENT(80,@status);
635   2         wrt_buf_vals.offset=null;
636   2         CALL SET_ERROR_HANDLER;
                $EJECT
```

```
637  2       do forever;
638  3         bytes_read=RQ$RECEIVE$UNITS(path_sem,one,wait_forever,@status);
639  3         call RQ$SUSPEND$TASK(main_task_token,@status);
640  3         call SAVE$REAL$STATUS(@path_real_save_area);
641  3         disk_token=RQ$LOOKUP$OBJECT(this_job,@(4,'DCON'),wait_forever,@status);
642  3         call RQ$A$ATTACHFILE(id,disk_token,@(7,'PATHDEF'),gen_mailbox,@status);
643  3         def_connection=RQ$RECEIVE$MESSAGE(gen_mailbox,wait_forever,no_response,@status);
644  3         type_code=RQ$GET$TYPE(def_connection,@status);
645  3         if type_code<>connection_code then call RQ$DELETE$SEGMENT(def_connection,@status);
647  3         else
                 do;
648  4              done=bad;
649  4              do while not done;
650  5                 call MOVB(@question3,@wrt_buf,77);
651  5                 call write_message;
652  5                 call RQ$SEND$MESSAGE(read_in,wrt_buf_vals.base,read_out,@status);
653  5                 wrt_buf_vals.base=RQ$RECEIVE$MESSAGE(read_out,wait_forever,no_response,@status);
654  5                 if read_buf(0)=null then done=good;
656  5                  else
                        do;
657  6                       if CMPB(@('DELETE'),@read_buf(1),6)=match then call path_delete;
659  6                       if CMPB(@('INPUT'),@read_buf(1),5)=match then call path_input;
661  6                       if CMPB(@('EDIT'),@read_buf(1),4)=match then call path_edit;
663  6                       if CMPB(@('TRACE'),@read_buf(1),5)=match then call path_trace;
665  6                       if CMPB(@('MOVE'),@read_buf(1),4)=match then call position;
667  6                       if CMPB(@('DIR'),@read_buf(1),3)=match then call path_dir;
669  6                       if CMPB(@('SCALE'),@read_buf(1),5)=match then call path_scale;
671  6                       if CMPB(@('LIST'),@read_buf(1),4)=match then call path_list;
673  6                       if CMPB(@('LOAD'),@read_buf(1),4)=match then
674  6                         do;
675  7                            done=good;
676  7                            call path_load;
677  7                         end;
678  6                     end;
679  5              end;
680  4           end;
681  3         call RQ$A$DELETE$CONNECTION(def_connection,no_response,@status);
682  3         call clear_the_screen;
683  3         call RESTORE$REAL$STATUS(@path_real_save_area);
684  3         call RQ$RESUME$TASK(main_task_token,@status);
685  3       end;

686  2       END PATH_TASK;

687  1       END PATH_LIBRARY;

              $EJECT
```

MODULE INFORMATION:

```
     CODE AREA SIZE      = 181AH   6170D
     CONSTANT AREA SIZE  = 0272H    626D
     VARIABLE AREA SIZE  = 00D3H    211D
     MAXIMUM STACK SIZE  = 0024H     36D
     1252 LINES READ
     0 PROGRAM WARNINGS
     0 PROGRAM ERRORS
```

DICTIONARY SUMMARY:

```
     114KB MEMORY AVAILABLE
     18KB MEMORY USED   (15%)
     0KB DISK SPACE USED
```

END OF PL/M-86 COMPILATION

4. ROBOT AND PENDANT KEYBOARD DRIVER.

```
iRMX 86 8086/87/88/186 MACRO ASSEMBLER V2.0 ASSEMBLY OF MODULE MOTOR_DRIVERS
OBJECT MODULE PLACED IN DRIVERS/MOTS.OBJ
ASSEMBLER INVOKED BY:  :LANG:asm86 DRIVERS/MOTS.A86

LOC  OBJ                  LINE     SOURCE

                             1 +1  $DEBUG
                             2 +1  $INCLUDE(INC/LITERALS.A86)
                     =1      3 +1  $SAVE NOLIST
                            69 +1  $TITLE ('SUPERWELD PATH INPUT, EDITOR AND TRACE DRIVER VER 86/4.2 DATE 10/1/85')
                            70
                            71     NAME MOTOR_DRIVERS
                            72
                            73     ASSUME CS:CODE,DS:DATA
                            74
----                        75     DATA SEGMENT PARA PUBLIC 'DATA'
                            76
0000 FFFF                   77     SPEEDS2 DW      0FFFFH
0002 F805                   78     SPEEDS  DW      05F8H,02FCH,01FDH,017EH,0131H,00FEH,00DAH,0C0H
0004 FC02
0006 FD01
0008 7E01
000A 3101
000C FE00
000E DA00
0010 C000
0012 0100                   79     PULSES  DW      1H   ,2H   ,3H   ,4H   ,5H   ,6H   ,7H   ,8H
0014 0200
0016 0300
0018 0400
001A 0500
001C 0600
001E 0700
0020 0800
0022 (200                   80     XARRAY  DW      200     DUP     (?)             ;OFFSET POINT MATRIX FOR PATH.
     ????
     )
01B2 (200                   81     YARRAY  DW      200     DUP     (?)             ;
     ????
     )
0342 (200                   82     ZARRAY  DW      200     DUP     (?)             ;
     ????
     )
04D2 (200                   83     ROARRAY DW      200     DUP     (?)             ;
     ????
     )
0662 (200                   84     R1ARRAY DW      200     DUP     (?)             ;
     ????
     )
07F2 (200                   85     R2ARRAY DW      200     DUP     (?)             ;
     ????
```

```
LOC  OBJ                  LINE      SOURCE

0982 ????                   87      XTOTAL  DW      ?                       ;TOTAL COUNTS (DISPLACEMENT
0984 ????                   88      YTOTAL  DW      ?                       ;FROM ORIGIN) DURRING I/P.
0986 ????                   89      ZTOTAL  DW      ?                       ;
0988 ????                   90      R0TOTAL DW      ?                       ;
098A ????                   91      R1TOTAL DW      ?                       ;
098C ????                   92      R2TOTAL DW      ?                       ;
098E ????                   93      XSPEED  DW      ?                       ;SPEED STORES FOR TRACER.
0990 ????                   94      YSPEED  DW      ?                       ;
0992 ????                   95      ZSPEED  DW      ?                       ;
0994 ????                   96      R0SPEED DW      ?                       ;
0996 ????                   97      R1SPEED DW      ?                       ;
0998 ????                   98      R2SPEED DW      ?                       ;
                            99
                           100 +1   $EJECT
```

```
LOC  OBJ                  LINE     SOURCE

                          101
                          102      ;BELOW DATA CLEARED BY INPUT & TRACE ON ENTRY
                          103
099A ????                 104      XSTEPS  DW      ?                       ;COUNT STORAGE OF PRESENT OFFSET.
099C ????                 105      YSTEPS  DW      ?                       ;
099E ????                 106      ZSTEPS  DW      ?                       ;
09A0 ????                 107      R0STEPS DW      ?                       ;
09A2 ????                 108      R1STEPS DW      ?                       ;
09A4 ????                 109      R2STEPS DW      ?                       ;
09A6 ????                 110      STATUS  DW      ?                       ;ERROR WORD FOR RMX CALLS.
09A8 ????                 111      TIMCNT  DW      ?                       ;NUMBER OF 10mS PULSES.
09AA ????                 112      TEMPOFF DW      ?                       ;TEMP OFFSET STORE FOR 8087.
09AC ????                 113      REALSTW DW      ?                       ;8087 STATUS WORD STORAGE.
09AE ??                   114      CONTROL DB      ?                       ;CONTROL BYTE.
09AF ??                   115      WELDON  DB      ?                       ;WELD INDICATOR (0=OFF).
09B0 ??                   116      SPLNON  DB      ?                       ;SPLINE INDICATOR (0=OFF).
09B1 ??                   117      FINISH  DB      ?                       ;END OF PATH FLAG (SET=FINISHED).
09B2 ??                   118      KEYPRS  DB      ?                       ;DEBOUNCE BYTE FOR SPECIAL KEYS.
09B3 ??                   119      AUTOFLG DB      ?                       ;FLAG SET IF AUTO MOVE IS ACTIVE.
09B4 ??                   120      SIGNF   DB      ?                       ;DIRECTION FLAGS FOR AUTO MOVE.
09B5 ??                   121      ENABLEF DB      ?                       ;ENABLE FLAGS STORE FOR AUTO MOVE.
09B6 ??                   122      AUTOPT  DB      ?                       ;FLAG SET WHEN AUTO POINT REQUIRED.
09B7 ??                   123      MOVSTAT DB      ?                       ;MOVE STATUS FOR TRACE.
09B8 ??                   124      INS     DB      ?                       ;INDICATES INSERT IN TRACE
09B9 ??                   125      MOTLOAD DB      ?                       ;
09BA ??                   126      ROT_ON  DB      ?
09BB ??                   127      ROT_FLAG DB     ?
09BC ??                   128      XYZ_FLAG DB     ?
                          129
                          130 +1   $EJECT
```

```
LOC  OBJ                  LINE     SOURCE

                          131
                          132      ;CLEARED BY INPUT ONLY
                          133
09BD ??                   134      SPLNCNT DB      ?                       ;SPLINE COUNT.
09BE ??                   135      WELDCNT DB      ?                       ;WELD COUNT.
09BF ????                 136      PTCOUNT DW      ?                       ;TOTAL MATRIX POINT COUNT.
09C1 ????                 137      PTBYTES DW      ?                       ;TOTAL BYTE FOR MATRIX LEG.
                          138
09C3 0304           R     139      SKEYTABLE       DW      FAST            ;SPECIAL KEY ADDRESS
09C5 E703           R     140                      DW      SLOW            ;JUMP TABLE FOR PATH
09C7 A203           R     141                      DW      WELDPT          ;INPUT ROUTINE.
09C9 D303           R     142                      DW      SPLINEPT        ;
09CB 2D03           R     143                      DW      ENTERPT         ;
09CD 2C04           R     144                      DW      ENDKEY          ;
09CF E904           R     145                      DW      FALSEX          ;
09D1 9602           R     146                      DW      MOVE_PT         ;
                          147
09D3 D007                 148      MINRD   DW      2000                    ;MINIMUM AUTO RETURN DISTANCE.
09D5 0006                 149      FREQ    DW      1536                    ;CLOCK FREQUENCY OF 8253'S.
09D7 0600                 150      TRACSPD DW      6                       ;TRACING VELOCITY FOR ROBOT.
09D9 ??                   151      PINPUT  DB      ?                       ;SPECIAL KEY MASK SET BY PL/M.
                          152
                          153      PUBLIC  PINPUT,PTBYTES
                          154      PUBLIC  SPLNCNT,WELDCNT
                          155      PUBLIC  PTCOUNT,XARRAY
                          156      PUBLIC  YARRAY,ZARRAY
                          157      PUBLIC  R0ARRAY,R1ARRAY
                          158      PUBLIC  R2ARRAY,ROT_FLAG
                          159
                          160      DATA ENDS
                          161
                          162      EXTRN   RQEXITINTERRUPT:FAR,RQSETINTERRUPT:FAR,RQRESETINTERRUPT:FAR
                          163
                          164 +1   $EJECT
```

```
LOC  OBJ                  LINE     SOURCE

                          165
----                      166     CODE SEGMENT PARA PUBLIC 'CODE'
                          167
0000                      168     GET_KEY         PROC
                          169                     ; This routine completely scans the key pendant which is a
                          170                     ; crosswire matrix attached to two ports, one output, one
                          171                     ; input. The information about which keys are at present
                          172                     ; pressed is returned in the AX register for the motor keys
                          173                     ; and the DL register for the special keys.
0000 51                   174                     PUSH    CX                      ;SAVE CX
0001 B401                 175                     MOV     AH,ONE                  ;RUNNING ZERO (PORT INVERTS)
0003 8AC4                 176                     MOV     AL,AH                   ;OUTPUT 00000001
0005 E6CC                 177                     OUT     KEYROW,AL               ;
0007 E4C8                 178                     IN      AL,KEYCOL               ;INPUT ROW= X X R3 R2 R1 R0 S1 S0
0009 D0E4                 179                     SHL     AH,1                    ;MOVE UP THE RUNNING ZERO
000B 243F                 180                     AND     AL,BIT76MASK            ;AL = 0 0 R3 R2 R1 R0 S1 S0
000D D0E8                 181                     SHR     AL,1                    ;S0 TO CARRY
000F D0D2                 182                     RCL     DL,1                    ;DL = XXXXXXXXXXXXXXXS0
0011 D0E8                 183                     SHR     AL,1                    ;S1 TO CARRY
0013 D0D2                 184                     RCL     DL,1                    ;DL = XXXXXXS0 S1
0015 D0E0                 185                     SHL     AL,1                    ;
0017 D0E0                 186                     SHL     AL,1                    ;
0019 D0E0                 187                     SHL     AL,1                    ;
001B D0E0                 188                     SHL     AL,1                    ;
001D 8AE8                 189                     MOV     CH,AL                   ;MOVE AL TO TOP NIBBLE CH
001F 8AC4                 190                     MOV     AL,AH                   ;
0021 E6CC                 191                     OUT     KEYROW,AL               ;
0023 E4C8                 192                     IN      AL,KEYCOL               ;
0025 D0E4                 193                     SHL     AH,1                    ;
0027 243F                 194                     AND     AL,BIT76MASK            ;
0029 D0E8                 195                     SHR     AL,1                    ;
002B D0D2                 196                     RCL     DL,1                    ;
002D D0E8                 197                     SHR     AL,1                    ;
002F D0D2                 198                     RCL     DL,1                    ;
0031 0AE8                 199                     OR      CH,AL                   ;MOVE AL TO BOTTOM NIBBLE CH
                          200 +1  $EJECT
```

```
LOC  OBJ                  LINE     SOURCE

0033 8AC4                  201                    MOV     AL,AH                   ;
0035 E6CC                  202                    OUT     KEYROW,AL               ;
0037 E4C8                  203                    IN      AL,KEYCOL               ;
0039 D0E4                  204                    SHL     AH,1                    ;
003B 243F                  205                    AND     AL,BIT76MASK            ;
003D D0E8                  206                    SHR     AL,1                    ;
003F D0D2                  207                    RCL     DL,1                    ;
0041 D0E8                  208                    SHR     AL,1                    ;
0043 D0D2                  209                    RCL     DL,1                    ;
0045 D0E0                  210                    SHL     AL,1                    ;
0047 D0E0                  211                    SHL     AL,1                    ;
0049 D0E0                  212                    SHL     AL,1                    ;
004B D0E0                  213                    SHL     AL,1                    ;
004D 8AC8                  214                    MOV     CL,AL                   ;MOVE AL TO TOP NIBBLE CL
004F 8AC4                  215                    MOV     AL,AH                   ;
0051 E6CC                  216                    OUT     KEYROW,AL               ;
0053 E4C8                  217                    IN      AL,KEYCOL               ;
0055 243F                  218                    AND     AL,BIT76MASK            ;
0057 D0E8                  219                    SHR     AL,1                    ;
0059 D0D2                  220                    RCL     DL,1                    ;
005B D0E8                  221                    SHR     AL,1                    ;
005D D0D2                  222                    RCL     DL,1                    ;
005F 0AC8                  223                    OR      CL,AL                   ;MOVE AL TO BOTTOM NIBBLE CL
0061 8BC1                  224                    MOV     AX,CX                   ;RESULT INTO AX
0063 59                    225                    POP     CX                      ;RESTORE CX
0064 C3                    226                    RET
                           227        GET_KEY     ENDP
                           228
                           229 +1     $EJECT
```

```
LOC  OBJ                  LINE     SOURCE

                          230
                          231      ;*****************************************************************************
                          232      ;*                                                                           *
                          233      ;*              MAIN ENTERANCE TO ASM86 ROUTINES FOR PL/M                    *
                          234      ;*                                                                           *
                          235      ;*****************************************************************************
                          236
0065                      237      TRACK           PROC    FAR                     ;CALLED BY PL/M
                          238                      ; This is the routine entrance for the trace and edit routines.
                          239                      ; The two are distiguishable by the contents of PINPUT.
                          240                      PUBLIC  TRACK                   ;
0065 BACD0590       R     241                      MOV     DX,OFFSET TRCINT        ;TRACE INTERRUPT ENTERANCE.
0069 EB04                 242                      JMP     SHORT MAINSTART         ;JUMP TO MAIN START
006B                      243      KEY_DRIVER      PROC    FAR                     ;CALLED BY PL/M
                          244                      ; This is the entrance for the move and input routines,
                          245                      ; again they can be separated from each other by the contents
                          246                      ; of the variable PINPUT.
                          247                      PUBLIC  KEY_DRIVER              ;
006B BAF30090       R     248                      MOV     DX,OFFSET KEYINT        ;INPUT INTERRUPT ENTRANCE.
006F 1E                   249      MAINSTART:      PUSH    DS                      ;SAVE OLD DATA SEGMENT
0070 BB----         R     250                      MOV     BX,DATA                 ;GET NEW DATA SEGMENT
0073 8EDB                 251                      MOV     DS,BX                   ;AND ES SEGMENT
0075 8EC3                 252                      MOV     ES,BX                   ;FOR STRING OPERATIONS
0077 52                   253                      PUSH    DX                      ;SAVE ROUTINE IDENTIFIER
0078 33DB                 254                      XOR     BX,BX                   ;BX = 0
007A B84800               255                      MOV     AX,LEVELM4              ;SET INTERRUPT MASTER LEVEL FOUR
007D 50                   256                      PUSH    AX                      ;
007E 53                   257                      PUSH    BX                      ;0 = NOT AN INTERRUPT TASK
007F 0E                   258                      PUSH    CS                      ;INTERRUPT ROUTINE CODE SEGMENT
0080 52                   259                      PUSH    DX                      ;INTERRUPT ROUTINE OFFSET
0081 53                   260                      PUSH    BX                      ;0 = NO DATA SEGMENT
0082 1E                   261                      PUSH    DS                      ;DATA SEGMENT AND
0083 B8A609         R     262                      MOV     AX,OFFSET STATUS        ;OFFSET ADDRESS FOR STATUS WORD
0086 50                   263                      PUSH    AX                      ;
0087 9A0000----     E     264                      CALL    ROSETINTERRUPT          ;INTERRUPT VECTOR SET
008C 5A                   265                      POP     DX                      ;GET ROUTINE IDENTIFIER
                          266 +1   $EJECT
```

```
LOC  OBJ                  LINE     SOURCE

008D 32C0                  267                 XOR     AL,AL                   ;CLEAR AL
008F 81FACD05        R     268                 CMP     DX,OFFSET TRCINT        ;IN TRACE MODE ?
0093 7422                  269                 JZ      NOCLEAR                 ;YES.......
0095 803ED90923      R     270                 CMP     PINPUT,MOVE_MASK        ;ARE WE IN MOVE ?
009A 741B                  271                 JZ      NOCLEAR                 ;YES.......
009C B90600                272                 MOV     CX,6                    ;6 BYTE CLEAR FOR INPUT ONLY
009F 8D3EBD09        R     273                 LEA     DI,SPLNCNT              ;CLEARING COUNTERS.
00A3 F3                    274                 REP     STOSB                   ;
00A4 AA
00A5 9BD9EE                275                 FLDZ                            ;ST =0.0    FOR LINE MATHS
00A8 9BDDD1                276                 FST     ST(1)                   ;ST(1)=0.0
00AB 9BDDD2                277                 FST     ST(2)                   ;ST(2)=0.0
00AE 9BDDD3                278                 FST     ST(3)                   ;ST(3)=0.0
00B1 9BDDD4                279                 FST     ST(4)                   ;ST(4)=0.0
00B4 9BDDD5                280                 FST     ST(5)                   ;ST(5)=0.0
00B7 B92300                281     NOCLEAR:    MOV     CX,35                   ;35 BYTES TO CLEAR (I/P OR MOVE)
00BA 8D3E9A09        R     282                 LEA     DI,XSTEPS               ;FIRST BYTE TO BE CLEARED
00BE F3                    283                 REP     STOSB                   ;CLEAR ALL INITIAL VARIABLES
00BF AA
00C0 8E0C00                284                 MOV     SI,DEFSPEED             ;STARTING SPEED INDEX IN SI
00C3 BF0000                285                 MOV     DI,00                   ;DI = 0 (MATRIX INDEX)
00C6 B070                  286                 MOV     AL,MODEWORD             ;PUT TIMER 1 ON 86/30
00C8 E6D6                  287                 OUT     TMRCNTRL,AL             ;IN MODE 0
00CA B8B75F                288                 MOV     AX,TIME10mS             ;LOAD 10mS COUNT
00CD E6D2                  289                 OUT     TMRINPUT,AL             ;OUTPUT LSB
00CF 8AC4                  290                 MOV     AL,AH                   ;GET MSB
00D1 E6D2                  291                 OUT     TMRINPUT,AL             ;OUTPUT MSB
00D3 B009                  292                 MOV     AL,GENABLE              ;8253 GATE PORT CONTROL WORD
00D5 E6C9                  293                 OUT     GATEPORT,AL             ;COUNTER ENABLED.
00D7 F4                    294     INT_LOOP:   HLT                             ;WAIT FOR INTERRUPT.
00D8 F606B10901      R     295                 TEST    FINISH,01H              ;FINISH SET ?
00DD 74F8                  296                 JZ      INT_LOOP                ;NO.......
00DF B001                  297                 MOV     AL,1                    ;COUNTER1 GATE OFF
00E1 E6C9                  298                 OUT     GATEPORT,AL             ;COUNTER DISABLED
00E3 B84800                299                 MOV     AX,LEVELM4              ;INTERRUPT MASTER LEVEL 4
00E6 50                    300                 PUSH    AX                      ;
00E7 1E                    301                 PUSH    DS                      ;DATA SEGMENT AND
00E8 B8A609          R     302                 MOV     AX,OFFSET STATUS        ;OFFSET OF STATUS WORD.
00EB 50                    303                 PUSH    AX                      ;
00EC 9A0000----      E     304                 CALL    RQRESETINTERRUPT        ;INTERRUPT VECTOR CLEARED.
00F1 1F                    305                 POP     DS                      ;RESTORE OLD DATA SEGMENT.
00F2 CB                    306                 RET                             ;RETURN TO PLM86.
                           307     KEY_DRIVER  ENDP
                           308     TRACK       ENDP
                           309
                           310 +1  $EJECT
```

```
LOC  OBJ                  LINE     SOURCE

                          311
                          312      ;*****************************************************************************
                          313      ;*                                                                           *
                          314      ;*              INTERRUPT ENTRANCE AT CS:KEYINT LABEL FAR                    *
                          315      ;*                                                                           *
                          316      ;*****************************************************************************
                          317
00F3                      318      KEYINT          LABEL   FAR                     ;INTERRUPT ENTRANCE POINT FOR
                          319                      PUBLIC  KEYINT                  ;THE INPUT AND MOVE ROUTINES.
00F3 B8B75F               320                      MOV     AX,TIME10mS             ;RELOAD 10mS TIME INTERVAL
00F6 E6D2                 321                      OUT     TMRINPUT,AL             ;LSB
00F8 8AC4                 322                      MOV     AL,AH                   ;
00FA E6D2                 323                      OUT     TMRINPUT,AL             ;MSB
00FC F606R30901      R    324                      TEST    AUTOFLG,1               ;AUTO MOVEMENT REQUIRED ?
0101 7403                 325                      JZ      NAUTOMOV                ;NO.........
0103 E9DA04               326                      JMP     NOT_INS                 ;YES......... (AT END OF INPUT).
0106 E8F7FE               327      NAUTOMOV:       CALL    GET_KEY                 ;SCAN THE KEYBOARD.
0109 23C0                 328                      AND     AX,AX                   ;IF MOTOR KEYS ARE PRESSED THEN
010B 7403                 329                      JZ      NO_MOTS                 ;IGNORE ALL THE SPECIAL KEYS
010D 80E203               330                      AND     DL,3                    ;EXCEPT THE FAST AND SLOW KEYS.
0110 52                   331      NO_MOTS:        PUSH    DX                      ;SAVE SPECIAL KEY WORD ON STACK.
0111 32C4                 332      MOTORS:         XOR     AL,AH                   ;ENABLES FOR 8 MOTORS
0113 22E0                 333                      AND     AH,AL                   ;DIRECTIONS FOR ALL 8
0115 8BD8                 334                      MOV     BX,AX                   ;BX=XXXDDDDDDXXEEEEEE
0117 32C0                 335                      XOR     AL,AL                   ;CLEAR AL
0119 BA0804               336                      MOV     DX,ENBPRTADDR           ;GET THE ENABLE PORT ADDRESS
011C EE                   337                      OUT     DX,AL                   ;TURN OFF COUNTER TO SYNC
011D F606R90901      R    338                      TEST    MOTLOAD,1               ;DO COUNTERS NEED RELOADING ?
0122 750A                 339                      JNZ     LOADED                  ;NO............
0124 8B840200        R    340                      MOV     AX,SPEEDS [SI]          ;LOAD THE PRESENT SPEED INTO
0128 E8C000               341                      CALL    XYZ_MOTORS              ;AX AND RELOAD THE XYZ AND
012B E8D700               342                      CALL    ROT_MOTORS              ;THE ROTATION AXES.
012E BA0004               343      LOADED:         MOV     DX,DIRPRTADDR           ;DIRECTION PORT ADDRESS
0131 8AC7                 344                      MOV     AL,BH                   ;GET MOTOR DIRECTIONS
0133 EE                   345                      OUT     DX,AL                   ;MOTOR DIRECTIONS NOW SET
0134 E8EE00               346                      CALL    CHECK_ENABLES           ;
0137 BA0804               347                      MOV     DX,ENBPRTADDR           ;GET ENABLE PORT ADDR
013A 8AC3                 348                      MOV     AL,BL                   ;GET ENABLE BYTE
013C EE                   349                      OUT     DX,AL                   ;START MOTORS
013D A0D909          R    350                      MOV     AL,PINPUT               ;GET ROUTINE MASK
0140 A8DC                 351                      TEST    AL,NOT MOVE_MASK        ;ARE WE IN MOVE?
0142 7460                 352                      JZ      SKEYS                   ;YES...THEN WE ARE NOT IN I/P
                          353 +1   $EJECT
```

```
LOC  OBJ                  LINE     SOURCE

                          354                    ; The following portion of code adds the number of steps about
                          355                    ; to be done to the present offset counts and to the relevent
                          356                    ; variable on the 8087 stack so that the total offset from the
                          357                    ; origin may be kept track of.
0144 8B0E9A09       R     358                    MOV     CX,XSTEPS               ;LOAD XSTEPS
0148 E80E01               359                    CALL    ADD_PULSES              ;ADD STEPS ABOUT TO BE DONE
014B 890E9A09       R     360                    MOV     XSTEPS,CX               ;SAVE X OFFSET
014F 8B0E9C09       R     361                    MOV     CX,YSTEPS               ;LOAD YSTEPS
0153 9BD9C9               362                    FXCH                            ;ST-ST(1)
0156 E80001               363                    CALL    ADD_PULSES              ;ADD STEPS ABOUT TO BE DONE
0159 9BD9C9               364                    FXCH                            ;ST-ST(1)
015C 890E9C09       R     365                    MOV     YSTEPS,CX               ;SAVE Y OFFSET
0160 8B0E9E09       R     366                    MOV     CX,ZSTEPS               ;LOAD ZSTEPS
0164 9BD9CA               367                    FXCH    ST(2)                   ;ST(1)-ST(2)
0167 E8EF00               368                    CALL    ADD_PULSES              ;ADD STEPS ABOUT TO BE DONE
016A 9BD9CA               369                    FXCH    ST(2)                   ;ST(1)-ST(2)
016D 890E9E09       R     370                    MOV     ZSTEPS,CX               ;SAVE Z OFFSET
0171 8B0EA009       R     371                    MOV     CX,R0STEPS              ;LOAD ZSTEPS
0175 9BD9CB               372                    FXCH    ST(3)                   ;ST(1)-ST(2)
0178 E8DE00               373                    CALL    ADD_PULSES              ;ADD STEPS ABOUT TO BE DONE
017B 9BD9CB               374                    FXCH    ST(3)                   ;ST(1)-ST(2)
017E 890EA009       R     375                    MOV     R0STEPS,CX              ;SAVE Z OFFSET
0182 8B0EA209       R     376                    MOV     CX,R1STEPS              ;LOAD ZSTEPS
0186 9BD9CC               377                    FXCH    ST(4)                   ;ST(1)-ST(2)
0189 E8CD00               378                    CALL    ADD_PULSES              ;ADD STEPS ABOUT TO BE DONE
018C 9BD9CC               379                    FXCH    ST(4)                   ;ST(1)-ST(2)
018F 890EA209       R     380                    MOV     R1STEPS,CX              ;SAVE Z OFFSET
0193 8B0EA409       R     381                    MOV     CX,R2STEPS              ;LOAD ZSTEPS
0197 9BD9CD               382                    FXCH    ST(5)                   ;ST(1)-ST(2)
019A E8BC00               383                    CALL    ADD_PULSES              ;ADD STEPS ABOUT TO BE DONE
019D 9BD9CD               384                    FXCH    ST(5)                   ;ST(1)-ST(2)
01A0 890EA409       R     385                    MOV     R2STEPS,CX              ;SAVE Z OFFSET
                          386 +1  $EJECT
```

```
LOC  OBJ                  LINE     SOURCE

                           387                 ; This next part of code deals with the special keys from
                           388                 ; the key pendant. The routines allowable are flagged in
                           389                 ; the variable PINPUT and so the unwanted are masked off.
                           390                 ; The keys are then debounced for 30 mS and then the jump
                           391                 ; table driven routine executes the correct routine.
01A4 5A                    392     SKEYS:      POP     DX                  ;GET SPECIAL KEY WORD BACK.
01A5 2216D909        R     393                 AND     DL,PINPUT           ;ANY SPECIAL KEYS PRESSED
01A9 7507                  394                 JNZ     DEBOUNCE            ;YES........
01AB C606B20900      R     395                 MOV     KEYPRS,0            ;CLEAR DEBOUNCE COUNTER AND
01B0 EB1C                  396                 JMP     SHORT CHECK_AUTO    ;MISS SPECIAL KEY ROUTINE.
01B2 803EB20903      R     397     DEBOUNCE:   CMP     KEYPRS,3            ;IS THE 30mS DEBOUNCE OVER?
01B7 7511                  398                 JNE     REPEATKEY           ;NO......SO COUNT ANOTHER 10mS
01B9 8D1EC109        R     399                 LEA     BX,SKEYTABLE-2      ;LOAD SPECIAL KEY JUMP TABLE
01BD 83C302                400     NEXTSKEY:   ADD     BX,2                ;ADDRESS. INC FOR NEXT ROUTINE.
01C0 D0EA                  401                 SHR     DL,1                ;LOOK AT EACH S KEY IN TURN
01C2 7304                  402                 JNC     ENDTEST             ;KEY NOT SET SO TRY THE NEXT.
01C4 FF17                  403                 CALL    WORD PTR [BX]       ;KEY SET, JUMP TO ROUTINE.
01C6 33DB                  404                 XOR     BX,BX               ;CLEAR BX SO NEXT TEST FAILS.
01C8 75F3                  405     ENDTEST:    JNZ     NEXTSKEY            ;FAILS IF 1.CALL DONE 2.DL=0
01CA FE06B209        R     406     REPEATKEY:  INC     KEYPRS              ;COUNT ANOTHER 10mS DEBOUNCE.
01CE 803EB60900      R     407     CHECK_AUTO: CMP     AUTOPT,0            ;IS AN AUTOPOINT REQUIRED ?
01D3 7403                  408                 JZ      INTER_RET           ;NO.....
01D5 E85501                409                 CALL    ENTERPT             ;YES.....SO ENTER A POINT.
01D8 B84800                410     INTER_RET:  MOV     AX,LEVELM4          ;MASTER INTERRUPT LEVEL 4
01DB BBA609          R     411                 MOV     BX,OFFSET STATUS    ;STATUS RELATIVE ADDRESS.
01DE 56                    412                 PUSH    SI                  ;SAVE SI AND DI.
01DF 57                    413                 PUSH    DI                  ;
01E0 50                    414                 PUSH    AX                  ;
01E1 1E                    415                 PUSH    DS                  ;
01E2 53                    416                 PUSH    BX                  ;
01E3 9A0000----      E     417                 CALL    ROEXITINTERRUPT     ;TELL PIC THE INTERRUPT HAS
01E8 5F                    418                 POP     DI                  ;BEEN PROCESSED.
01E9 5E                    419                 POP     SI                  ;RESTORE SI AND DI.
01EA CF                    420                 IRET                        ;
                           421
                           422 +1  $EJECT
```

```
LOC  OBJ                  LINE     SOURCE

                           423
01EB                       424     XYZ_MOTORS      PROC
                           425                     ; This routine loads all the xyz counters with the count in
                           426                     ; the AX register so that the correct speeds are given out.
01EB BA1404                427                     MOV     DX,TIMER3               ;
01EE EE                    428                     OUT     DX,AL                   ;OUT LSB
01EF 86C4                  429                     XCHG    AL,AH                   ;SWOP LSB & MSB
01F1 EE                    430                     OUT     DX,AL                   ;OUT MSB
01F2 BA1204                431                     MOV     DX,TIMER2               ;NEXT TIMER ADDRESS
01F5 86C4                  432                     XCHG    AL,AH                   ;SWOP LSB & MSB
01F7 EE                    433                     OUT     DX,AL                   ;
01F8 86C4                  434                     XCHG    AL,AH                   ;SWOP LSB & MSB
01FA EE                    435                     OUT     DX,AL                   ;OUT MSB
01FB BA1004                436                     MOV     DX,TIMER1               ;NEXT TIMER ADDRESS
01FE 86C4                  437                     XCHG    AL,AH                   ;SWOP LSB & MSB
0200 EE                    438                     OUT     DX,AL                   ;
0201 86C4                  439                     XCHG    AL,AH                   ;SWOP LSB & MSB
0203 EE                    440                     OUT     DX,AL                   ;OUT MSB
0204 C3                    441                     RET
                           442     XYZ_MOTORS      ENDP
                           443
                           444 +1  $EJECT
```

```
LOC  OBJ                  LINE     SOURCE

                           445
0205                       446     ROT_MOTORS      PROC
                           447                     ; This routine loads all the rotation counters with the count
                           448                     ; in the AX register so that the correct speeds are given out.
0205 BA1804                449                     MOV     DX,TIMER4               ;ALL THREE COUNTERS.
0208 86C4                  450                     XCHG    AL,AH                   ;
020A EE                    451                     OUT     DX,AL                   ;OUT LSB
020B 86C4                  452                     XCHG    AL,AH                   ;SWOP LSB & MSB
020D EE                    453                     OUT     DX,AL                   ;OUT MSB
020E BA1A04                454                     MOV     DX,TIMER5               ;NEXT TIMER ADDRESS
0211 86C4                  455                     XCHG    AL,AH                   ;SWOP LSB & MSB
0213 EE                    456                     OUT     DX,AL                   ;
0214 86C4                  457                     XCHG    AL,AH                   ;SWOP LSB & MSB
0216 EE                    458                     OUT     DX,AL                   ;OUT MSB
0217 BA1C04                459                     MOV     DX,TIMER6               ;NEXT TIMER ADDRESS
021A 86C4                  460                     XCHG    AL,AH                   ;SWOP LSB & MSB
021C EE                    461                     OUT     DX,AL                   ;
021D 86C4                  462                     XCHG    AL,AH                   ;SWOP LSB & MSB
021F EE                    463                     OUT     DX,AL                   ;OUT MSB
0220 FE06B909        R     464                     INC     MOTLOAD
0224 C3                    465                     RET
                           466     ROT_MOTORS      ENDP
                           467
                           468 +1  $EJECT
```

```
LOC  OBJ                  LINE     SOURCE

                           469
0225                       470     CHECK_ENABLES   PROC
                           471                     ; The check enables routine makes sure that during input the
                           472                     ; xyz and the rotation movements are mutually exclusive.
                           473                     ; That is either the rotation moves or the xyz axes moves but
                           474                     ; never can the two be moved during the same point as the
                           475                     ; rot_flag and xyz_flag variables dictate.
0225 A0D909          R     476                     MOV     AL,PINPUT               ;GET KEY FLAGS
0228 A8DC                  477                     TEST    AL,NOT MOVE_MASK        ;ARE WE IN MOVE
022A 741A                  478                     JZ      IN_MOVE                 ;YES.....
022C F606BB09FF      R     479                     TEST    ROT_FLAG,0FFH           ;IS ROT FLAG SET
0231 751D                  480                     JNZ     ROTS_ONLY               ;YES.....
0233 F606BC09FF      R     481                     TEST    XYZ_FLAG,0FFH           ;IS XYZ FLAG SET
0238 750D                  482                     JNZ     XYZS_ONLY               ;YES.....
023A 8AC3                  483                     MOV     AL,BL                   ;GET ENABLE FLAGS
023C 2407                  484                     AND     AL,00000111B            ;JUST XYZ ENABLES
023E 7507                  485                     JNZ     XYZS_ONLY
0240 8AC3                  486                     MOV     AL,BL
0242 2438                  487                     AND     AL,00111000B            ;JUST ROT ENABLES
0244 750A                  488                     JNZ     ROTS_ONLY
0246 C3                    489     IN_MOVE:        RET
0247 C606BC09FF      R     490     XYZS_ONLY:      MOV     XYZ_FLAG,0FFH
024C 80E307                491                     AND     BL,00000111B
024F C3                    492                     RET
0250 C606BB09FF      R     493     ROTS_ONLY:      MOV     ROT_FLAG,0FFH
0255 80E338                494                     AND     BL,00111000B
0258 C3                    495                     RET
                           496     CHECK_ENABLES   ENDP
                           497
                           498 +1  $EJECT
```

```
LOC  OBJ                  LINE     SOURCE

                           499
0259                       500     ADD_PULSES      PROC
                           501                     ; The procedure add_pulses adds the number of pulses which
                           502                     ; are about to be done by the motors to ;-
                           503                     ;       a). The total offset of that axis which it assumes
                           504                     ;           is on the top of the real stack.
                           505                     ;       b). The CX register which contains the present offset
                           506                     ;           of that axis since the last point. If this puts the
                           507                     ;           offset near to overflow of the word storage location
                           508                     ;           then the AUTOPT flag is set indicating a point must
                           509                     ;           entered even though the enter button has not been
                           510                     ;           pushed. This should rarely happen as the gearing of
                           511                     ;           axes allows approximately 32.7cm before the variables
                           512                     ;           become close to overflow.
0259 8B841200        R     513                     MOV     AX,PULSES [SI]          ;TO PULSES ABOUT TO BE DONE.
025D D0EF                  514                     SHR     BH,1                    ;GET DIRECTION FROM ENABLEF.
025F 7302                  515                     JNC     POS                     ;DIRECTION POSITIVE
0261 F7D8                  516                     NEG     AX                      ;DIRECTION NEGATIVE
0263 D0EB                  517     POS:            SHR     BL,1                    ;IS THE AXIS ENABLED ?
0265 731E                  518                     JNC     NO_ADD                  ;NO........
0267 A3AA09          R     519                     MOV     TEMPOFF,AX              ;PUT OFFSET IN TEMP STORE FOR
026A 9BDE06AA09      R     520                     FIADD   TEMPOFF                 ;THE NPX TO ADD TO TOTAL OFFSET
026F 03C8                  521                     ADD     CX,AX                   ;ADD STEPS TO AXIS OFFSET
0271 81F9BC7F              522                     CMP     CX,32700                ;MAX DISTANCE MOVED YET ?
0275 7E04                  523                     JLE     POS_OK                  ;NO LESS THAN 327mm MOVED.
0277 FE06B609        R     524                     INC     AUTOPT                  ;YES ....AUTO POINT REQUIRED.
027B 81F94480              525     POS_OK:         CMP     CX,-32700               ;MAX NEG DISTANCE MOVED YET ?
027F 7D04                  526                     JGE     NO_ADD                  ;NO LESS THAN -327mm MOVED.
0281 FE06B609        R     527                     INC     AUTOPT                  ;YES ....AUTO POINT REQUIRED.
0285 C3                    528     NO_ADD:         RET                             ;CX=xSTEPS
                           529     ADD_PULSES      ENDP                            ;
                           530
                           531 +1  $EJECT
```

```
LOC  OBJ                  LINE     SOURCE

                           532
0286                       533     MOVE_PT         PROC
                           534                     ; This routine is insigated after a point has been moved and
                           535                     ; the second move point button has been pressed. The input
                           536                     ; routine is used briefly to get an offset to be added to the
                           537                     ; old offset of the point. This offset adjustment must be then
                           538                     ; subtracted from the next xyz point.
0286 32C0                  539                     XOR     AL,AL                   ;STOP THE MOTORS
0288 BA0804                540                     MOV     DX,ENBPRTADDR           ;
028B EE                    541                     OUT     DX,AL                   ;
028C A2B809          R     542                     MOV     INS,AL                  ;CLEAR THE INS FLAG
028F 83FF00                543                     CMP     DI,0                    ;IS IT THE FIRST POINT
0292 744F                  544                     JZ      END_INSERT
0294 F606BC09FF      R     545                     TEST    XYZ_FLAG,0FFH
0299 7449                  546                     JZ      ROT_MOVE
029B A19A09          R     547                     MOV     AX,XSTEPS
029E 8B1E9C09        R     548                     MOV     BX,YSTEPS
02A2 8B0E9E09        R     549                     MOV     CX,ZSTEPS
02A6 018D4003        R     550                     ADD     ZARRAY[DI-2],CX
02AA 019DB001        R     551                     ADD     YARRAY[DI-2],BX
02AE 01852000        R     552                     ADD     XARRAY[DI-2],AX
02B2 57                    553                     PUSH    DI
02B3 81BD42030080    R     554     TRY_NEXT_PT:    CMP     ZARRAY[DI],8000H
02B9 7502                  555                     JNZ     NEXT_OFFSET
02BB 47                    556                     INC     DI
02BC 47                    557                     INC     DI
02BD 50                    558     NEXT_OFFSET:    PUSH    AX
02BE 8B85D204        R     559                     MOV     AX,R0ARRAY[DI]
02C2 0B856206        R     560                     OR      AX,R1ARRAY[DI]
02C6 0B85F207        R     561                     OR      AX,R2ARRAY[DI]
02CA 7404                  562                     JZ      XYZ_POINT
02CC 47                    563                     INC     DI
02CD 47                    564                     INC     DI
02CE EBE3                  565                     JMP     SHORT TRY_NEXT_PT
02D0 58                    566     XYZ_POINT:      POP     AX
02D1 298D4203        R     567                     SUB     ZARRAY[DI],CX
02D5 299DB201        R     568                     SUB     YARRAY[DI],BX
02D9 29852200        R     569                     SUB     XARRAY[DI],AX
02DD C606BC0900      R     570                     MOV     XYZ_FLAG,0
02E2 5F                    571                     POP     DI
02E3 C3                    572     END_INSERT:     RET
                           573 +1  $EJECT
```

```
LOC  OBJ                  LINE     SOURCE

02E4 A1A009          R      574     ROT_MOVE:       MOV     AX,R0STEPS
02E7 8B1EA209        R      575                     MOV     BX,R1STEPS
02EB 8B0EA409        R      576                     MOV     CX,R2STEPS
02EF 0185D004        R      577                     ADD     R0ARRAY[DI-2],AX
02F3 0197D006        R      578                     ADD     R1ARRAY[DI-2],BX
02F7 018DF007        R      579                     ADD     R2ARRAY[DI-2],CX
02FB 57                     580                     PUSH    DI
02FC 81BD42030080    R      581     TRY_NEXT_PT2:   CMP     ZARRAY[DI],8000H
0302 7502                   582                     JNZ     NEXT_OFFSET2
0304 47                     583                     INC     DI
0305 47                     584                     INC     DI
0306 50                     585     NEXT_OFFSET2:   PUSH    AX
0307 8B852200        R      586                     MOV     AX,XARRAY[DI]
030B 0B85B201        R      587                     OR      AX,YARRAY[DI]
030F 0B854203        R      588                     OR      AX,ZARRAY[DI]
0313 7404                   589                     JZ      ROT_POINT
0315 47                     590                     INC     DI
0316 47                     591                     INC     DI
0317 EBE3                   592                     JMP     SHORT TRY_NEXT_PT2
0319 58                     593     ROT_POINT:      POP     AX
031A 2985D204        R      594                     SUB     R0ARRAY[DI],AX
031E 2997D206        R      595                     SUB     R1ARRAY[DI],BX
0322 298DF207        R      596                     SUB     R2ARRAY[DI],CX
0326 C606BB0900      R      597                     MOV     ROT_FLAG,0
032B 5F                     598                     POP     DI
032C C3                     599                     RET
                            600     MOVE_PT         ENDP
                            601
                            602 +1  $EJECT
```

```
LOC  OBJ                  LINE     SOURCE

                          603
032D                      604     ENTERPT         PROC                    ;
                          605                     ; This routine is used to enter a point in the matrix and also
                          606                     ; clear the xSTEPS variables ready for the next offsets. The
                          607                     ; new point is then checked to see if it is a bogus all zero
                          608                     ; point, if it is then the point index is not incremented and
                          609                     ; the routine is exited. If the point is okay then the point
                          610                     ; counter is incremented and then xyz and rot flags are cleared.
                          611                     ; We also clear the autopt variable as it may have been this that
                          612                     ; forced the point to be entered.
032D 33C0                 613                     XOR     AX,AX           ;AX = 0
032F 87069A09       R     614                     XCHG    AX,XSTEPS       ;MOVE XSTEPS TO AX AND CLEAR XSTEPS
0333 89852200       R     615                     MOV     XARRAY [DI],AX  ;STORE IN MATRIX
0337 33C0                 616                     XOR     AX,AX           ;REPEAT THIS FOR ALL THE AXES
0339 87069C09       R     617                     XCHG    AX,YSTEPS       ;MOVING THE OFFSETS INTO THE
033D 8985B201       R     618                     MOV     YARRAY [DI],AX  ;MATRIX AND CLEARING THE OFFSETS
0341 33C0                 619                     XOR     AX,AX           ;FOR THE NEXT POINT AT THE SAME
0343 87069E09       R     620                     XCHG    AX,ZSTEPS       ;TIME.
0347 89854203       R     621                     MOV     ZARRAY [DI],AX  ;
034B 33C0                 622                     XOR     AX,AX           ;
034D 8706A009       R     623                     XCHG    AX,ROSTEPS      ;
0351 8985D204       R     624                     MOV     ROARRAY [DI],AX ;
0355 33C0                 625                     XOR     AX,AX           ;
0357 8706A209       R     626                     XCHG    AX,R1STEPS      ;
035B 89856206       R     627                     MOV     R1ARRAY [DI],AX ;
035F 33C0                 628                     XOR     AX,AX           ;
0361 8706A409       R     629                     XCHG    AX,R2STEPS      ;
0365 8985F207       R     630                     MOV     R2ARRAY [DI],AX ;
0369 23C0                 631                     AND     AX,AX           ;CHECK THE OFFETS ARE NOT ALL
036B 7523                 632                     JNZ     ENTEREND        ;ZERO AS THIS WOULD BE AN
036D 83BD420300     R     633                     CMP     ZARRAY [DI],0   ;ILLEGAL POINT.
0372 751C                 634                     JNZ     ENTEREND        ;
0374 83BDB20100     R     635                     CMP     YARRAY [DI],0   ;
0379 7515                 636                     JNZ     ENTEREND        ;
037B 83BD220000     R     637                     CMP     XARRAY [DI],0   ;
0380 750E                 638                     JNZ     ENTEREND        ;
0382 83BDD20400     R     639                     CMP     ROARRAY [DI],0  ;
0387 7507                 640                     JNZ     ENTEREND        ;
0389 83BD620600     R     641                     CMP     R1ARRAY [DI],0  ;
038E 7406                 642                     JZ      NOK_END         ;
0390 47                   643     ENTEREND:       INC     DI              ;MOVE THE ARRAY INDEX TO POINT
0391 47                   644                     INC     DI              ;TO THE NEXT LOCATION.
0392 FF06BF09       R     645                     INC     PTCOUNT         ;INCREMENT POINT COUNT
0396 32C0                 646     NOK_END:        XOR     AL,AL           ;AL=0
0398 A2B609         R     647                     MOV     AUTOPT,AL       ;TURN THE AUTOPT FLAG OFF
039B A2BB09         R     648                     MOV     ROT_FLAG,AL     ;AND CLEAR THE AXES FLAGS.
039E A2BC09         R     649                     MOV     XYZ_FLAG,AL
03A1 C3                   650                     RET
                          651     ENTERPT         ENDP
                          652
```

```
LOC  OBJ                  LINE     SOURCE

                           658                    ; if the splines are still on.
0302 803EAF0900      R     659                    CMP     WELDON,0                ;GET THE WELD INDICATOR.
0307 7507                  660                    JNZ     STOPWELD                ;WELDON IS SET THEREFORE GOTO STOP
0309 C606AE0901      R     661                    MOV     CONTROL,1               ;CONTROL = 1 = START WELD
030E EB1B                  662                    JMP     SHORT ENDSSW            ;GO AND ENTER IN MATRIX.
0310 803EB00900      R     663    STOPWELD:       CMP     SPLNON,0                ;GET THE SPLINE INDICATOR.
0315 7408                  664                    JZ      NOSPLN                  ;NO SPLINE IS OFF.
0317 FE06BD09        R     665                    INC     SPLNCNT                 ;SPLINE WAS ON SO TURN IT OFF AND
031B F616B009        R     666                    NOT     SPLNON                  ;INCREMENT THE SPLINE COUNT.
031F FE06BE09        R     667    NOSPLN:         INC     WELDCNT                 ;INCREMENT NUMBER OF WELDS.
0323 C606AE0900      R     668                    MOV     CONTROL,0               ;CONTROL = SPLINE & WELD OFF.
0328 F616AF09        R     669    ENDSSW:         NOT     WELDON                  ;TOGGLE THE WELD FLAG.
032C E85EFF                670                    CALL    ENTERPT                 ;ENTER A POINT.
032F E81901                671                    CALL    CONTROLPUT              ;ENTER THE CONTROL TO INDICATE
0332 C3                    672                    RET                             ;THE CHANGE AT THIS POINT.
                           673    WELDPT          ENDP
                           674
                           675 +1 $EJECT
```

```
LOC  OBJ                  LINE     SOURCE

                           676
03D3                       677     SPLINEPT        PROC                            ;
                           678                     ; This routine toggles the spline situation when the spline
                           679                     ; key is pressed. The splines may only be turned on during
                           680                     ; a weld.
03D3 803EB00900      R     681                     CMP     SPLNON,0                ;GET THE SPLINE INDICATOR.
03D8 7509                  682                     JNZ     SPLNOFF                 ;SPLINE WANTS TURNING OFF
03DA 803EAF0900      R     683                     CMP     WELDON,0                ;GET WELD INDICATOR.
03DF 7415                  684                     JZ      NOWELD                  ;NO ..... THEREFORE NO SPLINES
03E1 EB09                  685                     JMP     SHORT ENDSSSP           ;YES .... TAKE ACTION.
03E3 C606AE0902      R     686     SPLNOFF:        MOV     CONTROL,2               ;SET CONTROL TO INDICATE OFF
03E8 FE06B009        R     687                     INC     SPLNCNT                 ;INCREMENT THE COUNT SPLINE.
03EC E83EFF                687     ENDSSSP:        CALL    ENTERPT                 ;PLACE POINT IN THE MATRIX.
03EF F616B009        R     689                     NOT     SPLNON                  ;TOGGLE THE SPLINE INDICATOR.
03F3 E8F500                690                     CALL    CONTROLPUT              ;ENTER THE CONTROL POINT.
03F6 C3                    691     NOWELD:         RET                             ;
                           692     SPLINEPT        ENDP
                           693
                           694 +1  $EJECT
```

```
LOC  OBJ                  LINE     SOURCE

                          695
03F7                      696      SLOW            PROC                            ;REDUCES MOTOR SPEEDS
03F7 23F6                 697                      AND     SI,SI                   ;IS SPEED ALREADY A MIN
03F9 7407                 698                      JZ      SLOWRET                 ;YES.......
03FB 4E                   699                      DEC     SI                      ;REDUCE SPEED POINTER
03FC 4E                   700                      DEC     SI                      ;
03FD C606B90900      R    701                      MOV     MOTLOAD,0
0402 C3                   702      SLOWRET:        RET
                          703      SLOW            ENDP
                          704
                          705
0403                      706      FAST            PROC                            ;INCREASES MOTOR SPEEDS
0403 83FE0E               707                      CMP     SI,SIZE SPEEDS-2        ;IS SPEED ALREADY A MAX
0406 7407                 708                      JZ      FASTRET                 ;YES.......
0408 46                   709                      INC     SI                      ;INCREASE SPEED POINTER
0409 46                   710                      INC     SI                      ;
040A C606B90900      R    711                      MOV     MOTLOAD,0
040F C3                   712      FASTRET:        RET
                          713      FAST            ENDP
                          714
                          715 +1   $EJECT
```

```
LOC  OBJ                  LINE     SOURCE

                           716
0410                       717     STCHECK         PROC                            ;
                           718                     ; This routine checks the real stack top against the
                           719                     ; minimum return distance. This is for when the exit
                           720                     ; button has been pressed we must check we are within
                           721                     ; a safe range to automatically return to the origin.
                           722                     ; The real stack top is compared with MINRD and then
                           723                     ; the 8086 flags are set to reflect the comparison
                           724                     ; result, so that on return the a decision can be made.
0410 9BDDD7                725                     FST     ST(7)                   ;SAVE ST (xTOTAL) IN ST(7)
0413 9BD9E1                726                     FABS                            ;MAKE x OFFSET TOTAL +VE
0416 9BDE16D309      R     727                     FICOM   MINRD                   ;COMPARE IT WITH THE MINIMUM
041B 9BDD3EAC09      R     728                     FSTSW   REALSTW                 ;RETURN DISTANCE & STORE REAL STATUS
0420 9BD9CF                729                     FXCH    ST(7)                   ;RESTORE x OFFSET & STORE FINISH.
0423 9BDDC7                730                     FFREE   ST(7)                   ;CLEAR ST(7)
0426 8A26AD09        R     731                     MOV     AH,BYTE PTR REALSTW+1   ;LOAD TOP BYTE OF 8087 STATUS
042A 9E                    732                     SAHF                            ;INTO THE 8086 FLAGS.
042B C3                    733                     RET                             ;FLAGS NOW SET (RET DOES NOT ALTER
                           734     STCHECK         ENDP                            ;THEM BUT Jcon MUST AFTER CALL ).
                           735
                           736 +1  $EJECT
```

```
LOC  OBJ                  LINE     SOURCE

                           737
042C                       738     ENDKEY          PROC                            ;EXIT KEY ROUTINE
                           739                     ; If we are in the move routine the exit will always return
                           740                     ; to the PL/M wherever the torch is. Otherwise we must be in
                           741                     ; the input routine, so first we check that we are not welding
                           742                     ; then STCHECK is used repeatedly to make sure that the x,y &z
                           743                     ; axes are all within the minimum return distance. If we are
                           744                     ; then the present offsets are entered in the matrix and the
                           745                     ; total offsets are removed from the real stack. These are then
                           746                     ; inverted and also placed in the matrix (two seperate points
                           747                     ; for rotation and xyz ) along with a control point to mark the
                           748                     ; end of path. The AUTOFLG is then set and PINPUT is changed to
                           749                     ; make the software think it is in the trace mode and the path
                           750                     ; index is set so that the last points are traced and the torch
                           751                     ; returns to the origin.
042C A0D909          R     752                     MOV     AL,PINPUT               ;GET ROUTINE MASK
042F A8DC                  753                     TEST    AL,NOT MOVE_MASK        ;ARE WE IN MOVE ROUTINE
0431 7503                  754                     JNZ     NOTMOVE                 ;EXIT ANY TIME AS IN MOVE ROUTINE.
0433 E9A800                755                     JMP     MOVEEXIT                ;*** MOVE EXIT ***
0436 803EAE0900      R     756     NOTMOVE:        CMP     CONTROL,0               ;CHECK WELDING IS OFF ?
043B 7505                  757                     JNZ     BAD_BNC                 ;NO.....STILL WELDING
043D E8D0FF                758                     CALL    STCHECK                 ;CHECK XTOTAL.
0440 7603                  759                     JBE     OFF_OK                  ;OFFSET TO BIG FOR US TO HANDLE.
0442 E9A300                760     BAD_BNC:        JMP     BADEND                  ;LONG ERROR JUMP.
0445 9BD9C9                761     OFF_OK:         FXCH                            ;NOW DO THE SAME TO YTOTAL.
0448 E8C5FF                762                     CALL    STCHECK                 ;CHECK YTOTAL.
044B 9BD9C9                763                     FXCH                            ;REAL STACK AS IT WAS.
044E 77F2                  764                     JA      BAD_BNC                 ;OFFSET TO BIG FOR US TO HANDLE.
0450 9BD9CA                765                     FXCH    ST(2)                   ;NOW DO THE SAME TO ZTOTAL.
0453 E8BAFF                766                     CALL    STCHECK                 ;CHECK ZTOTAL.
0456 9BD9CA                767                     FXCH    ST(2)                   ;REAL STACK AS IT WAS.
0459 77E7                  768                     JA      BAD_BNC                 ;OFFSET TO BIG FOR US TO HANDLE.
045B 9BDF1E8209      R     769                     FISTP   XTOTAL                  ;MUST BE <2000 THEREFORE DW
0460 9BDF1E8409      R     770                     FISTP   YTOTAL                  ;STORRAGE IS LARGE ENOUGH.
0465 9BDF1E8609      R     771                     FISTP   ZTOTAL                  ;SO GET ALL THE TOTALS
046A 9BDF1E8B09      R     772                     FISTP   ROTOTAL                 ;
046F 9BDF1E8A09      R     773                     FISTP   R1TOTAL                 ;
0474 9BDF1E8C09      R     774                     FISTP   R2TOTAL                 ;
0479 9B                    775                     FWAIT                           ;WAIT FOR 8087 TO STORE
047A E8B0FE                776                     CALL    ENTERPT                 ;ENTER POINT AT WHICH END PRESSED
                           777 +1  $EJECT
```

```
LOC  OBJ                  LINE     SOURCE

047D A18209          R     778                MOV     AX,XTOTAL               ;STORE FINAL POINT TO BE DONE
0480 F7D8                  779                NEG     AX                      ;IN THE MATRIX BEFORE WE DO IT.
0482 A39A09          R     780                MOV     XSTEPS,AX               ;NOT ALL TOTAL OFFSETS MUST
0485 A18409          R     781                MOV     AX,YTOTAL               ;BE NEGATED.
0488 F7D8                  782                NEG     AX                      ;
048A A39C09          R     783                MOV     YSTEPS,AX               ;
048D A18609          R     784                MOV     AX,ZTOTAL               ;
0490 F7D8                  785                NEG     AX                      ;
0492 A39E09          R     786                MOV     ZSTEPS,AX               ;
0495 33C0                  787                XOR     AX,AX                   ;
0497 A3A009          R     788                MOV     ROSTEPS,AX              ;CLEAR THE ROTATION OFFSETS
049A A3A209          R     789                MOV     R1STEPS,AX              ;
049D A3A409          R     790                MOV     R2STEPS,AX              ;
04A0 57                    791                PUSH    DI                      ;SAVE THE MATRIX INDEX
04A1 E889FE                792                CALL    ENTERPT                 ;STORE THE XYZ OFFSETS
04A4 33C0                  793                XOR     AX,AX                   ;CLEAR THE XYZ OFFSETS
04A6 A39E09          R     794                MOV     ZSTEPS,AX               ;
04A9 A39C09          R     795                MOV     YSTEPS,AX               ;
04AC A39A09          R     796                MOV     XSTEPS,AX               ;
04AF A18809          R     797                MOV     AX,ROTOTAL              ;LOAD ALL THE ROTATIONS
04B2 F7D8                  798                NEG     AX                      ;THEY MUST BE NEGATED THE
04B4 A3A009          R     799                MOV     ROSTEPS,AX              ;SAME AS THE XYZ OFFSETS
04B7 A18A09          R     800                MOV     AX,R1TOTAL              ;
04BA F7D8                  801                NEG     AX                      ;
04BC A3A209          R     802                MOV     R1STEPS,AX              ;
04BF A18C09          R     803                MOV     AX,R2TOTAL              ;
04C2 F7D8                  804                NEG     AX                      ;
04C4 A3A409          R     805                MOV     R2STEPS,AX              ;
04C7 E863FE                806                CALL    ENTERPT                 ;ENTER THE ROTATION POINT.
04CA C606AE0904      R     807                MOV     CONTROL,4               ;PLACE THE END OF PATH
04CF E81900                808                CALL    CONTROLPUT              ;MARKER IN THE MATRIX.
04D2 5F                    809                POP     DI                      ;RESTORE THE PATH INDEX.
04D3 FE06B309        R     810                INC     AUTOFLG                 ;TURN ON AUTO FLAG.
04D7 C606D909FF      R     811                MOV     PINPUT,0FFH             ;
04DC EB0A                  812                JMP     SHORT BADEND            ;END NOT REACHED YET.
04DE 33C0                  813     MOVEEXIT:  XOR     AX,AX                   ;TURN OFF ALL THE MOTORS BEFORE
04E0 BA0804                814                MOV     DX,ENBPRTADDR           ;THE MOVE EXIT.
04E3 EE                    815                OUT     DX,AL                   ;MOTORS STOPPED.
04E4 FE06B109        R     816                INC     FINISH                  ;TURN ON FINISH FLAG.
04E8 C3                    817     BADEND:    RET
                           818     ENDKEY     ENDP
                           819
                           820 +1  $EJECT
```

```
LOC  OBJ                  LINE     SOURCE

                           821
04E9                       822     FALSEX          PROC
04E9 CC                    823                     INT 3
04EA C3                    824                     RET
                           825     FALSEX          ENDP
                           826
                           827
04EB                       828     CONTROLPUT      PROC                            ;ENTER CONTROL POINT ROUTINE.
                           829                     ; This routine takes the byte stored in the control variable
                           830                     ; and places it in the X leg of the matrix. The rest of the
                           831                     ; offsets are then set to 8000H which is an impossible number.
                           832                     ; This way other routines can indentify it as a control point
                           833                     ; and not a normal offset point.
04EB A0AE09          R     834                     MOV     AL,CONTROL              ;GET CONTROL BYTE
04EE 98                    835                     CBW                             ;CONVERT TO A WORD
04EF 89852200        R     836                     MOV     XARRAY [DI],AX          ;STORE CONTROL IN  THE MATRIX
04F3 C785B2010080    R     837                     MOV     YARRAY [DI],8000H       ;THEN STORE 8000H IN ALL THE
04F9 C78542030080    R     838                     MOV     ZARRAY [DI],8000H       ;OTHER LEGS OF THE MATRIX SO
04FF C785D2040080    R     839                     MOV     R0ARRAY[DI],8000H       ;IT CAN BE IDENTIFIED AS A
0505 C78562060080    R     840                     MOV     R1ARRAY[DI],8000H       ;CONTROL POINT.
050B C785F2070080    R     841                     MOV     R2ARRAY[DI],8000H       ;
0511 FF06BF09        R     842                     INC     PTCOUNT                 ;INCREMENT THE NUMBER OF
0515 47                    843                     INC     DI                      ;MATRIX  ENTRIES  AND
0516 47                    844                     INC     DI                      ;MOVE TO NEXT MATRIX POINT.
0517 C3                    845                     RET
                           846     CONTROLPUT      ENDP
                           847
                           848 +1  $EJECT
```

```
LOC  OBJ                  LINE     SOURCE

                          849
0518                      850      SETSIGN         PROC                            ;SETS UP THE SIGN FLAGS.
                          851                      ; This routine takes all the offsets and sets the corresponding
                          852                      ; sign bit in the sign flags for output to the motor drivers.
                          853                      ; The offsets are all made positive for the maths routines so
                          854                      ; we do not obtain negative speeds from the maths.
0518 C606B40900      R    855                      MOV     SIGNF,0                 ;CLEAR ALL SIGN FLAGS.
051D A18209          R    856                      MOV     AX,XTOTAL               ;GET XTOTAL
0520 23C0                 857                      AND     AX,AX                   ;SET FLAGS
0522 790A                 858                      JNS     YSIGN                   ;JUMP IF POSITIVE
0524 800EB40901      R    859                      OR      SIGNF,1                 ;SET THE SIGN FLAG
0529 F7D8                 860                      NEG     AX                      ;ABS(XTOTAL)
052B A38209          R    861                      MOV     XTOTAL,AX               ;STORE XTOTAL
052E A18409          R    862      YSIGN:          MOV     AX,YTOTAL               ;GET YTOTAL
0531 23C0                 863                      AND     AX,AX                   ;SET FLAGS
0533 790A                 864                      JNS     ZSIGN                   ;JUMP IF POSITIVE
0535 800EB40902      R    865                      OR      SIGNF,2                 ;SET THE SIGN FLAG
053A F7D8                 866                      NEG     AX                      ;ABS(YTOTAL)
053C A38409          R    867                      MOV     YTOTAL,AX               ;STORE YTOTAL
053F A18609          R    868      ZSIGN:          MOV     AX,ZTOTAL               ;GET ZTOTAL
0542 23C0                 869                      AND     AX,AX                   ;SET FLAGS
0544 790A                 870                      JNS     ROSIGN                  ;JUMP IF POSITIVE
0546 800EB40904      R    871                      OR      SIGNF,4                 ;SET THE SIGN FLAG
054B F7D8                 872                      NEG     AX                      ;ABS(ZTOTAL)
054D A38609          R    873                      MOV     ZTOTAL,AX               ;STORE ZTOTAL
                          874 +1   $EJECT
```

```
LOC  OBJ                  LINE     SOURCE

0550 A18809         R      875     ROSIGN:         MOV     AX,ROTOTAL              ;GET YTOTAL
0553 23C0                  876                     AND     AX,AX                   ;SET FLAGS
0555 790A                  877                     JNS     R1SIGN
0557 800EB40908     R      878                     OR      SIGNF,8H                ;SET THE SIGN FLAG
055C F7D8                  879                     NEG     AX                      ;ABS(YTOTAL)
055E A38809         R      880                     MOV     ROTOTAL,AX              ;STORE YTOTAL
0561 A18A09         R      881     R1SIGN:         MOV     AX,R1TOTAL              ;GET YTOTAL
0564 23C0                  882                     AND     AX,AX                   ;SET FLAGS
0566 790A                  883                     JNS     R2SIGN                  ;JUMP IF POSITIVE
0568 800EB40910     R      884                     OR      SIGNF,10H               ;SET THE SIGN FLAG
056D F7D8                  885                     NEG     AX                      ;ABS(YTOTAL)
056F A38A09         R      886                     MOV     R1TOTAL,AX              ;STORE YTOTAL
0572 A18C09         R      887     R2SIGN:         MOV     AX,R2TOTAL              ;GET YTOTAL
0575 23C0                  888                     AND     AX,AX                   ;SET FLAGS
0577 790A                  889                     JNS     SSEXIT                  ;JUMP IF POSITIVE
0579 800EB40920     R      890                     OR      SIGNF,20H               ;SET THE SIGN FLAG
057E F7D8                  891                     NEG     AX                      ;ABS(YTOTAL)
0580 A38C09         R      892                     MOV     R2TOTAL,AX              ;STORE YTOTAL
0583 C3                    893     SSEXIT:         RET                             ;SIGNF IS NOW VALID
                           894     SETSIGN         ENDP
                           895
                           896 +1  $EJECT
```

```
LOC  OBJ                  LINE     SOURCE

                           897
0584                       898     SETENABLE       PROC                    ;SETS THE ENABLE FLAGS.
                           899                     ; This routine sets the corresponding enable bits in the
                           900                     ; enable flags which when output are the clock gates. All the
                           901                     ; routine does is to check if the offsets are none zero and if
                           902                     ; so the correct enable bit is set.
0584 32C9                  903                     XOR     CL,CL           ;CLEAR CL FOR ENABLES
0586 F7068C09FFFF    R     904                     TEST    R2TOTAL,0FFFFH  ;IS DISTANCE ZERO ?
058C 7401                  905                     JZ      SEL1            ;NO.....
058E F9                    906                     STC                     ;YES..... SET CARRY
058F D0D1                  907     SEL1:           RCL     CL,1            ;CL=XXXXXXXE
0591 F7068A09FFFF    R     908                     TEST    R1TOTAL,0FFFFH  ;IS DISTANCE ZERO ?
0597 7401                  909                     JZ      SEL2            ;NO.....
0599 F9                    910                     STC                     ;YES..... SET CARRY
059A D0D1                  911     SEL2:           RCL     CL,1            ;CL=XXXXXXEE
059C F7068809FFFF    R     912                     TEST    R0TOTAL,0FFFFH  ;IS DISTANCE ZERO ?
05A2 7401                  913                     JZ      SEL3            ;NO.....
05A4 F9                    914                     STC                     ;YES..... FLAG IT
05A5 D0D1                  915     SEL3:           RCL     CL,1            ;CL=XXXXXEEE
05A7 F7068609FFFF    R     916                     TEST    ZTOTAL,0FFFFH   ;IS DISTANCE ZERO ?
05AD 7401                  917                     JZ      SEL4            ;AND SO ON FOR THE OTHER
05AF F9                    918                     STC                     ;OFFSET IN THE TOTALS.
05B0 D0D1                  919     SEL4:           RCL     CL,1            ;
05B2 F7068409FFFF    R     920                     TEST    YTOTAL,0FFFFH   ;IS DISTANCE ZERO ?
05B8 7401                  921                     JZ      SEL5            ;
05BA F9                    922                     STC                     ;
05BB D0D1                  923     SEL5:           RCL     CL,1            ;
05BD F7068209FFFF    R     924                     TEST    XTOTAL,0FFFFH   ;IS DISTANCE ZERO ?
05C3 7401                  925                     JZ      SEL6            ;
05C5 F9                    926                     STC                     ;
05C6 D0D1                  927     SEL6:           RCL     CL,1            ;
05C8 880EB509        R     928                     MOV     ENABLEF,CL      ;SAVE ENABLE FLAGS
05CC C3                    929                     RET                     ;ENABLEF IS NOW VALID
                           930     SETENABLE       ENDP
                           931
                           932 +1  $EJECT
```

```
LOC  OBJ                  LINE     SOURCE

                          933
                          934      ;*****************************************************************************
                          935      ;*                                                                           *
                          936      ;*              INTERRUPT ENTRANCE AT CS:TRCINT LABEL FAR                    *
                          937      ;*                                                                           *
                          938      ;*****************************************************************************
                          939
05CD                      940      TRCINT          LABEL   FAR                 ;INTERRUPT ENTRANCE FOR THE
                          941                      PUBLIC  TRCINT              ;TRACE AND EDIT ROUTINES.
05CD B8B75F               942                      MOV     AX,TIME10mS         ;RELOAD 10mS TIME INTERVAL
05D0 E6D2                 943                      OUT     TMRINPUT,AL         ;LSB
05D2 8AC4                 944                      MOV     AL,AH               ;
05D4 E6D2                 945                      OUT     TMRINPUT,AL         ;MSB
05D6 F606B80901      R    946                      TEST    INS,1               ;ARE WE IN INSERT MODE FOR
05DB 7403                 947                      JZ      NOT_INS             ;MOVING POINTS IN EDIT ?
05DD E926FB               948                      JMP     SHORT NAUTOMOV      ;YES......
05E0 F606BA0901      R    949      NOT_INS:        TEST    ROT_ON,1            ;IS THE ROTATION MOVING ?
05E5 740D                 950                      JZ      NOT_ROTATE          ;NO.......
05E7 E83F00               951                      CALL    ROTATION            ;UPDATE THE ROTATION MATHS.
05EA E8AC01               952                      CALL    START_MOTORS        ;RESET THE ENABLES.
05ED C606B70900      R    953                      MOV     MOVSTAT,0           ;
05F2 EB2B                 954                      JMP     SHORT RET_BOUNCE    ;FINISHED.
05F4 F606B70901      R    955      NOT_ROTATE:     TEST    MOVSTAT,1           ;IS THE MOVING STATUS SET
05F9 7405                 956                      JZ      NOT_MOVE            ;FOR THE XYZ AXES ?
05FB E8EC01               957                      CALL    TRACE_MOVE          ;YES......
05FE EB1F                 958                      JMP     SHORT RET_BOUNCE    ;FINISHED.
0600 803ED909FF      R    959      NOT_MOVE:       CMP     PINPUT,0FFH         ;ARE WE IN THE TRACE ROUTINE ?
0605 7412                 960                      JE      START_MOVE          ;YES......
0607 E8F6F9               961                      CALL    GET_KEY             ;SEE IF ANY EDIT KEYS ARE
060A 8ADA                 962                      MOV     BL,DL               ;PRESSED ?
060C 80E350               963                      AND     BL,01010000B        ;ONLY STEP AND MOVE KEYS.
060F 740E                 964                      JZ      RET_BOUNCE          ;NO KEYS PRESSED......
0611 F6C340               965      NOT_DEL:        TEST    BL,01000000B        ;IS THE MOVE KEY PRESSED ?
0614 7403                 966                      JZ      START_MOVE          ;NO....... MUST BE STEP.
0616 E99401               967                      JMP     INSERT_PT           ;YES......, MOVE THE POINT.
0619 E89100               968      START_MOVE:     CALL    LINEMATH            ;DO THE MATHS.
061C E87A01               969                      CALL    START_MOTORS        ;START MOVING.
061F E9B6FB               970      RET_BOUNCE:     JMP     INTER_RET           ;FINISHED.
                          971
                          972 +1   $EJECT
```

```
LOC  OBJ                  LINE     SOURCE

                           973
0622                       974     ROT_MATHS       LABEL   NEAR                    ;INITIAL ROTATION ENTERANCE.
0622 FE06RA09        R     975                     INC     ROT_ON                  ;SET ROTATION ON FLAG.
0626 E8EFFE                976                     CALL    SETSIGN                 ;SET THE SIGNS.
0629                       977     ROTATION        PROC
                           978                     ; The rotation routine is called every time any rotation steps
                           979                     ; are done. It checks whether eight steps may be taken (maximum
                           980                     ; speed) if not then the remainder is done by looking up the
                           981                     ; speed in the speed matrix. The routine knows it has finished
                           982                     ; when the enable flags are down.
0629 E858FF                983                     CALL    SETENABLE               ;SET THE ENABLE FLAGS
062C 56                    984                     PUSH    SI                      ;SAVE SI
062D BE0800                985                     MOV     SI,8                    ;ASSUME EIGHT STEPS
0630 833E880908      R     986                     CMP     ROTOTAL,8               ;CAN WE DO EIGHT STEPS
0635 7704                  987                     JA      ROMAX                   ;YES......
0637 8B368809        R     988                     MOV     SI,ROTOTAL              ;NO..... GET REMAINDER.
063B 29368809        R     989     ROMAX:          SUB     ROTOTAL,SI              ;SUBTRACT STEPS BEING DONE.
063F D1E6                  990                     SHL     SI,1                    ;ADJUST THIS AS AN INDEX.
0641 8B840000        R     991                     MOV     AX,SPEEDS2[SI]          ;GET THE SPEED FROM THE SPEED
0645 BA1004                992                     MOV     DX,TIMER1               ;MATRIX AND LOAD IT TO THE
0648 EE                    993                     OUT     DX,AL                   ;CORRECT CLOCK.
0649 86C4                  994                     XCHG    AL,AH                   ;
064B EE                    995                     OUT     DX,AL                   ;
064C BE0800                996                     MOV     SI,8                    ;REPEAT THIS FOR THE OTHER
064F 833E8A0908      R     997                     CMP     R1TOTAL,8               ;TWO ROTATION AXES.
0654 7704                  998                     JA      R1MAX                   ;
0656 8B368A09        R     999                     MOV     SI,R1TOTAL              ;
                          1000 +1  $EJECT
```

```
LOC  OBJ                  LINE     SOURCE

065A 29368A09        R    1001     R1MAX:          SUB     R1TOTAL,SI              ;
065E D1E6                 1002                      SHL     SI,1                    ;
0660 8B840000        R    1003                      MOV     AX,SPEEDS2[SI]          ;
0664 BA1204               1004                      MOV     DX,TIMER2               ;
0667 EE                   1005                      OUT     DX,AL                   ;
0668 86C4                 1006                      XCHG    AL,AH                   ;
066A EE                   1007                      OUT     DX,AL                   ;
066B BE0800               1008                      MOV     SI,8                    ;
066E 833E8C0908      R    1009                      CMP     R2TOTAL,8               ;
0673 7704                 1010                      JA      R2MAX                   ;
0675 8B368C09        R    1011                      MOV     SI,R2TOTAL              ;
0679 29368C09        R    1012     R2MAX:          SUB     R2TOTAL,SI              ;
067D D1E6                 1013                      SHL     SI,1                    ;
067F 8B840000        R    1014                      MOV     AX,SPEEDS2[SI]          ;
0683 BA1404               1015                      MOV     DX,TIMER3               ;
0686 EE                   1016                      OUT     DX,AL                   ;
0687 86C4                 1017                      XCHG    AL,AH                   ;
0689 EE                   1018                      OUT     DX,AL                   ;
068A 5E                   1019                      POP     SI                      ;RESTORE THE SI REGISTER.
068B 803EB50900      R    1020                      CMP     ENABLEF,0               ;ROTATION FINISHED ?
0690 751A                 1021                      JNZ     MORE_ROTS               ;NO........
0692 C606BA0900      R    1022                      MOV     ROT_ON,0                ;YES....... CLEAR THE FLAG
0697 47                   1023                      INC     DI                      ;MOVE TO THE NEXT POINT.
0698 47                   1024                      INC     DI                      ;
0699 83BD220004      R    1025                      CMP     XARRAY[DI],4            ;IS IT AN END OF PATH POINT ?
069E 750C                 1026                      JNZ     MORE_ROTS               ;NO........
06A0 81BD42030080    R    1027                      CMP     ZARRAY[DI],8000H        ;WAS THIS A CONTROL ?
06A6 7504                 1028                      JNZ     MORE_ROTS               ;NO........
06A8 FE06B109        R    1029                      INC     FINISH                  ;YES....... TRACE FINISHED.
06AC C3                   1030     MORE_ROTS:      RET
                          1031     ROTATION        ENDP
                          1032
                          1033 +1  $EJECT
```

```
LOC  OBJ                  LINE     SOURCE

                          1034
06AD                      1035     LINEMATH        PROC
                          1036                     ;This procedure is responsible for mainly the xyz maths in
                          1037                     ;the trace and edit routines. It is called to set up the
                          1038                     ;speeds in the xyz counters. First though it must pass over
                          1039                     ;any control points which are ignored. Then it must decide
                          1040                     ;if it is an xyz point or a rotation point by seeing if the
                          1041                     ;xyz offsets are all zero, if they are it passes over to the
                          1042                     ;rotation maths. If it is an xyz point it computes the speeds
                          1043                     ;and loads the correct counters and then moves the path index
                          1044                     ;on to the next point.
06AD 8B854203       R     1045                     MOV     AX,ZARRAY [DI]          ;GET NEXT Z POINT
06B1 3D0080               1046                     CMP     AX,8000H                ;IS IT A CONTROL?
06B4 7504                 1047                     JNZ     REST_PTS                ;NO......
06B6 47                   1048                     INC     DI                      ;YES..... MOVE TO NEXT POINT
06B7 47                   1049                     INC     DI                      ;
06B8 EBF3                 1050                     JMP     SHORT LINEMATH          ;GET NEXT POINT
06BA 8B85D204       R     1051     REST_PTS:       MOV     AX,R0ARRAY[DI]          ;LOAD ROTATION TOTALS
06BE A38809         R     1052                     MOV     R0TOTAL,AX              ;
06C1 8B856206       R     1053                     MOV     AX,R1ARRAY[DI]          ;
06C5 A38A09         R     1054                     MOV     R1TOTAL,AX              ;
06C8 8B85F207       R     1055                     MOV     AX,R2ARRAY[DI]          ;
06CC A38C09         R     1056                     MOV     R2TOTAL,AX              ;
06CF 33C0                 1057                     XOR     AX,AX                   ;CLEAR THE XYZ TOTALS INCASE
06D1 A38209         R     1058                     MOV     XTOTAL,AX               ;IT IS A ROTATION POINT.
06D4 A38409         R     1059                     MOV     YTOTAL,AX               ;
06D7 A38609         R     1060                     MOV     ZTOTAL,AX               ;
06DA 0B852200       R     1061                     OR      AX,XARRAY[DI]           ;ARE ALL THE XYZ OFFSETS
06DE 0B85B201       R     1062                     OR      AX,YARRAY[DI]           ;ZERO ?
06E2 0B854203       R     1063                     OR      AX,ZARRAY[DI]           ;
06E6 85C0                 1064                     TEST    AX,AX                   ;
06E8 7503                 1065                     JNZ     XYZ_MATHS               ;NO...... MUST BE XYZ POINT
06EA E935FF               1066                     JMP     SHORT ROT_MATHS         ;ELSE ITS A ROTATION POINT.
06ED 8B854203       R     1067     XYZ_MATHS:      MOV     AX,ZARRAY[DI]           ;
06F1 A38609         R     1068                     MOV     ZTOTAL,AX               ;LOAD UP ALL THE xTOTALS
06F4 9BDF068609     R     1069                     FILD    ZTOTAL                  ;FOR THE SET SIGN ROUTINE
06F9 8B85B201       R     1070                     MOV     AX,YARRAY [DI]          ;AND LOAD THE NPX
06FD 9BDCC8               1071                     FMUL    ST,ST                   ;Z*Z
0700 A38409         R     1072                     MOV     YTOTAL,AX               ;REPEAT FOR X & Y
0703 9BDF068409     R     1073                     FILD    YTOTAL                  ;
0708 8B852200       R     1074                     MOV     AX,XARRAY [DI]          ;
070C 9BDCC8               1075                     FMUL    ST,ST                   ;Y*Y
070F A38209         R     1076                     MOV     XTOTAL,AX               ;
0712 9BDF068209     R     1077                     FILD    XTOTAL                  ;
0717 9BDCC8               1078                     FMUL    ST,ST                   ;X*X
071A 9BDEC1               1079                     FADD                            ;ST=(X*X+Y*Y)
071D 9BDEC1               1080                     FADD                            ;ST=(X*X+Y*Y+Z*Z)
0720 9BD9FA               1081                     FSQRT                           ;ST=LENGTH OF STRAIGHT LINE.
                          1082 +1  $EJECT
```

```
LOC  OBJ                  LINE     SOURCE

0723 9BDE36D709      R    1083                FIDIV   TRACSPD                 ;DISTANCE/VELOCITY=TIME
0728 E8EDFD               1084                CALL    SETSIGN                 ;SET SIGN FLAGS
072B 9BDDD1               1085                FST     ST(1)                   ;ST=ST(1)=TIME
072E 9BDF1EA809      R    1086                FISTP   TIMCNT                  ;SAVE NUMBER OF 10mS PULSES
0733 9BDE0ED509      R    1087                FIMUL   FREQ                    ;MULTIPLY BY CLOCK FREQUENCY
0738 E849FE               1088                CALL    SETENABLE               ;SET ENABLE BITS.
073B 9BDDD1               1089                FST     ST(1)                   ;ST(1)=ST=TIME
073E 833E820900      R    1090                CMP     XTOTAL,0                ;IS THE XTOTAL ZERO?
0743 7405                 1091                JZ      XZERO                   ;NO.....
0745 9BDE368209      R    1092                FIDIV   XTOTAL                  ;RESULT=XCOUNT+1
074A 9BDF1E8E09      R    1093    XZERO:      FISTP   XSPEED                  ;STORE X COUNT
074F 9BDDD1               1094                FST     ST(1)                   ;ST(1)=ST
0752 833E840900      R    1095                CMP     YTOTAL,0                ;IS YTOTAL ZERO?
0757 7405                 1096                JZ      YZERO                   ;NO....
0759 9BDE368409      R    1097                FIDIV   YTOTAL                  ;RESULT=YCOUNT+1
075E 9BDF1E9009      R    1098    YZERO:      FISTP   YSPEED                  ;STORE Y COUNT
0763 833E860900      R    1099                CMP     ZTOTAL,0                ;IS THE ZTOTAL ZERO?
0768 7405                 1100                JZ      ZZERO                   ;NO....
076A 9BDE368609      R    1101                FIDIV   ZTOTAL                  ;RESULT=ZCOUNT+1
076F 9BDF1E9209      R    1102    ZZERO:      FISTP   ZSPEED                  ;STORE Z COUNT
0774 A18E09          R    1103                MOV     AX,XSPEED               ;GET X SPEED (XCOUNT+1)
0777 48                   1104                DEC     AX                      ;WAS XCOUNT+1 NOW XCOUNT
0778 BA1804               1105                MOV     DX,TIMER4               ;LOAD XTIMER ADDRESS AND
077B EE                   1106                OUT     DX,AL                   ;LOAD LOW BYTE TO COUNTER
077C 86C4                 1107                XCHG    AL,AH                   ;SWOP LSB & MSB
077E EE                   1108                OUT     DX,AL                   ;OUT MSB TO COUNTER
077F A19009          R    1109                MOV     AX,YSPEED               ;GET Y SPEED (YCOUNT)
0782 48                   1110                DEC     AX                      ;WAS YCOUNT+1 NOW YCOUNT
0783 BA1A04               1111                MOV     DX,TIMER5               ;LOAD YTIMER ADDRESS AND
0786 EE                   1112                OUT     DX,AL                   ;LOAD LOW BYTE TO COUNTER
0787 86C4                 1113                XCHG    AL,AH                   ;SWOP LSB & MSB
0789 EE                   1114                OUT     DX,AL                   ;OUT MSB TO COUNTER
078A 9B                   1115                FWAIT                           ;WAIT FOR LAST 8087 INSTRUCTION
078B A19209          R    1116                MOV     AX,ZSPEED               ;GET Z SPEED (ZCOUNT)
078E 48                   1117                DEC     AX                      ;WAS ZCOUNT+1 NOW ZCOUNT
078F BA1C04               1118                MOV     DX,TIMER6               ;LOAD ZTIMER ADDRESS AND
0792 EE                   1119                OUT     DX,AL                   ;LOAD LOW BYTE TO COUNTER
0793 86C4                 1120                XCHG    AL,AH                   ;SWOP LSB & MSB
0795 EE                   1121                OUT     DX,AL                   ;OUT MSB TO COUNTER
0796 47                   1122                INC     DI                      ;MOVE TO NEXT POINT
0797 47                   1123                INC     DI                      ;IN THE MATRIX.
0798 C3                   1124                RET
                          1125    LINEMATH    ENDP
                          1126
                          1127 +1 $EJECT
```

```
LOC  OBJ                  LINE     SOURCE

                          1128
0799                      1129     START_MOTORS    PROC
                          1130                     ; This routine starts the motors moving by outputing the
                          1131                     ; direction bits to the drivers and then the enables to the
                          1132                     ; clock gates.
0799 C606B70901      R    1133                     MOV     MOVSTAT,1               ;FLAG MOVING IS IN PROGRESS
079E A0B409          R    1134                     MOV     AL,SIGNF                ;GET THE SIGNS AND THE
07A1 BA0004               1135                     MOV     DX,DIRPRTADDR           ;DIRECTION PORT ADDRESS.
07A4 EE                   1136                     OUT     DX,AL                   ;OUTPUT THE SIGNS.
07A5 A0B509          R    1137                     MOV     AL,ENABLEF              ;GET ENABLE FLAGS
07A8 BA0804               1138                     MOV     DX,ENBPRTADDR           ;GET MOTOR ADDRESSES.
07AB EE                   1139                     OUT     DX,AL                   ;START MOTORS.
07AC C3                   1140                     RET
                          1141     START_MOTORS    ENDP
                          1142
                          1143 +1  $EJECT
```

```
LOC  OBJ                  LINE     SOURCE

                          1144
07AD                      1145     INSERT_PT       LABEL   NEAR
                          1146                     ; This routine does not insert a point but is part of the move
                          1147                     ; point process. It clears the step variables and checks to see
                          1148                     ; if a xyz or rotation point is being moved and sets the correct
                          1149                     ; flag. The INS flag is then set which causes the software to
                          1150                     ; temporary execute the insert routines and update the steps so
                          1151                     ; when the second move button is pressed the point may be moved
                          1152                     ; and the INS flag lowered so it returns to normal edit mode.
07AD 33C0                 1153                     XOR     AX,AX                   ;CLEAR THE STEP LOCATIONS
07AF A39A09          R    1154                     MOV     XSTEPS,AX               ;FOR THE INSERT ROUTINES TO
07B2 A39C09          R    1155                     MOV     YSTEPS,AX               ;UPDATE.
07B5 A39E09          R    1156                     MOV     ZSTEPS,AX               ;
07B8 A3A009          R    1157                     MOV     R0STEPS,AX              ;
07BB A3A209          R    1158                     MOV     R1STEPS,AX              ;
07BE A3A409          R    1159                     MOV     R2STEPS,AX              ;
07C1 A2BB09          R    1160                     MOV     ROT_FLAG,AL             ;
07C4 FE06B809        R    1161                     INC     INS                     ;SET THE INSERT FLAG SO THAT
07C8 FE06BC09        R    1162                     INC     XYZ_FLAG                ;INSERT ROUTINS ARE EXECUTED.
07CC BE0C00               1163                     MOV     SI,DEFSPEED             ;LOAD THE DEFAULT SPEED.
07CF 0B852000        R    1164                     OR      AX,XARRAY[DI-2]         ;IDENTIFY THE NATURE OF THE
07D3 0B85B001        R    1165                     OR      AX,YARRAY[DI-2]         ;POINT AND SET THE CORRECT
07D7 0B854003        R    1166                     OR      AX,ZARRAY[DI-2]         ;FLAG.
07DB 23C0                 1167                     AND     AX,AX                   ;
07DD 7508                 1168                     JNZ     IR_BNC                  ;INTERRUPT RETURN BOUNCE.
07DF FE0EBC09        R    1169                     DEC     XYZ_FLAG                ;CHANGE TO ROTATION.
07E3 FE06BB09        R    1170                     INC     ROT_FLAG                ;
07E7 E9EEF9               1171     IR_BNC:         JMP     SHORT INTER_RET         ;
                          1172
                          1173 +1  $EJECT
```

```
LOC  OBJ                  LINE     SOURCE

                          1174
07EA                      1175     TRACE_MOVE      PROC                            ;XYZ MOVEMENT ROUTINE.
                          1176                     ;Once the xyz maths is done this routine is executed to
                          1177                     ;count out the time and so wait until the line is fully
                          1178                     ;described. Once it is finished the routine moves to the
                          1179                     ;next point and checks if it is the end of the path.
07EA FF0EA809        R    1180                     DEC     TIMCNT                  ;DECREMENT THE TIME COUNTER AND
07EE 751C                 1181                     JNZ     END_MOVE                ;NO..... WAIT LONGER.
07F0 BA0804               1182                     MOV     DX,ENBPRTADDR           ;GET THE MOTORS ENABLE ADDRESS
07F3 32C0                 1183                     XOR     AL,AL                   ;AND CLEAR THE ENABLE FLAGS TO
07F5 EE                   1184                     OUT     DX,AL                   ;STOP THE MOTORS.
07F6 A2B709          R    1185                     MOV     MOVSTAT,AL              ;MOVING      = FALSE.
07F9 83BD220004      R    1186                     CMP     XARRAY[DI],4            ;ARE WE NOW POINTING TO
07FE 750C                 1187                     JNZ     END_MOVE                ;THE END OF PATH MARKER ?
0800 81BD42030080    R    1188                     CMP     ZARRAY[DI],8000H        ;IF SO SET FINISH FLAG.
0806 7504                 1189                     JNZ     END_MOVE                ;
0808 FE06B109        R    1190                     INC     FINISH
080C C3                   1191     END_MOVE:       RET
                          1192     TRACE_MOVE      ENDP
                          1193
----                      1194     CODE    ENDS
                          1195
                          1196     END

ASSEMBLY COMPLETE, NO ERRORS FOUND
```

5. WELDING PREPARATION TASK.

```
iRMX 86 PL/M-86 V2.3 COMPILATION OF MODULE WELDER
OBJECT MODULE PLACED IN WELD/PROG.OBJ
COMPILER INVOKED BY:  :LANG:plm86 WELD/PROG.P86


           $TITLE ('SUPERWELD  WELDING TASK VER 86/2.0 DATE 08/02/85')

           $DEBUG
           $LARGE
           $OPTIMIZE(0)

1          WELDER: DO;

           $INCLUDE (INC/LITERALS.P86)
           $SAVE NOLIST
           $INCLUDE (/INC/RMX86/NCRMBX.EXT)
           $SAVE NOLIST
           $INCLUDE (/INC/RMX86/NRCMES.EXT)
           $SAVE NOLIST
           $INCLUDE (/INC/RMX86/NLUOBJ.EXT)
           $SAVE NOLIST
           $INCLUDE (/INC/RMX86/NRSTSK.EXT)
           $SAVE NOLIST
           $INCLUDE (/INC/RMX86/NRCUNI.EXT)
           $SAVE NOLIST
           $INCLUDE (/INC/RMX86/NSUTSK.EXT)
           $SAVE NOLIST
           $INCLUDE (/INC/RMX86/NSTEXH.EXT)
           $SAVE NOLIST
           $INCLUDE (/INC/RMX86/NSTPRI.EXT)
           $SAVE NOLIST
           $INCLUDE (/INC/RMX86/NSNMES.EXT)
           $SAVE NOLIST
           $INCLUDE (/INC/RMX86/NDLMBX.EXT)
           $SAVE NOLIST
           $INCLUDE (/INC/RMX86/NDLSEG.EXT)
           $SAVE NOLIST
           $INCLUDE (/INC/RMX86/NCRSEG.EXT)
           $SAVE NOLIST
           $INCLUDE (/INC/RMX86/NSLEEP.EXT)
           $SAVE NOLIST
           $INCLUDE (/INC/RMX86/IASEEK.EXT)
           $SAVE NOLIST
           $EJECT
```

```
                $INCLUDE (/INC/RMX86/IADLCN.EXT)
              =  $SAVE NOLIST
                $INCLUDE (/INC/RMX86/IAREAD.EXT)
              =  $SAVE NOLIST
                $INCLUDE (/INC/RMX86/IAOPEN.EXT)
              =  $SAVE NOLIST
                $INCLUDE (/INC/RMX86/IACLOS.EXT)
              =  $SAVE NOLIST
                $INCLUDE (/INC/RMX86/IAGTPC.EXT)
              =  $SAVE NOLIST
                $INCLUDE (/INC/RMX86/IWTIO.EXT)
              =  $SAVE NOLIST

                /***************************** LOCALS ***********************************/

157   1         declare (loop,number_sets,sets_read,welds_done)             byte;
158   1         declare wrt_buf_ptr                          pointer;
159   1         declare wrt_buf_vals structure (offset word,base word) AT   (@wrt_buf_ptr);
160   1         declare (wrt_buf based wrt_buf_ptr)(80)                  byte;
161   1         declare real_save_area(94)                     byte;
162   1         declare (status,gen_word,data_status)                    word;
163   1         declare (path_mailbox,data_mailbox,weld_sem,th_in_mbx,th_out_mbx)   token;
164   1         declare (data_connection,path_connection,read_in,read_out,co)       token;
165   1         declare (main_task_token,gen_mailbox)                    token;
166   1         declare path_info structure  (mat_dim word,num_points word,num_spln byte,
                                                num_weld byte);
167   1         declare error1(*)  byte  data  (22,0,29,'No data file has been loaded!');
168   1         declare error2(*)  byte  data  (22,0,29,'No path file has been loaded!');
169   1         declare error3(*)  byte  data  (22,0,31,'A data frame error has occured!');
170   1         declare error4(*)  byte  data  (22,0,24,'Current demand to large!');
171   1         declare error5(*)  byte  data  (22,0,22,'Timming error occured!');
172   1         declare error6(*)  byte  data  (22,0,19,'Data/Path mismatch!');
173   1         declare message(*) byte  data  (20,0,19,'***** WELDING *****');

                $EJECT
```

```
                /************************* REAL DATA MATRIX ******************************/

1/4   1         declare data_set     structure   (I_p         real,
                                                 l_b      real,
                                                 t_p      real,
                                                 t_b      real,
                                                 I_s1         real,
                                                 I_s2         real,
                                                 t_s1         real,
                                                 t_s2         real,
                                                 t_su         real,
                                                 t_sd         real,
                                                 t_g1         real,
                                                 t_g2         real,
                                                 V_t      real,
                                                 P_m      real,
                                                 spare0       real,
                                                 spare1       real,
                                                 spare2       real,
                                                 spare3       real,
                                                 spare4       real,
                                                 spare5       byte,
                                                 spare6       byte,
                                                 set      byte,
                                                 stop         byte);
175   1         declare (I_psu,I_inc)                    real;

                $EJECT
```

```
                /*************************** EXTERNALS *********************************/
                        /******************/
                        /* FROM ASM PROG */
                        /******************/

176   1         asmweld: procedure external;
177   2         end asmweld;

178   1         seam_time: procedure external;
179   2         end seam_time;

180   1         declare (path_finished,math_done)           byte external;
181   1         declare (t_pt,t_bt)                    byte external;
182   1         declare (pen_monf,pulse_flags)              byte external;
183   1         declare (splncnt,weldcnt)                 byte external;
184   1         declare (PRES_I,C1,flags)                 word external;
185   1         declare (ptbytes,ptcount,path_index)            word external;
186   1         declare (t_s1t,t_s2t,t_sut)               word external;
187   1         declare (t_sdt,t_g1t,t_g2t,t_mt)            word external;
188   1         declare (I_bw,I_s1w,I_s2w)                word external;
189   1         declare (I_uw,I_dw,I_pw)                  word external;
190   1         declare (Velocity,t_seam)                 real external;
191   1         declare (r0array,r1array,r2array)(200)          word external;
192   1         declare (xarray,yarray,zarray)(200)         word external;

                $EJECT
```

```
                /****************************** PROCEDURES ******************************/

193   1         INSERT_BYTE: PROCEDURE(byte_no);

                /* This procedure take a byte number as a parameter and converts it to its
                   ASCII  equivalent and  inserts the ASCII into the write buffer with its
                   leading zeros supressed with spaces. */

194   2         declare byte_no                         byte;

195   2         wrt_buf(2)=3;
196   2         do loop=0 to 2;
197   3            wrt_buf(5-loop)=(byte_no MOD 10)+30H;
198   3            byte_no=byte_no/10;
199   3         end;
200   2         if wrt_buf(3)='0' then
201   2           do;
202   3              wrt_buf(3)=space;
203   3              if wrt_buf(4)='0' then wrt_buf(4)=space;
205   3           end;

206   2         END INSERT_BYTE;

                $EJECT
```

```
207   1     FIXWORD: PROCEDURE(real_no) WORD;

            /* This routine takes a real number as its parameter and returns it as word
               number. It is assumed that no overflow will occur and this is assured by
               the  calling  routines.  Overflow  however must be  taken care of as the
               number passes thruogh the Integer stage. */

208   2     declare real_no                    real;
209   2     declare word_no                    word;
210   2     declare ovrflw                     byte;

211   2     ovrflw=false;
212   2     if real_no>32767.0 then
213   2       do;
214   3          ovrflw=true;
215   3          real_no=real_no-32767.0;
216   3       end;
217   2     word_no=UNSIGN(FIX(real_no));
218   2     if ovrflw then word_no=word_no+32767;
220   2     return word_no;

221   2     END FIXWORD;

            $EJECT
```

```
222   1     GET_DATA: PROCEDURE WORD;

            /* This routine sets the data file connection from the data mailbox. If no data
               file is found there the error is reported to the screen and the weld command
               aborts. If a data file is  found then it is opened and the number of sets of
               data that it contains is read. */

223   2     data_connection=RQ$RECEIVE$MESSAGE(data_mailbox,no_waiting,no_response,@status);
224   2     if status<>E$OK then
225   2       do;
226   3          call MOVB(@error1,@wrt_buf,32);
227   3          call write_message;
228   3          return error_flag;
229   3       end;
230   2     call RQ$A$OPEN(data_connection,file_read,share_mode,no_response,@status);
231   2     call RQ$A$READ(data_connection,@number_sets,one,gen_mailbox,@status);
232   2     gen_word=RQ$WAIT$IO(data_connection,gen_mailbox,wait_forever,@status);
233   2     sets_read=0;
234   2     return E$OK;

235   2     END GET_DATA;

            $EJECT
```

```
236   1     GET_PATH: PROCEDURE WORD;

            /* As in the get data routine this routine calls at the path mailbox and tries
               to  receive a token for a path file. If no tokens are waiting then the weld
               command is  aborted. The data  file  connection that must have been already
               read is sent  back. The file is then opened and the  information matrix and
               the path  matrix are then read.  Framing error in the data are  checked for
               and the routine returns. */

237   2     declare (bytes_read1,bytes_read2,bytes_read3,bytes_read4,bytes_read5,bytes_read6) word;

238   2     path_connection=RQ$RECEIVE$MESSAGE(path_mailbox,no_waiting,no_response,@status);
239   2     if status<>E$OK then
240   2       do;
241   3          call MOVB(@error2,@wrt_buf,33);
242   3          call write_message;
243   3          call RQ$SEND$MESSAGE(data_mailbox,data_connection,no_response,@status);
244   3          return error_flag;
245   3       end;
246   2     call RQ$A$OPEN(path_connection,file_read,share_mode,no_response,@status);
247   2     call RQ$A$READ(path_connection,@path_info,6,gen_mailbox,@status);
248   2     gen_word=RQ$WAIT$IO(path_connection,gen_mailbox,wait_forever,@status);
249   2     call RQ$A$SEEK(path_connection,address_mode,416,gen_mailbox,@status);
250   2     bytes_read1=RQ$WAIT$IO(path_connection,gen_mailbox,wait_forever,@status);
251   2     call RQ$A$READ(path_connection,@xarray,path_info.mat_dim,gen_mailbox,@status);
252   2     bytes_read1=RQ$WAIT$IO(path_connection,gen_mailbox,wait_forever,@status);
253   2     call RQ$A$READ(path_connection,@yarray,path_info.mat_dim,gen_mailbox,@status);
254   2     bytes_read2=RQ$WAIT$IO(path_connection,gen_mailbox,wait_forever,@status);
255   2     call RQ$A$READ(path_connection,@zarray,path_info.mat_dim,gen_mailbox,@status);
256   2     bytes_read3=RQ$WAIT$IO(path_connection,gen_mailbox,wait_forever,@status);
257   2     call RQ$A$READ(path_connection,@r0array,path_info.mat_dim,gen_mailbox,@status);
258   2     bytes_read4=RQ$WAIT$IO(path_connection,gen_mailbox,wait_forever,@status);
259   2     call RQ$A$READ(path_connection,@r1array,path_info.mat_dim,gen_mailbox,@status);
260   2     bytes_read5=RQ$WAIT$IO(path_connection,gen_mailbox,wait_forever,@status);
261   2     call RQ$A$READ(path_connection,@r2array,path_info.mat_dim,gen_mailbox,@status);
262   2     bytes_read6=RQ$WAIT$IO(path_connection,gen_mailbox,wait_forever,@status);
263   2     if (bytes_read1<>path_info.mat_dim OR bytes_read2<>path_info.mat_dim
                 OR bytes_read3<>path_info.mat_dim OR bytes_read4<>path_info.mat_dim
                 OR bytes_read5<>path_info.mat_dim OR bytes_read6<>path_info.mat_dim) then
264   2       do;
265   3          call MOVB(@error2,@wrt_buf,33);
266   3          call write_message;
267   3          return error_flag;
268   3       end;
269   2     ptbytes=path_info.mat_dim;
270   2     ptcount=path_info.num_points;
271   2     weldcnt=path_info.num_weld;
272   2     splncnt=path_info.num_spln;
273   2     return E$OK;

274   2     END GET_PATH;
```

```
275   1       READ_DATA_SET: PROCEDURE WORD;

              /* This routine reads one complete set of data if there are any left in the
                 file, if not we  rewind one  set so the same  set is read again. Framing
                 errors are check for in the data. */

276   2       if sets_read<number_sets then sets_read=sets_read+1;
278   2       else
                do;
279   3          call RQ$A$SEEK(data_connection,backwards,one_set,gen_mailbox,@status);
280   3          gen_word=RQ$WAIT$IO(data_connection,gen_mailbox,wait_forever,@status);
281   3         end;
282   2       call RQ$A$READ(data_connection,@data_set,one_set,gen_mailbox,@status);
283   2       gen_word=RQ$WAIT$IO(data_connection,gen_mailbox,wait_forever,@status);
284   2       if data_set.stop<>0FFH then
285   2        do;
286   3           call MOVB(@error3,@wrt_buf,34);
287   3           call write_message;
288   3           return error_flag;
289   3        end;
290   2       return E$OK;

291   2       END READ_DATA_SET;

              $EJECT
```

```
292   1     PREPARE_DATA: PROCEDURE WORD;

            /* This routine prepares all the data for the asm86 routine. I_psu is the
               maximum current the power supply will give. The assembly language routine
               which calculates the next seam time is called (it is written in ASM86 due
               to the lack of a square root in PL/M). The times from the data file are
               then converted to multiples of 10mS as this is our base unit of time. The
               current are converted to analogue increments and inlined for the iSBX 328.
               The current flags are then setup to indicate to the current handler which
               routines need to be called and the starting currents filled in. */

293   2     declare main_t_real                         real;

294   2     data_status=read_data_set;
295   2     if data_status<>E$OK then return error_flag;
297   2     Velocity=data_set.V_t;
298   2     call seam_time;
299   2     I_psu=500.0;
300   2     if data_set.I_p>I_psu then
301   2       do;
302   3         call MOVB(@error4,@wrt_buf,27);
303   3         call write_message;
304   3         return error_flag;
305   3       end;
306   2     if data_set.P_m<>0.0 then
307   2       do;
308   3          data_set.t_p=data_set.t_p*1.2;
309   3          pen_monf=on;
310   3       end;
311   2     else pen_monf=off;
            $EJECT
```

```
312   2      t_pt=LOW(FIXWORD(data_set.t_p*100.0));
313   2      t_bt=LOW(FIXWORD(data_set.t_b*100.0));
314   2      t_s1t=FIXWORD(data_set.t_s1*100.0);
315   2      t_s2t=FIXWORD(data_set.t_s2*100.0);
316   2      t_sut=FIXWORD(data_set.t_su*100.0);
317   2      t_sdt=FIXWORD(data_set.t_sd*100.0);
318   2      t_g1t=FIXWORD(data_set.t_g1*100.0);
319   2      t_g2t=FIXWORD(data_set.t_g2*100.0);
320   2      main_t_real=t_seam-((data_set.t_su-data_set.t_sd)*100.0);
321   2      if main_t_real<0.0 then
322   2        do;
323   3          call MOVB(@error5,@wrt_buf,25);
324   3          call write_message;
325   3          return error_flag;
326   3        end;
327   2      t_mt=FIXWORD(main_t_real);
328   2      I_inc=I_psu/4095.0;
329   2      I_pw=SHL(FIXWORD(data_set.I_p/I_inc),4);
330   2      I_bw=SHL(FIXWORD(data_set.I_b/I_inc),4);
331   2      I_s1w=SHL(FIXWORD(data_set.I_s1/I_inc),4);
332   2      I_s2w=SHL(FIXWORD(data_set.I_s2/I_inc),4);
333   2      if data_set.t_su<>0.0 then I_uw=SHL(FIXWORD((data_set.I_p-data_set.I_s1)/(data_set.t_su*100.0*I_inc)),4);
335   2      else I_uw=0;
336   2      if data_set.t_sd<>0.0 then I_dw=SHL(FIXWORD((data_set.I_p-data_set.I_s2)/(data_set.t_sd*100.0*I_inc)),4);
338   2      else I_dw=0;
339   2      if t_bt<>0 then pulse_flags=02H;
341   2      else pulse_flags=00H;
342   2      flags=0100100010000000B;                          /* 4880H */
343   2      if t_g1t<>0 then flags=flags OR 1000000001000000B; /* 0804H */
345   2      if t_s1t<>0 then flags=flags OR 0010000000000000B; /* 2000H */
347   2      if t_sdt<>0 then flags=flags OR 0000010000000000B; /* 0400H */
349   2      if t_s2t<>0 then flags=flags OR 0000001100000000B; /* 0300H */
351   2      if t_sut<>0 then flags=flags OR 0001000000000000B; /* 1000H */
353   2      PRES_I=I_s1w;
354   2      C1=I_s1w;
355   2      if (t_sut=0 AND t_s1t=0) then C1=I_pw;
357   2      if t_sut=0 then PRES_I=I_pw;
359   2      return E$OK;

360   2      END PREPARE_DATA;

              $EJECT
```

```
361   1       DISPLAY_FILE_NAMES: PROCEDURE;

              /* This routine simply displays on the screen the names of the path and data
                 files that are about to be welded with. */

362   2       declare file_name_ptr                    pointer;
363   2       declare file_name based file_name_ptr structure (status word,name(14) byte);
364   2       declare file_seg  structure  (offset word,base word)  AT   (@file_name_ptr);

365   2       call MOVB(clear_screen,@wrt_buf,6);
366   2       call write_message;
367   2       call SETB(space,@wrt_buf,80);
368   2       call MOVB(@(5,0,52,ESC,'$3'),@wrt_buf,6);
369   2       file_seg.offset=null;
370   2       call RQ$A$GET$PATH$COMPONENT(data_connection,gen_mailbox,@status);
371   2       file_seg.base=RQ$RECEIVE$MESSAGE(gen_mailbox,wait_forever,no_response,@status);
372   2       call MOVB(@file_name.name,@wrt_buf(6),14);
373   2       call RQ$DELETE$SEGMENT(file_seg.base,@status);
374   2       call RQ$A$GET$PATH$COMPONENT(path_connection,gen_mailbox,@status);
375   2       file_seg.base=RQ$RECEIVE$MESSAGE(gen_mailbox,wait_forever,no_response,@status);
376   2       call MOVB(@file_name.name,@wrt_buf(36),14);
377   2       call RQ$DELETE$SEGMENT(file_seg.base,@status);
378   2       call write_message;
379   2       wrt_buf(5)='4';
380   2       wrt_buf(0)=6;
381   2       call write_message;

382   2       END DISPLAY_FILE_NAMES;

              $EJECT
```

```
383   1       COMPARE: PROCEDURE BYTE;

              /* This routine  displays the number of sets of data  under the data file name
                 and the number of seams under the path file name. If then number of sets of
                 data is less than the number of seams the last set of data will be reused. */

384   2       call insert_byte(number_sets);
385   2       wrt_buf(0)=8;
386   2       wrt_buf(1)=4;
387   2       call write_message;
388   2       call insert_byte(path_info.num_weld);
389   2       wrt_buf(1)=44;
390   2       call write_message;
391   2       return E$OK;

392   2       END COMPARE;

393   1       WRITE_MESSAGE: PROCEDURE;

394   2       call RQ$SEND$MESSAGE(th_in_mbx,wrt_buf_vals.base,th_out_mbx,@status);
395   2       wrt_buf_vals.base=RQ$RECEIVE$MESSAGE(th_out_mbx,wait_forever,no_response,@status);

396   2       END WRITE_MESSAGE;

397   1       SET_EH: PROCEDURE;

398   2       declare e_h structure (offset word,base word,mode byte);

399   2       e_h.offset,e_h.base,e_h.mode=0;
400   2       call RQ$SET$EXCEPTION$HANDLER(@e_h,@status);

401   2       END SET_EH;

              $EJECT
```

```
                /****************************** MAIN LOOP ***************************************/

402   1         WELD_TASK: PROCEDURE PUBLIC;

                /* Once  the weld task has  intialised  itself by  looking up all the required
                   tokens in the  Job  directory it waits for  one unit at its own  semaphore.
                   When this unit is  received it  suspends the  main task and  saves the real
                   stack, then  creates its terminal  communication  buffer. It then retrieves
                   the path and  data file  connections  from the  mailboxes, checking each is
                   done successfully or aborting by  restarting the main task and  waiting for
                   the next semaphore unit. The first data set is then read and the  filenames
                   display along with the set and seam information. The path index is then set
                   to the start of the path and the main welding loop is entered. This loop is
                   left on the  path_finished flag  being set. If all the  welds have not been
                   completed the data is then prepared and the welding message  displayed. The
                   task then  sleeps to allow the other tasks such as the  terminal write task
                   to  finish.  Its  priority  is  then  raised to  stop any  lower  levels of
                   interrupts from occuring. The ASM86 program is then called, this takes care
                   of all the  welding and only  returns when the path is  finished or another
                   set of data is required. Once all the welds are completed this loop is left
                   and the files are  closed and detached, the screen is  cleared and the real
                   stack is  restored.  Finally the  Supervisor is  resumed and the  weld task
                   waits for the next unit at its semaphore. */

403   2         th_in_mbx=RQ$LOOKUP$OBJECT(this_Job,@(7,'THINMBX'),wait_forever,@status);
404   2         th_out_mbx=RQ$LOOKUP$OBJECT(this_Job,@(8,'THOUTMBX'),wait_forever,@status);
405   2         read_in=RQ$LOOKUP$OBJECT(this_Job,@(6,'READIN'),wait_forever,@status);
406   2         read_out=RQ$LOOKUP$OBJECT(this_Job,@(7,'READOUT'),wait_forever,@status);
407   2         gen_mailbox=RQ$LOOKUP$OBJECT(this_Job,@(4,'GMBX'),wait_forever,@status);
408   2         data_mailbox=RQ$LOOKUP$OBJECT(this_Job,@(4,'DMBX'),wait_forever,@status);
409   2         path_mailbox=RQ$LOOKUP$OBJECT(this_Job,@(4,'PMBX'),wait_forever,@status);
410   2         main_task_token=RQ$LOOKUP$OBJECT(this_Job,@(4,'MTSK'),wait_forever,@status);
411   2         weld_sem=RQ$LOOK$UP$OBJECT(this_Job,@(4,'WSEM'),wait_forever,@status);
412   2         co=RQ$LOOK$UP$OBJECT(this_Job,@(2,'CO'),wait_forever,@status);
413   2         wrt_buf_vals.offset=null;
414   2         call set_eh;
                $EJECT
```

```
415   2       do forever;
416   3         gen_word=RQ$RECEIVE$UNITS(weld_sem,one,wait_forever,@status);
417   3         call RQ$SUSPEND$TASK(main_task_token,@status);
418   3         call SAVE$REAL$STATUS(@real_save_area);
419   3         wrt_buf_vals.base=RQ$CREATE$SEGMENT(80,@status);
420   3         data_status=get_path;
421   3         if data_status=E$OK then data_status=get_data;
423   3         if data_status=E$OK then
424   3           do;
425   4             call display_file_names;
426   4             data_status=compare;
427   4           end;
428   3         path_finished=bad;
429   3         path_index=0;
430   3         welds_done=0;
431   3         do while path_finished=bad;
432   4            if (data_status=E$OK AND welds_done<path_info.num_weld) then data_status=prepare_data;
434   4            if data_status=E$OK then
435   4              do;
436   5                call MOVB(@message,@wrt_buf,22);
437   5                call write_message;
438   5                call RQ$SLEEP(100,@status);
439   5                call RQ$SET$PRIORITY(this_task,while_weld_priority,@status);
440   5                call asmweld;
441   5                call RQ$SET$PRIORITY(this_task,weld_priority,@status);
442   5                welds_done=welds_done+1;
443   5              end;
444   4            else
                     do;
445   5                call RQ$SLEEP(500,@status);
446   5                path_finished=good;
447   5              end;
448   4       end;
449   3         call MOVB(bell,@wrt_buf,4);
450   3         call write_message;
451   3         call RQ$A$CLOSE(data_connection,no_response,@status);
452   3         call RQ$A$CLOSE(path_connection,no_response,@status);
453   3         call RQ$A$DELETE$CONNECTION(data_connection,no_response,@status);
454   3         call RQ$A$DELETE$CONNECTION(path_connection,no_response,@status);
455   3         call MOVB(clear_screen,@wrt_buf,6);
456   3         call write_message;
457   3         call RQ$DELETE$SEGMENT(wrt_buf_vals.base,@status);
458   3         call RESTORE$REAL$STATUS(@real_save_area);
459   3         call RQ$RESUME$TASK(main_task_token,@status);
460   3       end;

461   2       END WELD_TASK;

462   1       END WELDER;

              $EJECT
```

```
MODULE INFORMATION:

     CODE AREA SIZE     = 0DB6H   3510D
     CONSTANT AREA SIZE = 01B4H    436D
     VARIABLE AREA SIZE = 00FEH    254D
     MAXIMUM STACK SIZE = 001EH     30D
     985 LINES READ
     0 PROGRAM WARNINGS
     0 PROGRAM ERRORS

DICTIONARY SUMMARY:

     116KB MEMORY AVAILABLE
     16KB MEMORY USED   (13%)
     0KB DISK SPACE USED

END OF PL/M-86 COMPILATION
```

6. WELDING POWER SUPPLY AND ROBOT DRIVER.

```
iRMX 86 8086/87/88/186 MACRO ASSEMBLER V2.0 ASSEMBLY OF MODULE WELDING_DRIVER
OBJECT MODULE PLACED IN DRIVERS/WELD.OBJ
ASSEMBLER INVOKED BY:  :LANG:ASM86 DRIVERS/WELD.A86

LOC  OBJ                  LINE     SOURCE

                            1 +1   $DEBUG
                            2 +1   $INCLUDE(INC/LITERALS.A86)
                     =1     3 +1   $SAVE NOLIST
                           69 +1   $TITLE ('SUPERWELD TUNGSTEN INERT GAS WELDING DRIVER VER 86/4.0 DATE 12/03/85')
                           70
                           71      NAME WELDING_DRIVER
                           72
                           73      ASSUME CS:CODE,DS:DATA
                           74
----                       75      DATA SEGMENT PARA PUBLIC 'DATA'
                           76
0000 FFFF                  77      SPEEDS          DW      0FFFFH,05F8H,02FCH,01FDH,017EH,0131H,00FEH,00DAH,0C0H
0002 F805
0004 FC02
0006 FD01
0008 7E01
000A 3101
000C FE00
000E DA00
0010 C000
0012 ????                  78      XTOTAL          DW      ?                       ;TOTAL COUNTS (DISPLACEMENT
0014 ????                  79      YTOTAL          DW      ?                       ;FROM ORIGIN).
0016 ????                  80      ZTOTAL          DW      ?                       ;
0018 ????                  81      R0TOTAL         DW      ?                       ;
001A ????                  82      R1TOTAL         DW      ?                       ;
001C ????                  83      R2TOTAL         DW      ?                       ;
                           84
001E ????                  85      XSPEED          DW      ?                       ;SPEED STORES FOR TRACER
0020 ????                  86      YSPEED          DW      ?                       ;
0022 ????                  87      ZSPEED          DW      ?                       ;
0024 ????                  88      R0SPEED         DW      ?                       ;
0026 ????                  89      R1SPEED         DW      ?                       ;
0028 ????                  90      R2SPEED         DW      ?                       ;
                           91
                           92 +1   $EJECT
```

```
LOC  OBJ                  LINE     SOURCE

                            93
002A ????                   94     STATUS          DW      ?               ;ERROR WORD FOR RMX CALLS.
002C ????                   95     TIMCNT          DW      ?               ;NUMBER OF 10mS PULSES.
002E ??                     96     WFINISH         DB      ?               ;END OF PATH FLAG.
002F ??                     97     DIRFLAGS        DB      ?               ;DIRECTION SIGN FLAGS FOR AUTO MOVE.
0030 ??                     98     ENBFLAGS        DB      ?               ;ENABLE FLAG STORE FOR AUTO MOVE.
0031 ??                     99     WELD_STOP       DB      ?               ;MOVE WHILE WELD INDICATOR.
0032 ??                    100     CURRENT         DB      ?               ;CURRENT ON INDICATOR
0033 ??                    101     D_E_FLAGS       DB      ?               ;DIRECTION AND ENABLE FLAGS.
0034 ??                    102     MATH_DONE       DB      ?               ;INDICATES IF LINE MATHS DONE.
0035 ????                  103     ROT_TIME_ST     DW      ?               ;ROTATION TIME STORE.
0037 ??                    104     ROT_ON          DB      ?
                           105
                           106 +1  $EJECT
```

```
LOC  OBJ                  LINE     SOURCE

                          107
                          108      ;NEVER CLEAR
                          109
0038 0006                 110      FREQ            DW      1536                    ;CLOCK FREQUENCY.
                          111
003A D304           R     112      BRANCH_TABLE    DW      GAS_ON
003C E104           R     113                      DW      G_COUNT
003E 1405           R     114                      DW      TS1_COUNT
0040 2105           R     115                      DW      TSU_COUNT
0042 5505           R     116                      DW      MAIN_COUNT
0044 6B05           R     117                      DW      TSD_COUNT
0046 8F05           R     118                      DW      IS2_OUT
0048 AC05           R     119                      DW      TS2_COUNT
004A B905           R     120                      DW      ARC_OUT
004C C705           R     121                      DW      GAS_OFF
                          122
004E ????                 123      FLAGS           DW      ?               ;POWER SUPPLY TIMING AND CURRENTS
0050 ????                 124      I_s1w           DW      ?               ;FROM THE DATA FILE.
0052 ????                 125      I_s2w           DW      ?
0054 ????                 126      t_s1t           DW      ?
0056 ????                 127      t_sut           DW      ?
0058 ????                 128      t_s1t           DW      ?
005A ????                 129      t_s2t           DW      ?
005C ????                 130      t_mt            DW      ?
005E ????                 131      t_sdt           DW      ?
0060 ????                 132      t_s2t           DW      ?
0062 ??                   133      t_bt            DB      ?
0063 ??                   134      t_pt            DB      ?
0064 ????                 135      I_uw            DW      ?
0066 ????                 136      I_dw            DW      ?
0068 ????                 137      PRES_I          DW      ?
006A ????                 138      C1              DW      ?
006C ????                 139      I_bw            DW      ?
006E ????                 140      I_pw            DW      ?
0070 ????????             141      Velocity        DD      ?               ;REQUIRED TORCH VELOCITY
0074 ????????             142      t_seam          DD      ?               ;WELDING TIME
0078 ??                   143      p_pres          DB      ?               ;PRESENT CURRENT STORE
0079 ??                   144      pen_monf        DB      ?               ;PENETRATION MONITOR FLAG
007A ??                   145      PULSE_FLAGS     DB      ?               ;PULSE/BACKGROUND CURRENT FLAG
007B FF                   146      PORT1STR        DB      0FFH            ;STORES WORD PRESENT AT PORT 0CCH
007C ????                 147      PATH_INDEX      DW      ?               ;STORES DI FOR SEAM_TIME TO USE
007E ??                   148      PATH_FINISHED   DB      ?               ;PATH FINISHED FLAG
                          149
                          150 +1   $EJECT
```

```
LOC  OBJ                  LINE     SOURCE

                           151
                           152     ;****************** PUBLICS AND EXTERNALS FOR PL/M HALF OF TASK ****************
                           153
                           154     PUBLIC  t_sut,t_sdt,t_s1t
                           155     PUBLIC  t_s2t,t_bt,t_pt
                           156     PUBLIC  t_mt,t_g1t,t_g2t
                           157     PUBLIC  PORT1STR,I_pw,Velocity
                           158     PUBLIC  t_seam,pen_monf
                           159     PUBLIC  PATH_FINISHED,PATH_INDEX
                           160     PUBLIC  I_uw,I_dw,I_bw
                           161     PUBLIC  PRES_I,C1,FLAGS,PULSE_FLAGS
                           162     PUBLIC  I_s1w,I_s2w,MATH_DONE
                           163     EXTRN   PTBYTES:WORD
                           164     EXTRN   SPLNCNT:BYTE
                           165     EXTRN   WELDCNT:BYTE
                           166     EXTRN   PTCOUNT:WORD
                           167     EXTRN   XARRAY:WORD
                           168     EXTRN   YARRAY:WORD
                           169     EXTRN   ZARRAY:WORD
                           170     EXTRN   R0ARRAY:WORD
                           171     EXTRN   R1ARRAY:WORD
                           172     EXTRN   R2ARRAY:WORD
                           173
----                       174     DATA ENDS
                           175
                           176     EXTRN   RQEXITINTERRUPT:FAR,RQSETINTERRUPT:FAR,RQRESETINTERRUPT:FAR
                           177
                           178 +1  $EJECT
```

```
LOC  OBJ                  LINE     SOURCE

                          179
----                      180      CODE SEGMENT PARA PUBLIC 'CODE'
                          181
                          182      ;*****************************************************************************
                          183
0000                      184      SEAM_TIME       PROC    FAR
                          185                      PUBLIC  SEAM_TIME
                          186                      ; This routine is called by the PL/M software totally
                          187                      ; independent of the rest of the below software. It is
                          188                      ; present in the assembly language file totally because
                          189                      ; of the lack of a square root function in PL/M. The
                          190                      ; SEAM_TIME routine firstly finds the next start of weld
                          191                      ; indicator on the path, then using pythagoras and knowing
                          192                      ; the velocity the time for each line on the weld can be
                          193                      ; calculated. The total time for the weld can then be
                          194                      ; calculated by adding the rotation times, Knowing that
                          195                      ; the rotation does 8 steps in 10mS apart from the last
                          196                      ; which is 10mS for the overflow. The time is returned to
                          197                      ; the PL/M via the public storage location t_seam.
0000 1E                   198                      PUSH    DS                      ;SAVE OLD DATA SEGMENT
0001 06                   199                      PUSH    ES
0002 BB----         R     200                      MOV     BX,DATA                 ;GET NEW DATA SEGMENT
0005 8EDB                 201                      MOV     DS,BX                   ;AND ES SEGMENT
0007 8EC3                 202                      MOV     ES,BX                   ;
0009 9BD9EE               203                      FLDZ                            ;T_SEAM=0.0
000C C70635000000   R     204                      MOV     ROT_TIME_ST,0           ;CLEAR ROTATION TIME
0012 8B3E7C00       R     205                      MOV     DI,PATH_INDEX           ;START LOOKING HERE FOR START
0016 E85F00               206                      CALL    FIND_SW                 ;FIND START OF WELD
0019 47                   207      NEXT_SEG:       INC     DI
001A 47                   208                      INC     DI                      ;NEXT POINT NOT WANTED
001B 81BD00000080   E     209                      CMP     ZARRAY[DI],8000H        ;IS IT A CONTROL POINT ?
0021 750A                 210                      JNZ     NOT_CNTRL               ;NO.........
0023 8B850000       E     211                      MOV     AX,XARRAY[DI]           ;
0027 47                   212                      INC     DI                      ;MOVE PAST THE CONTROL
0028 47                   213                      INC     DI                      ;
0029 23C0                 214                      AND     AX,AX                   ;IS IT AN END WELD POINT ?
002B 7438                 215                      JZ      CALC_LEN                ;FOUND END POINT. YES......
002D 33C0                 216      NOT_CNTRL:      XOR     AX,AX                   ;IS IT A ROTATION POINT ?
002F 0B850000       E     217                      OR      AX,XARRAY[DI]           ;(SIGNIFIED BY X, Y AND
0033 0B850000       E     218                      OR      AX,YARRAY[DI]           ;Z POINTS BEING ZERO)
0037 0B850000       E     219                      OR      AX,ZARRAY[DI]
003B 23C0                 220                      AND     AX,AX
003D 744D                 221                      JZ      ROT_TIME                ;YES.......(ROTATION POINT).
                          222 +1   $EJECT
```

```
LOC  OBJ                  LINE     SOURCE

003F 9BDF850000      E     223                 FILD    WORD PTR XARRAY[DI]     ;LOAD ALL OFFSETS (X,Y & Z)
0044 9BDCC8                224                 FMUL    ST,ST                   ;AND SQUARE THEM.
0047 9BDF850000      E     225                 FILD    WORD PTR YARRAY[DI]     ;
004C 9BDCC8                226                 FMUL    ST,ST
004F 9BDF850000      E     227                 FILD    WORD PTR ZARRAY[DI]     ;
0054 9BDCC8                228                 FMUL    ST,ST                   ;
0057 9BDEC1                229                 FADDP   ST(1),ST                ;ST=(X*X+Y*Y)
005A 9BDEC1                230                 FADDP   ST(1),ST                ;ST=(X*X+Y*Y+Z*)
005D 9BD9FA                231                 FSQRT                           ;ST=LENGTH OF STRAIGHT LINE.
0060 9BDEC1                232                 FADDP   ST(1),ST                ;= TOTAL LENGTH SO FAR.
0063 EBB4                  233                 JMP     SHORT NEXT_SEG          ;GO GET NEXT LINE.....
0065 9BD8367000      R     234     CALC_LEN:   FDIV    DWORD PTR Velocity      ;ST = SEAM TIME
006A 9BDE063500      R     235                 FIADD   WORD PTR ROT_TIME_ST    ;ADD ROTATION TIME
006F 9BD91E7400      R     236                 FSTP    DWORD PTR t_seam        ;SAVE FOR PL/M
0074 9B                    237                 FWAIT                           ;WAIT FOR STORE TO FINISH
0075 07                    238                 POP     ES                      ;
0076 1F                    239                 POP     DS                      ;
0077 CB                    240                 RET                             ;RETURN TO PLM
                           241     SEAM_TIME   ENDP
                           242
0078                       243     FIND_SW     PROC
0078 81BD00000080    E     244     NOT_CONTRL: CMP     ZARRAY[DI],8000H        ;IS IT A CONTROL POINT ?
007E 7508                  245                 JNZ     NOT_SW                  ;NO..... TRY NEXT POINT
0080 83BD000001      E     246                 CMP     XARRAY[DI],1            ;IS IS WELD START POINT ?
0085 7501                  247                 JNZ     NOT_SW                  ;NO..... TRY NEXT POINT
0087 C3                    248                 RET                             ;FOUND WELD START.
0088 47                    249     NOT_SW:     INC     DI                      ;MOVE POINTER TO NEXT POINT
0089 47                    250                 INC     DI                      ;
008A EBEC                  251                 JMP     SHORT FIND_SW           ;TEST NEXT POINT.....
                           252     FIND_SW     ENDP
                           253
008C 8B850000        E     254     ROT_TIME:   MOV     AX,R0ARRAY[DI]          ;FIRST FIND THE LARGEST
0090 3B850000        E     255                 CMP     AX,R1ARRAY[DI]          ;ROTATION AS THIS WILL
0094 7D04                  256                 JGE     NO_SWOP                 ;TAKE THE LONGEST.
0096 8B850000        E     257                 MOV     AX,R1ARRAY[DI]
009A 3B850000        E     258     NO_SWOP:    CMP     AX,R2ARRAY[DI]
009E 7DFA                  259                 JGE     NO_SWOP
00A0 8B850000        E     260                 MOV     AX,R2ARRAY[DI]
00A4 D1E8                  261     NO_SWOP2:   SHR     AX,1                    ;DIVIDE LARGEST BY 8
00A6 D1E8                  262                 SHR     AX,1                    ;(8 STEPS IN 10mS)
00A8 D1E8                  263                 SHR     AX,1                    ;
00AA 40                    264                 INC     AX                      ;ADD ONE FOR REMAINDER
00AB 01063500        R     265                 ADD     ROT_TIME_ST,AX          ;ADD TO TOTAL ROTATION TIME
00AF E967FF                266                 JMP     SHORT NEXT_SEG          ;NEXT POINT
                           267
                           268 +1  $EJECT
```

```
LOC  OBJ                  LINE     SOURCE

                          269
                          270      ;*****************************************************************************
                          271
00B2                      272      ASMWELD         PROC    FAR                     ;CALLED BY PLM
                          273                      PUBLIC  ASMWELD                 ;
00B2 1E                   274                      PUSH    DS                      ;SAVE OLD DATA SEGMENT
00B3 06                   275                      PUSH    ES                      ;SAVE EXTRA SEGMENT
00B4 BB----         R     276                      MOV     BX,DATA                 ;GET NEW DATA SEGMENT
00B7 8EDB                 277                      MOV     DS,BX                   ;AND ES SEGMENT
00B9 8EC3                 278                      MOV     ES,BX                   ;FOR STRING OPERATIONS
00BB BAE30190       R     279                      MOV     DX,OFFSET WELD_INT      ;
00BF 33DB                 280                      XOR     BX,BX                   ;BX = 0
00C1 B84800               281                      MOV     AX,LEVELM4              ;SET INTERRUPT MASTER LEVEL FOUR
00C4 50                   282                      PUSH    AX                      ;
00C5 53                   283                      PUSH    BX                      ;0 = NOT AN INTERRUPT TASK
00C6 0E                   284                      PUSH    CS                      ;CODE SEGMENT
00C7 52                   285                      PUSH    DX                      ;FOR SET INTERRUPT
00C8 53                   286                      PUSH    BX                      ;0 = NO DATA SEGMENT
00C9 1E                   287                      PUSH    DS                      ;DATA SEGMENT
00CA 8D062A00       R     288                      LEA     AX,STATUS               ;ADDRESS FOR STATUS RETURNED
00CE 50                   289                      PUSH    AX                      ;
00CF 9A0000----     E     290                      CALL    RQSETINTERRUPT          ;****RMX SET INTERRUPT****
00D4 32C0                 291                      XOR     AL,AL                   ;CLEAR VARIABLE STORE
00D6 B91000               292                      MOV     CX,16                   ;16 BYTES TO CLEAR
00D9 8D3E2A00       R     293                      LEA     DI,STATUS               ;
00DD F3                   294                      REP     STOSB                   ;
00DE AA
                          295 +1   $EJECT
```

```
LOC  OBJ                  LINE     SOURCE

00DF E6A2                  296                 OUT     STATPRT,AL              ;RESET D-A
00E1 E4A2                  297     DAINIT:     IN      AL,STATPRT              ;GET D-A STATUS WORD
00E3 2404                  298                 AND     AL,BIT2MASK             ;CHECK BIT 2
00E5 74FA                  299                 JZ      DAINIT                  ;WAIT UNTIL READY
00E7 B008                  300                 MOV     AL,DAWORD               ;LOAD D-A INIT WORD
00E9 E6A0                  301                 OUT     CRRNTPRT,AL             ;INITIALISE D-A
00EB 8B3E7C00        R     302                 MOV     DI,PATH_INDEX           ;LOAD UP PATH POINTER
00EF 8A266300        R     303                 MOV     AH,t_pt                 ;AH = PULSE TIME
00F3 88267B00        R     304                 MOV     p_pres,AH               ;PRESENT CURRENT=I_p
00F7 8B164E00        R     305                 MOV     DX,FLAGS                ;DX=ROUTINE FLAGS FOR CURRENT
00FB 8D1E3A00        R     306                 LEA     BX,BRANCH_TABLE         ;BX=JUMP TABLE 1ST ADDRESS
00FF B070                  307                 MOV     AL,MODEWORD             ;PUT TIMER 1 ON 86/30
0101 E6D6                  308                 OUT     TMRCNTRL,AL             ;IN MODE 0
0103 B8B75F                309                 MOV     AX,TIME10mS             ;LOAD 10mS COUNT
0106 E6D2                  310                 OUT     TMRINPUT,AL             ;OUTPUT LSB
0108 8AC4                  311                 MOV     AL,AH                   ;
010A E6D2                  312                 OUT     TMRINPUT,AL             ;OUTPUT MSB
010C B009                  313                 MOV     AL,GENABLE              ;8253 GATE PORT CONTROL WORD
010E E6C9                  314                 OUT     GATEPORT,AL             ;COUNTER ENABLED
0110 F4                    315     INT_LOOP:   HLT                             ;INTERRUPT WAIT
0111 F6062E0001      R     316                 TEST    WFINISH,01H             ;FINISHED FLAG SET ?
0116 74F8                  317                 JZ      INT_LOOP                ;NO.......
0118 B001                  318                 MOV     AL,1                    ;COUNTER1 GATE OFF
011A E6C9                  319                 OUT     GATEPORT,AL             ;COUNTER DISABLED
011C B84B00                320                 MOV     AX,LEVELM4              ;INTERRUPT MASTER LEVEL 4
011F 50                    321                 PUSH    AX                      ;
0120 1E                    322                 PUSH    DS                      ;DATA SEGMENT
0121 8D062A00        R     323                 LEA     AX,STATUS               ;STATUS ADDRESS
0125 50                    324                 PUSH    AX                      ;
0126 9A0000----      E     325                 CALL    RQRESETINTERRUPT        ;****RMX RESET INTERRUPT****
012B 07                    326                 POP     ES                      ;OLD EXTRA-SEGMENT
012C 1F                    327                 POP     DS                      ;RESTORE OLD DATA SEGMENT
012D CB                    328                 RET                             ;RETURN TO PLM86
                           329     ASMWELD     ENDP
                           330
                           331 +1  $EJECT
```

```
LOC  OBJ                  LINE     SOURCE

                          332
012E                      333     SETSIGN         PROC
                          334                     ; This routine scans the present offsets and if they are not
                          335                     ; positive then the corresponding direction flag for the motor
                          336                     ; driver is set and the value is then made positive.
012E C6062F0000      R    337                     MOV     DIRFLAGS,0              ;CLEAR ALL SIGN FLAGS.
0133 A11200          R    338                     MOV     AX,XTOTAL               ;GET XTOTAL     *X DIRECTION*
0136 23C0                 339                     AND     AX,AX                   ;SET FLAGS
0138 790A                 340                     JNS     YSIGN                   ;JUMP IF POSITIVE
013A 800E2F0001      R    341                     OR      DIRFLAGS,1              ;
013F F7D8                 342                     NEG     AX                      ;ABS(XTOTAL)
0141 A31200          R    343                     MOV     XTOTAL,AX               ;STORE XTOTAL
0144 A11400          R    344     YSIGN:          MOV     AX,YTOTAL               ;GET YTOTAL     *Y DIRECTION*
0147 23C0                 345                     AND     AX,AX                   ;SET FLAGS
0149 790A                 346                     JNS     ZSIGN                   ;JUMP IF POSITIVE
014B 800E2F0002      R    347                     OR      DIRFLAGS,2              ;
0150 F7D8                 348                     NEG     AX                      ;ABS(YTOTAL)
0152 A31400          R    349                     MOV     YTOTAL,AX               ;STORE YTOTAL
0155 A11600          R    350     ZSIGN:          MOV     AX,ZTOTAL               ;GET ZTOTAL     *Z DIRECTION*
0158 23C0                 351                     AND     AX,AX                   ;SET FLAGS
015A 790A                 352                     JNS     ROSIGN                  ;JUMP IF POSITIVE
015C 800E2F0004      R    353                     OR      DIRFLAGS,4              ;
0161 F7D8                 354                     NEG     AX                      ;ABS(ZTOTAL)
0163 A31600          R    355                     MOV     ZTOTAL,AX               ;STORE ZTOTAL
                          356
0166 A11800          R    357     ROSIGN:         MOV     AX,ROTOTAL              ;GET ROTOTAL    *R0 DIRECTION*
0169 23C0                 358                     AND     AX,AX                   ;SET FLAGS
016B 790A                 359                     JNS     R1SIGN                  ;JUMP IF POSITIVE
016D 800E2F0008      R    360                     OR      DIRFLAGS,8              ;
0172 F7D8                 361                     NEG     AX                      ;ABS(ROTOTAL)
0174 A31800          R    362                     MOV     ROTOTAL,AX              ;STORE ROTOTAL
0177 A11A00          R    363     R1SIGN:         MOV     AX,R1TOTAL              ;GET R1TOTAL    *R1 DIRECTION*
017A 23C0                 364                     AND     AX,AX                   ;SET FLAGS
017C 790A                 365                     JNS     R2SIGN                  ;JUMP IF POSITIVE
017E 800E2F0010      R    366                     OR      DIRFLAGS,10H            ;
0183 F7D8                 367                     NEG     AX                      ;ABS(R1TOTAL)
0185 A31A00          R    368                     MOV     R1TOTAL,AX              ;STORE R1TOTAL
0188 A11C00          R    369     R2SIGN:         MOV     AX,R2TOTAL              ;GET R2TOTAL    *R2 DIRECTION*
018B 23C0                 370                     AND     AX,AX                   ;SET FLAGS
018D 790A                 371                     JNS     SSEXIT                  ;JUMP IF POSITIVE
018F 800E2F0020      R    372                     OR      DIRFLAGS,20H            ;
0194 F7D8                 373                     NEG     AX                      ;ABS(R2TOTAL)
0196 A31C00          R    374                     MOV     R2TOTAL,AX              ;STORE R2TOTAL
0199 C3                   375     SSEXIT:         RET                             ;RETURN WITH DIR FLAGS SET
                          376     SETSIGN         ENDP
                          377
                          378 +1  $EJECT
```

```
LOC  OBJ                  LINE     SOURCE

                           379
019A                       380     SETENABLE       PROC
                           381                     ; This routine checks all the present offsets and sets
                           382                     ; the corresponding timer enable flag if the values are
                           383                     ; non zero. The rotation routines use the result to see
                           384                     ; if the rotation has finished as well as for the enables.
019A 32C9                  385                     XOR     CL,CL                   ;CLEAR CL FOR ENABLES
019C F7061C00FFFF    R     386                     TEST    R2TOTAL,0FFFFH          ;IS THE R2 OFFSET ZERO ?
01A2 7401                  387                     JZ      SEL1                    ;NO.....
01A4 F9                    388                     STC                             ;YES..... SET CARRY
01A5 D0D1                  389     SEL1:           RCL     CL,1                    ;CL=XXXXXXXE
01A7 F7061A00FFFF    R     390                     TEST    R1TOTAL,0FFFFH          ;IS THE R1 OFFSET ZERO ?
01AD 7401                  391                     JZ      SEL2                    ;NO.....
01AF F9                    392                     STC                             ;YES..... SET CARRY
01B0 D0D1                  393     SEL2:           RCL     CL,1                    ;CL=XXXXXXEE
01B2 F7061800FFFF    R     394                     TEST    R0TOTAL,0FFFFH          ;IS THE R0 OFFSET ZERO ?
01B8 7401                  395                     JZ      SEL3                    ;NO.....
01BA F9                    396                     STC                             ;YES..... FLAG IT
01BB D0D1                  397     SEL3:           RCL     CL,1                    ;MOVE FLAG IN.
                           398
01BD F7061600FFFF    R     399                     TEST    ZTOTAL,0FFFFH           ;IS THE Z OFFSET ZERO ?
01C3 7401                  400                     JZ      SEL4                    ;NO.....
01C5 F9                    401                     STC                             ;YES..... FLAG IT
01C6 D0D1                  402     SEL4:           RCL     CL,1                    ;MOVE FLAG IN.
01C8 F7061400FFFF    R     403                     TEST    YTOTAL,0FFFFH           ;IS THE Y OFFSET ZERO ?
01CE 7401                  404                     JZ      SEL5                    ;NO.....
01D0 F9                    405                     STC                             ;YES..... FLAG IT
01D1 D0D1                  406     SEL5:           RCL     CL,1                    ;MOVE FLAG IN.
01D3 F7061200FFFF    R     407                     TEST    XTOTAL,0FFFFH           ;IS THE X OFFSET ZERO ?
01D9 7401                  408                     JZ      SEL6                    ;NO.....
01DB F9                    409                     STC                             ;YES..... FLAG IT
01DC D0D1                  410     SEL6:           RCL     CL,1                    ;MOVE FLAG IN.
01DE 880E3000        R     411                     MOV     ENBFLAGS,CL             ;SAVE ENABLE FLAGS
01E2 C3                    412                     RET                             ;CL=XXXXXEEE
                           413     SETENABLE       ENDP
                           414
                           415 +1  $EJECT
```

```
LOC  OBJ                  LINE     SOURCE

                          416
                          417            ;*************************************************************************
                          418            ;*                                                                       *
                          419            ;*              INTERRUPT ENTRANCE AT CS:WELD_INT LABEL FAR              *
                          420            ;*                                                                       *
                          421            ;*************************************************************************
                          422
01E3                      423     WELD_INT        LABEL   FAR                     ;INTERRUPT SERVICE ENTRANCE
                          424                     PUBLIC  WELD_INT                ;
01E3 B8B75F               425                     MOV     AX,TIME10mS             ;RELOAD 10mS TIME INTERVAL
01E6 E6D2                 426                     OUT     TMRINPUT,AL             ;LSB
01E8 8AC4                 427                     MOV     AL,AH                   ;
01EA E6D2                 428                     OUT     TMRINPUT,AL             ;MSB
01EC E8F701               429                     CALL    WELD_MOVE               ;
01EF F6063200FF      R    430                     TEST    CURRENT,0FFH            ;CURRENT ON? (=1 IF YES)
01F4 7403                 431                     JZ      I_OFF                   ;NO.........CURRENT OFF.
01F6 E83102               432                     CALL    I_HANDLER               ;**** CURRENT SERVICE ****
01F9 F6063200FF      R    433     I_OFF:          TEST    CURRENT,0FFH            ;IF THE CURRENT IS OFF AND
01FE 750F                 434                     JNZ     NOT_FIN                 ;THE TORCH IS STATIONARY THEN
0200 F6063100FF      R    435                     TEST    WELD_STOP,0FFH          ;WE HAVE FINISHED A WELD AND
0205 7408                 436                     JZ      NOT_FIN                 ;WE MUST RETURN TO PL/M FOR
0207 893E7C00        R    437                     MOV     PATH_INDEX,DI           ;THE NEXT SET OF PARAMETERS.
020B FE062E00        R    438                     INC     WFINISH                 ;SET FINISHED FLAG.
020F B84800               439     NOT_FIN:        MOV     AX,LEVELM4              ;MASTER INTERRUPT LEVEL 4 (RL)
0212 56                   440                     PUSH    SI                      ;
0213 57                   441                     PUSH    DI                      ;
0214 52                   442                     PUSH    DX                      ;
0215 53                   443                     PUSH    BX                      ;
0216 50                   444                     PUSH    AX                      ;
0217 1E                   445                     PUSH    DS                      ;
0218 B82A00          R    446                     MOV     AX,OFFSET STATUS        ;
021B 50                   447                     PUSH    AX                      ;
021C 9A0000----      E    448                     CALL    RQEXITINTERRUPT         ;****SEND INTERRUPT ACKNOWLEDGE***
0221 5B                   449                     POP     BX                      ;
0222 5A                   450                     POP     DX                      ;
0223 5F                   451                     POP     DI                      ;
0224 5E                   452                     POP     SI                      ;
0225 CF                   453                     IRET                            ;
                          454
                          455 +1  $EJECT
```

```
LOC  OBJ                  LINE     SOURCE

                          456
                          457
0226                      458      START_ROT       LABEL   NEAR
0226 FE063700       R     459                      INC     ROT_ON                  ;INDICATION ROTATION IS
022A 8B850000       E     460                      MOV     AX,R0ARRAY[DI]          ;NOW MOVING AND LOAD ROTATION
022E A31800         R     461                      MOV     R0TOTAL,AX              ;OFFSETS.
0231 8B850000       E     462                      MOV     AX,R1ARRAY[DI]
0235 A31A00         R     463                      MOV     R1TOTAL,AX
0238 8B850000       E     464                      MOV     AX,R2ARRAY[DI]
023C A31C00         R     465                      MOV     R2TOTAL,AX
023F E8ECFE               466                      CALL    SETSIGN                 ;SET DIRECTION FLAG
0242                      467      ROTATION        PROC
0242 E855FF               468                      CALL    SETENABLE               ;SET ENABLES
0245 8026300038     R     469                      AND     ENBFLAGS,ROTONLY        ;CLEAR OUT X,Y & Z ENABLES
024A 56                   470                      PUSH    SI                      ;INCASE OF RUBBISH IN TOTALS
024B BE0800               471                      MOV     SI,8
024E 833E180008     R     472                      CMP     R0TOTAL,8               ;CAN WE DO A FULL EIGHT STEPS
0253 7704                 473                      JA      R0MAX                   ;YES.......
0255 8B361800       R     474                      MOV     SI,R0TOTAL              ;NO..... LOAD REMAINDER (<8)
0259 29361800       R     475      R0MAX:          SUB     R0TOTAL,SI              ;REMOVE STEPS TO BE DONE
025D D1E6                 476                      SHL     SI,1                    ;LOAD THE WORD COUNT FROM
025F 8B840000       R     477                      MOV     AX,SPEEDS[SI]           ;THE SPEED LOOKUP TABLE.
0263 BA1004               478                      MOV     DX,TIMER1               ;LOAD THE TIMER.
0266 EE                   479                      OUT     DX,AL
0267 86C4                 480                      XCHG    AL,AH
0269 EE                   481                      OUT     DX,AL
                          482 +1   $EJECT
```

```
LOC  OBJ                  LINE     SOURCE

026A BE0800                483                      MOV     SI,8                    ;REPEAT ABOVE OPERATION FOR
026D 833E1A0008      R     484                      CMP     R1TOTAL,8               ;THE R0 AND R1 AXES.
0272 7704                  485                      JA      R1MAX
0274 8B361A00        R     486                      MOV     SI,R1TOTAL
0278 29361A00        R     487      R1MAX:          SUB     R1TOTAL,SI
027C D1E6                  488                      SHL     SI,1
027E 8B840000        R     489                      MOV     AX,SPEEDS[SI]
0282 BA1204                490                      MOV     DX,TIMER2
0285 EE                    491                      OUT     DX,AL
0286 86C4                  492                      XCHG    AL,AH
0288 EE                    493                      OUT     DX,AL
0289 BE0800                494                      MOV     SI,8
028C 833E1C0008      R     495                      CMP     R2TOTAL,8
0291 7704                  496                      JA      R2MAX
0293 8B361C00        R     497                      MOV     SI,R2TOTAL
0297 29361C00        R     498      R2MAX:          SUB     R2TOTAL,SI
029B D1E6                  499                      SHL     SI,1
029D 8B840000        R     500                      MOV     AX,SPEEDS[SI]
02A1 BA1404                501                      MOV     DX,TIMER3
02A4 EE                    502                      OUT     DX,AL
02A5 86C4                  503                      XCHG    AL,AH
02A7 EE                    504                      OUT     DX,AL
02A8 5E                    505                      POP     SI
02A9 E82801                506                      CALL    START_MOTORS            ;SET THE MOTORS OFF.
02AC 803E300000      R     507                      CMP     ENBFLAGS,0              ;HAVE WE FINISHED THE ROTATIONS
02B1 750C                  508                      JNZ     MORE_ROTS               ;NO.........
02B3 C606370000      R     509                      MOV     ROT_ON,0                ;TURN ROTATION FLAG OFF.
02B8 C606340000      R     510                      MOV     MATH_DONE,0             ;NEED LINEMATHS DOING FOR THE
02BD 47                    511                      INC     DI                      ;NEXT LINE ON THE PATH.
02BE 47                    512                      INC     DI
02BF C3                    513      MORE_ROTS:      RET
                           514      ROTATION        ENDP
                           515
                           516 +1   $EJECT
```

```
LOC  OBJ                  LINE     SOURCE

                          517
02C0                      518      WELDMATH        PROC
02C0 8B850000       E     519      NEXT_OFS:       MOV     AX,ZARRAY [DI]          ;GET NEXT Z POINT
02C4 3D0080               520                      CMP     AX,8000H                ;IS IT A CONTROL?
02C7 7532                 521                      JNZ     REST_PTS                ;NO......
02C9 47                   522                      INC     DI
02CA 47                   523                      INC     DI
02CB 83BDFEFF04     E     524                      CMP     XARRAY[DI-2],4          ;END OF PATH?
02D0 741C                 525                      JZ      PEND                    ;YES..... PATH END FOUND
02D2 83BDFEFF00     E     526                      CMP     XARRAY[DI-2],0          ;IS IT END OF WELD
02D7 7410                 527                      JZ      WEND                    ;YES......
02D9 83BDFEFF01     E     528                      CMP     XARRAY[DI-2],1          ;
02DE 75E0                 529                      JNZ     NEXT_OFS                ;LOOK FOR END OF WELD
02E0 FE063100       R     530                      INC     WELD_STOP               ;STOP MOVING UNTIL SLOPE
02E4 FE063200       R     531                      INC     CURRENT                 ;START CURRENT ROUTINE
02E8 C3                   532                      RET                             ;
02E9 FE063100       R     533      WEND:           INC     WELD_STOP               ;STOP THE MOTORS
02ED C3                   534                      RET
02EE FE062E00       R     535      PEND:           INC     WFINISH                 ;PATH FINISHED
02F2 FE067E00       R     536                      INC     PATH_FINISHED           ;TELL PLM END OF PATH FOUND
02F6 FE063100       R     537                      INC     WELD_STOP               ;STOP MOTORS
02FA C3                   538                      RET                             ;
02FB FE063400       R     539      REST_PTS:       INC     MATH_DONE               ;FLAG MATH IS DONE.
02FF 8B850000       E     540                      MOV     AX,XARRAY[DI]           ;CHECK TO SEE IF IT IS A
0303 0B850000       E     541                      OR      AX,YARRAY[DI]           ;ROTATION POINT SIGNIFIED BY
0307 0B850000       E     542                      OR      AX,ZARRAY[DI]           ;THE X, Y & Z POINTS BEING
030B 23C0                 543                      AND     AX,AX                   ;ZERO.
030D 7503                 544                      JNZ     XYZ_CALC                ;NOT ZERO SO DO XYZ MATHS.
030F E914FF               545                      JMP     START_ROT               ;DO ROTATION MATHS INSTEAD.
                          546 +1   $EJECT
```

```
LOC  OBJ                  LINE     SOURCE

0312 8B850000       E      547     XYZ_CALC:       MOV     AX,ZARRAY[DI]
0316 A31600         R      548                     MOV     ZTOTAL,AX               ;LOAD ALL THE OFFSETS ONTO
0319 9BDF061600     R      549                     FILD    WORD PTR ZTOTAL         ;THE 8087 AND TO THE TOTALS
031E 8B850000       E      550                     MOV     AX,YARRAY [DI]          ;
0322 9BDCC8                551                     FMUL    ST,ST                   ;SQUARE THE OFFSETS
0325 A31400         R      552                     MOV     YTOTAL,AX               ;
0328 9BDF061400     R      553                     FILD    WORD PTR YTOTAL         ;
032D 8B850000       E      554                     MOV     AX,XARRAY [DI]          ;
0331 9BDCC8                555                     FMUL    ST,ST                   ;
0334 A31200         R      556                     MOV     XTOTAL,AX               ;
0337 9BDF061200     R      557                     FILD    WORD PTR XTOTAL         ;
033C 9BDCC8                558                     FMUL    ST,ST                   ;
033F 9BDEC1                559                     FADDP   ST(1),ST                ;ST=(X*X+Y*Y)
0342 9BDEC1                560                     FADDP   ST(1),ST                ;ST=(X*X+Y*Y+Z*)
0345 9BD9FA                561                     FSQRT                           ;ST=LENGTH OF STRAIGHT LINE.
0348 9BD8367000     R      562                     FDIV    DWORD PTR Velocity      ;TIME = ST
034D E8DEFD                563                     CALL    SETSIGN                 ;SET SIGN FLAGS
0350 9BDDD1                564                     FST     ST(1)                   ;ST=ST(1)
0353 9BDF1E2C00     R      565                     FISTP   WORD PTR TIMCNT         ;SAVE NUMBER OF 10mS PULSES
0358 9BDE0E3800     R      566                     FIMUL   WORD PTR FREQ           ;MULTIPLY BY CLOCK FREQUENCY
035D E83AFE                567,                    CALL    SETENABLE               ;SET ENABLE BITS.
0360 8026300007     R      568                     AND     ENBFLAGS,XYZONLY        ;MAY BE RUBBISH IN THE ROTS
0365 9BDDD1                569                     FST     ST(1)                   ;ST(1)=ST
0368 833E120000     R      570                     CMP     XTOTAL,0
036D 7405                  571                     JZ      X0                      ;AVOID DIVIDE BY ZERO
036F 9BDE361200     R      572                     FIDIV   WORD PTR XTOTAL         ;RESULT=XCOUNT+1
0374 9BDF1E1E00     R      573     X0:             FISTP   WORD PTR XSPEED         ;STORE X COUNT
                           574 +1  $EJECT
```

```
LOC  OBJ                  LINE     SOURCE

0379 9BDDD1               575                    FST     ST(1)                   ;ST(1)=ST
037C 833E140000      R    576                    CMP     YTOTAL,0
0381 7405                 577                    JZ      Y0                      ;AVOID THE DIVIDE BY ZERO
0383 9BDE361400      R    578                    FIDIV   WORD PTR YTOTAL         ;RESULT=YCOUNT+1
0388 9BDF1E2000      R    579    Y0:             FISTP   WORD PTR YSPEED         ;STORE Y COUNT
038D 9BDDD1               580                    FST     ST(1)                   ;
0390 833E160000      R    581                    CMP     ZTOTAL,0
0395 7405                 582                    JZ      Z0                      ;AVOID THE DIVIDE BY ZERO
0397 9BDE361600      R    583                    FIDIV   WORD PTR ZTOTAL         ;RESULT=ZCOUNT+1
039C 9BDF1E2200      R    584    Z0:             FISTP   WORD PTR ZSPEED         ;STORE Z COUNT
                          585
03A1 FF0E1E00        R    586                    DEC     XSPEED                  ;WAS C+1 NOW C
03A5 FF0E2000        R    587                    DEC     YSPEED                  ;
03A9 9B                   588                    FWAIT                           ;WAIT FOR ZSPEED
03AA FF0E2200        R    589                    DEC     ZSPEED
03AE 52                   590                    PUSH    DX                      ;SAVE CURRENT FLAGS
                          591
03AF A11E00          R    592                    MOV     AX,XSPEED               ;GET X SPEED
03B2 BA1804               593                    MOV     DX,TIMER4               ;LOAD TIMER BASE ADDRESS
03B5 EE                   594                    OUT     DX,AL                   ;LOW BYTE TO ALL 3 TIMERS
03B6 86C4                 595                    XCHG    AL,AH                   ;SWOP LSB & MSB
03B8 EE                   596                    OUT     DX,AL                   ;OUT MSB
03B9 A12000          R    597                    MOV     AX,YSPEED               ;GET Y SPEED
03BC BA1A04               598                    MOV     DX,TIMER5               ;
03BF EE                   599                    OUT     DX,AL                   ;
03C0 86C4                 600                    XCHG    AL,AH                   ;SWOP LSB & MSB
03C2 EE                   601                    OUT     DX,AL                   ;OUT MSB
03C3 A12200          R    602                    MOV     AX,ZSPEED               ;GET Y SPEED
03C6 BA1C04               603                    MOV     DX,TIMER6               ;
03C9 EE                   604                    OUT     DX,AL                   ;
03CA 86C4                 605                    XCHG    AL,AH                   ;SWOP LSB & MSB
03CC EE                   606                    OUT     DX,AL                   ;OUT MSB
03CD E80400               607                    CALL    START_MOTORS
03D0 47                   608                    INC     DI                      ;MOVE TO NEXT POINT
03D1 47                   609                    INC     DI                      ;
03D2 5A                   610                    POP     DX                      ;RECOVER CURRENT FLAGS
03D3 C3                   611                    RET
                          612    WELDMATH        ENDP
                          613
                          614 +1 $EJECT
```

```
LOC  OBJ                  LINE     SOURCE

                           615
03D4                       616     START_MOTORS    PROC
                           617                     ; This routine simple outputs the direction bits to the
                           618                     ; drivers and the enable flags to the timer gates.
03D4 8A263000        R     619                     MOV     AH,ENBFLAGS             ;GET ENABLE FLAGS
03D8 A02F00          R     620                     MOV     AL,DIRFLAGS             ;GET SIGN FLAGS
03DB BA0004                621                     MOV     DX,DIRPRTADDR           ;GET MOTOR ADDRESSES
03DE EE                    622                     OUT     DX,AL                   ;START MOTORS.
03DF BA0804                623                     MOV     DX,ENBPRTADDR
03E2 8AC4                  624                     MOV     AL,AH
03E4 EE                    625                     OUT     DX,AL
03E5 C3                    626                     RET
                           627     START_MOTORS    ENDP
                           628
                           629 +1  $EJECT
```

```
LOC  OBJ                  LINE     SOURCE

                          630
03E6                      631      WELD_MOVE       PROC
                          632                      ; This routine is resonsible for the movement of the
                          633                      ; robot. It checks the weld stop flag to see if the robot
                          634                      ; is required to be stationary by the current and also
                          635                      ; checks the maths flag to see if the maths needs doing
                          636                      ; as a result of the last points being completed. The maths
                          637                      ; routine may also set the weld stop flag as a result of
                          638                      ; finding a weld start flag so again this must be checked.
                          639                      ; The routine then decides if the rotation or the XYZ is
                          640                      ; running. In the case of rotation the rotation routine is
                          641                      ; called, in the case of XYZ the time is simply decremented
                          642                      ; and if the time is up the maths flag is cleared.
03E6 52                   643                      PUSH    DX                      ;SAVE CURRENT FLAGS
03E7 BA0804               644                      MOV     DX,ENBPRTADDR           ;GET PORT ADDRESS
03EA F6063100FF      R    645                      TEST    WELD_STOP,0FFH          ;STATIONARY? 1 = YES
03EF 7534                 646                      JNZ     STOPPED                 ;CURRENT WANTS TORCH TO STOP ?
03F1 F6063400FF      R    647                      TEST    MATH_DONE,0FFH          ;MATH DONE (0 = NO)
03F6 750A                 648                      JNZ     MATHOK                  ;MATHS DONE.
03F8 E8C5FE               649                      CALL    WELDMATH                ;DO MATHS.
03FB F6063100FF      R    650                      TEST    WELD_STOP,0FFH          ;DOES THE MATHS WANT THE
0400 7523                 651                      JNZ     STOPPED                 ;TORCH STOPPED.
0402 A02F00          R    652      MATHOK:         MOV     AL,DIRFLAGS
0405 52                   653                      PUSH    DX
0406 BA0004               654                      MOV     DX,DIRPRTADDR           ;
0409 EE                   655                      OUT     DX,AL                   ;OUTPUT DIRECTION FLAGS
040A 5A                   656                      POP     DX
040B F6063700FF      R    657                      TEST    ROT_ON,0FFH             ;IS THE ROTATION ON ?
0410 7405                 658                      JZ      XYZ_MOT                 ;NO.............. XYZ
0412 E82DFE               659                      CALL    ROTATION                ;UPDATE THE ROTATION
0415 EB11                 660                      JMP     SHORT ROT_EXIT          ;
0417 A03000          R    661      XYZ_MOT:        MOV     AL,ENBFLAGS             ;SET TORCH MOVING.
041A FF0E2C00        R    662                      DEC     TIMCNT                  ;COUNT TIME DOWN
041E 7507                 663                      JNZ     MOVING                  ;STILL MORE TO GO
0420 C606340000      R    664                      MOV     MATH_DONE,0             ;MATHS NEEDS DOING
0425 32C0                 665      STOPPED:        XOR     AL,AL                   ;
0427 EE                   666      MOVING:         OUT     DX,AL                   ;
0428 5A                   667      ROT_EXIT:       POP     DX                      ;RECOVER CURRENT FLAGS
0429 C3                   668                      RET
                          669      WELD_MOVE       ENDP
                          670
                          671 +1   $EJECT
```

```
LOC  OBJ                  LINE     SOURCE

                          672
                          673              ;*************************************************************************
                          674              ;*
                          675              ;*           THIS ROUTINE HANDLES ALL THE CURRENT CONTROL
                          676              ;*
                          677              ;*************************************************************************
                          678
042A                      679      I_HANDLER       PROC
042A 0BD2                 680                      OR      DX,DX                   ;ARE ALL FLAGS DOWN?
042C 740B                 681                      JZ      CURRENT_END             ;YES? THEN FINISHED
042E D1E2                 682      NEXT:           SHL     DX,1                    ;IS NEXT FLAG SET
0430 7302                 683                      JNC     NOT_YET                 ;FLAG NOT SET?
0432 FF27                 684                      JMP     WORD PTR [BX]           ;FLAG SET,GO TO ROUTINE
0434 83C302               685      NOT_YET:        ADD     BX,TYPE BRANCH_TABLE    ;LOAD NEXT ADDRESS
0437 EBF5                 686                      JMP     SHORT NEXT              ;GO TEST NEXT FLAG
0439 33C0                 687      CURRENT_END:    XOR     AX,AX                   ;CLEAR THE GATE LINE TO LOW
043B E6A2                 688                      OUT     STATPRT,AL              ;CLEAR THE D-A WITH RESET
043D A27B00          R    689                      MOV     PORT1STR,AL
0440 A23200          R    690                      MOV     CURRENT,AL              ;ALLOW MAIN PROG OUT OF
0443 C3                   691                      RET                             ;THE WAIT LOOP ON RETURN
                          692
                          693 +1   $EJECT
```

```
LOC  OBJ                  LINE     SOURCE

                          694
0444                      695     PULSE           PROC
0444 F6067A0002     R     696                     TEST    PULSE_FLAGS,BIT1MASK    ;IS PULSING REQUIRED?
0449 7472                 697                     JZ      NO_PULSE                ;NO THEN OUT PRESENT I
044B A16B00         R     698                     MOV     AX,PRES_I               ;IS PRESENT I> BACK
044E 3B066C00       R     699                     CMP     AX,I_bw                 ;YES THEN PULSING ALLOWED
0452 7669                 700                     JBE     NO_PULSE                ;
0454 F6067A0001     R     701                     TEST    PULSE_FLAGS,BITOMASK    ;TEST FOR PULSE OR BACK
0459 7445                 702                     JZ      BACK                    ;
                          703                                                     ;
045B F606790001     R     704                     TEST    pen_monf,1              ;IS PENETRATION MONITOR ON?
0460 740B                 705                     JZ      NO_PENETRATION          ;NO.......
0462 E4C8                 706                     IN      AL,KEYCOL               ;
0464 2450                 707                     AND     AL,80                   ;IS BIT SET
0466 7405                 708                     JZ      NO_PENETRATION          ;NO.......
0468 C606780001     R     709                     MOV     p_pres,1                ;BIT SET THEN PULSE OVER
046D FE0E7800       R     710     NO_PENETRATION: DEC     p_pres                  ;DEC PULSE TIME COUNT
0471 752C                 711                     JNZ     P_RET                   ;COUNT NOT ZERO? RETURN
0473 8A266200       R     712                     MOV     AH,t_bt                 ;ZERO? 1.LOAD BACK COUNT
0477 88267800       R     713                     MOV     p_pres,AH               ;
047B 80367A0001     R     714                     XOR     PULSE_FLAGS,1           ;       2. FLIP FLAG
0480 A07B00         R     715                     MOV     AL,PORT1STR             ;       3. TURN WIRE FEED OFF
0483 0C40                 716                     OR      AL,01000000B            ;          PORT C8
0485 A27B00         R     717                     MOV     PORT1STR,AL             ;          BIT 6
0488 E6CA                 718                     OUT     SWTCHPRT,AL             ;          WIRE FEED OFF
048A E4A2                 719     WAIT1:          IN      AL,STATPRT              ;CHECK THE D-A IS READY
048C 2402                 720                     AND     AL,DAMASK               ;
048E 75FA                 721                     JNZ     WAIT1                   ;
0490 A16C00         R     722                     MOV     AX,I_bw                 ;GET BACK CURRENT
0493 E6A0                 723                     OUT     CRRNTPRT,AL             ;
0495 E4A2                 724     WAIT2:          IN      AL,STATPRT              ;CHECK D-A IS READY
0497 2402                 725                     AND     AL,DAMASK               ;
0499 75FA                 726                     JNZ     WAIT2                   ;
049B 8AC4                 727                     MOV     AL,AH                   ;
049D E6A0                 728                     OUT     CRRNTPRT,AL             ;OUTPUT THE HIGH BYTE
049F C3                   729     P_RET:          RET                             ;RETURN
                          730 +1  $EJECT                                          ;
```

```
LOC  OBJ                  LINE     SOURCE

04A0 FE0E7800        R     731     BACK:           DEC     P_Pres                  ;DEC BACK TIME COUNT
04A4 75F9                  732                     JNZ     P_RET                   ;COUNT NOT ZERO? RETURN
04A6 8A266300        R     733                     MOV     AH,t_pt                 ;ZERO?1.LOAD PULSE COUNT
04AA 88267800        R     734                     MOV     P_Pres,AH               ;
04AE 80367A0001      R     735                     XOR     PULSE_FLAGS,1           ;     2. FLIP FLAG
04B3 A07B00          R     736                     MOV     AL,PORT1STR             ;     3. TURN THE WIRE FEED ON
04B6 24BF                  737                     AND     AL,BIT6MASK             ;        PORT C8
04B8 A27B00          R     738                     MOV     PORT1STR,AL             ;        BIT 6
04BB E6CA                  739                     OUT     SWTCHPRT,AL             ;        WIRE FEED ON
04BD E4A2                  740     NO_PULSE:       IN      AL,STATPRT              ;CHECK D-A IS READY
04BF 2402                  741                     AND     AL,DAMASK               ;
04C1 75FA                  742                     JNZ     NO_PULSE                ;
04C3 A16800          R     743                     MOV     AX,PRES_I               ;GET CURRENT AND OUT LSB
04C6 E6A0                  744                     OUT     CRRNTPRT,AL             ;
04C8 E4A2                  745     WAIT3:          IN      AL,STATPRT              ;CHECK D-A IS READY
04CA 2402                  746                     AND     AL,DAMASK               ;
04CC 75FA                  747                     JNZ     WAIT3                   ;
04CE 8AC4                  748                     MOV     AL,AH                   ;
04D0 E6A0                  749                     OUT     CRRNTPRT,AL             ;OUTPUT THE HIGH BYTE
04D2 C3                    750                     RET                             ;RETURN
                           751     PULSE           ENDP
                           752
                           753 +1  $EJECT
```

```
LOC  OBJ                  LINE     SOURCE

                          754
04D3 F8                   755      GAS_ON:         CLC                             ;ONCE ONLY ROUTINE
04D4 D1DA                 756                      RCR     DX,1                    ;MOVE FLAGS BACK
04D6 A07B00          R    757                      MOV     AL,PORT1STR             ;TURN THE GAS ON
04D9 24DF                 758                      AND     AL,BIT5MASK             ;PORT C8
04DB A27B00          R    759                      MOV     PORT1STR,AL             ;BIT 5
04DE E6CA                 760                      OUT     SWTCHPRT,AL             ;GAS ON
04E0 C3                   761                      RET                             ;WAIT FOR NEXT INTRPT
                          762
04E1 D1DA                 763      G_COUNT:        RCR     DX,1                    ;MOVE FLAGS BACK
04E3 F7065800FF00    R    764                      TEST    t_g1t,0FFH              ;CHECK IF GAS1 COUNT IS
04E9 7405                 765                      JZ      G_END                   ;ZERO
04EB FF0E5800        R    766                      DEC     t_g1t                   ;DEC TIME IF NOT ZERO
04EF C3                   767                      RET                             ;WAIT FOR NEXT INTRPT
04F0 81E2FF7F             768      G_END:          AND     DX,FLAGMASK             ;CLEAR FLAG
04F4 E4A2                 769      WAIT4:          IN      AL,STATPRT              ;OUTPUT 1ST CURRENT C1
04F6 2402                 770                      AND     AL,DAMASK               ;CHECK THE D-A IS READY
04F8 75FA                 771                      JNZ     WAIT4                   ;
04FA A16A00          R    772                      MOV     AX,C1                   ;GET C1
04FD E6A0                 773                      OUT     CRRNTPRT,AL             ;OUT LSB
04FF E4A2                 774      WAIT5:          IN      AL,STATPRT              ;CHECK D-A IS READY
0501 2402                 775                      AND     AL,DAMASK               ;
0503 75FA                 776                      JNZ     WAIT5                   ;JUMP IF NOT
0505 8AC4                 777                      MOV     AL,AH                   ;
0507 E6A0                 778                      OUT     CRRNTPRT,AL             ;OUTPUT THE HIGH BYTE
0509 A07B00          R    779                      MOV     AL,PORT1STR             ;AND THEN STRIKE
050C 247F                 780                      AND     AL,BIT7MASK             ;THE ARC UP
050E A27B00          R    781                      MOV     PORT1STR,AL             ;
0511 E6CA                 782                      OUT     SWTCHPRT,AL             ;ARC STRUCK
0513 C3                   783                      RET                             ;WAIT FOR NEXT INTRPT
                          784
                          785 +1   $EJECT
```

```
LOC  OBJ                  LINE     SOURCE

                          786
0514 D1DA                 787      TS1_COUNT:      RCR     DX,1                    ;MOVE FLAGS BACK
0516 FF0E5400        R    788                      DEC     t_s1t                   ;DEC ts1 TIME
051A 7504                 789                      JNZ     TS1_RET                 ;FINISHED?
051C 81E2FF7F             790                      AND     DX,FLAGMASK             ;CLEAR FLAG
0520 C3                   791      TS1_RET:        RET                             ;WAIT FOR NEXT INTRPT
                          792
0521 D1DA                 793      TSU_COUNT:      RCR     DX,1                    ;MOVE FLAGS BACK
0523 C606310000      R    794                      MOV     WELD_STOP,0             ;START MOVING.
0528 FF0E5600        R    795                      DEC     t_sut                   ;DEC t_su TIME
052C 741C                 796                      JZ      TSU_END                 ;FINISHED?
052E A16400          R    797                      MOV     AX,I_uw                 ;GET THE CURRENT INC
0531 01066800        R    798                      ADD     PRES_I,AX               ;ADD INC TO PRESENT I
0535 7209                 799                      JC      I_LOOP                  ;
0537 A16E00          R    800                      MOV     AX,I_pw                 ;THEN CHECK THAT PRES_I
053A 39066800        R    801                      CMP     PRES_I,AX               ;IS NOT GREATER THAN I_pw
053E 7206                 802                      JB      PRESSMALL               ;BECAUSE OF MATH ROUNDING
0540 A16E00          R    803      I_LOOP:         MOV     AX,I_pw                 ;THEN CHECK THAT PRES_I
0543 A36800          R    804                      MOV     PRES_I,AX               ;ERRORS
0546 E8F8FE               805      PRESSMALL:      CALL    PULSE                   ;OUTPUT CURRENT
0549 C3                   806                      RET                             ;WAIT FOR NEXT INTRPT
054A A16E00          R    807      TSU_END:        MOV     AX,I_pw                 ;LOAD I_p TO OFFSET ERRORS
054D A36800          R    808                      MOV     PRES_I,AX               ;WHEN I_psu IS LARGE
0550 81E2FF7F             809                      AND     DX,FLAGMASK             ;CLEAR FLAG
0554 C3                   810                      RET                             ;WAIT FOR NEXT INTRPT
                          811
0555 D1DA                 812      MAIN_COUNT:     RCR     DX,1                    ;MOVE FLAGS BACK
0557 C606310000      R    813                      MOV     WELD_STOP,0             ;START/CONTINUE MOVING.
055C FF0E5C00        R    814                      DEC     t_mt                    ;DEC MAIN PULSE TIME
0560 7404                 815                      JZ      MAIN_END                ;FINISHED?
0562 E8DFFE               816                      CALL    PULSE                   ;OUTPUT CURRENT
0565 C3                   817                      RET                             ;WAIT FOR NEXT INTRPT
0566 81E2FF7F             818      MAIN_END:       AND     DX,FLAGMASK             ;CLEAR FLAG
056A C3                   819                      RET                             ;WAIT FOR NEXT INTRPT
                          820
                          821 +1   $EJECT
```

```
LOC  OBJ                  LINE     SOURCE

                          822
056B D1DA                 823      TSD_COUNT:      RCR     DX,1                    ;MOVE FLAGS BACK
056D FF0E5E00        R    824                      DEC     t_sdt                   ;DEC t_sd TIME
0571 7417                 825                      JZ      TSD_END                 ;FINISHED?
0573 A16600          R    826                      MOV     AX,I_dw                 ;GET CURRENT DEC
0576 29066800        R    827                      SUB     PRES_I,AX               ;DEC PRESENT I
057A A15200          R    828                      MOV     AX,I_s2w                ;THEN CHECK THAT PRES_I
057D 39066800        R    829                      CMP     PRES_I,AX               ;IS NOT SMALLER THAN I_s2
0581 7703                 830                      JA      PRESGREATER             ;DUE TO MATH ROUNDING
0583 A36800          R    831                      MOV     PRES_I,AX               ;ERRORS
0586 E8BBFE               832      PRESGREATER:    CALL    PULSE                   ;OUTPUT CURRENT
0589 C3                   833                      RET                             ;WAIT FOR NEXT INTRPT
058A 81E2FF7F             834      TSD_END:        AND     DX,FLAGMASK             ;CLEAR FLAG
058E C3                   835                      RET                             ;WAIT FOR NEXT INTRPT
                          836
058F F8                   837      IS2_OUT:        CLC
0590 D1DA                 838                      RCR     DX,1                    ;MOVE FLAGS BACK
0592 FE063100        R    839                      INC     WELD_STOP               ;STOP MOVING.
                          840                                                      ;CHANGE VELOCITY HERE
0596 E4A2                 841      WAIT6:          IN      AL,STATPRT              ;OUTPUT I_s2
0598 2402                 842                      AND     AL,DAMASK               ;CHECK THE D-A IS READY
059A 75FA                 843                      JNZ     WAIT6                   ;
059C A15200          R    844                      MOV     AX,I_s2w                ;GET I_s2
059F E6A0                 845                      OUT     CRRNTPRT,AL             ;
05A1 E4A2                 846      WAIT7:          IN      AL,STATPRT              ;CHECK D-A IS READY
05A3 2402                 847                      AND     AL,DAMASK               ;
05A5 75FA                 848                      JNZ     WAIT7                   ;
05A7 8AC4                 849                      MOV     AL,AH                   ;
05A9 E6A0                 850                      OUT     CRRNTPRT,AL             ;OUTPUT THE HIGH BYTE
05AB C3                   851                      RET                             ;WAIT FOR NEXT INTRPT
                          852
05AC D1DA                 853      TS2_COUNT:      RCR     DX,1                    ;COUNT OUT TS2
05AE FF0E6000        R    854                      DEC     t_s2t                   ;DEC t_s2 time
05B2 7504                 855                      JNZ     TS2_RET                 ;WAIT FOR NEXT INTRPT
05B4 81E2FF7F             856                      AND     DX,FLAGMASK             ;CLEAR FLAG
05B8 C3                   857      TS2_RET:        RET                             ;WAIT FOR NEXT INTRPT
                          858
                          859 +1   $EJECT
```

```
LOC  OBJ                  LINE     SOURCE

                          860
05B9 F8                   861      ARC_OUT:        CLC                             ;ONCE ONLY ROUTINE
05BA D1DA                 862                      RCR     DX,1                    ;MOVE FLAGS BACK
05BC A07B00          R    863                      MOV     AL,PORT1STR             ;EXTINGUISH THE
05BF 0C80                 864                      OR      AL,10000000B            ;ARC ON PORT C8
05C1 A27B00          R    865                      MOV     PORT1STR,AL             ;BIT 7
05C4 E6CA                 866                      OUT     SWTCHPRT,AL             ;ARC OFF
05C6 C3                   867                      RET                             ;WAIT FOR NEXT INTRPT
                          868
05C7 D1DA                 869      GAS_OFF:        RCR     DX,1                    ;COUNT OUT GAS
05C9 F7065A00FF00    R    870                      TEST    t_g2t,0FFH              ;POST-PURGE TIME
05CF 7405                 871                      JZ      GAS2_END                ;FINISHED
05D1 FF0E5A00        R    872                      DEC     t_g2t                   ;DEC t_g2 TIME
05D5 C3                   873                      RET                             ;WAIT FOR NEXT INTRPT
05D6 81E2FF7F             874      GAS2_END:       AND     DX,FLAGMASK             ;CLEAR FLAG
05DA A07B00          R    875                      MOV     AL,PORT1STR             ;TURN GAS OFF
05DD 0C20                 876                      OR      AL,00100000B            ;PORT C8
05DF A27B00          R    877                      MOV     PORT1STR,AL             ;BIT 5
05E2 E6CA                 878                      OUT     SWTCHPRT,AL             ;GAS OFF
05E4 C3                   879                      RET                             ;WAIT FOR NEXT INTRPT
                          880      I_HANDLER       ENDP
                          881
----                      882      CODE    ENDS
                          883
                          884      END

ASSEMBLY COMPLETE, NO ERRORS FOUND
```

7. DISK SUPERVISOR TASK.

```
iRMX 86 PL/M-86 V2.3 COMPILATION OF MODULE FLOPSEC
OBJECT MODULE PLACED IN DISK/SUP.OBJ
COMPILER INVOKED BY:  :LANG:PLM86 DISK/SUP.P86


              $TITLE ('FLOPPY DISK SUPERVISOR & SECURITY CHECKER VER 86/2.0')

              $DEBUG
              $LARGE
              $OPTIMIZE(3)

   1          FLOPSEC: DO;

              $INCLUDE(INC/LITERALS.P86)
          =   $SAVE NOLIST
              $INCLUDE(/INC/RMX86/NCRMBX.EXT)
          -   $SAVE NOLIST
              $INCLUDE(/INC/RMX86/NSNMES.EXT)
          =   $SAVE NOLIST
              $INCLUDE(/INC/RMX86/NRCMES.EXT)
          -   $SAVE NOLIST
              $INCLUDE(/INC/RMX86/NCRSEG.EXT)
          =   $SAVE NOLIST
              $INCLUDE(/INC/RMX86/NDLSEG.EXT)
          =   $SAVE NOLIST
              $INCLUDE(/INC/RMX86/NGTTYP.EXT)
          =   $SAVE NOLIST
              $INCLUDE(/INC/RMX86/NLUOBJ.EXT)
          -   $SAVE NOLIST
              $INCLUDE(/INC/RMX86/NCTOBJ.EXT)
          =   $SAVE NOLIST
              $INCLUDE(/INC/RMX86/NUCOBJ.EXT)
          =   $SAVE NOLIST
              $INCLUDE(/INC/RMX86/NRCUNI.EXT)
          =   $SAVE NOLIST
              $INCLUDE(/INC/RMX86/NSUTSK.EXT)
          =   $SAVE NOLIST
              $INCLUDE(/INC/RMX86/NRSTSK.EXT)
          =   $SAVE NOLIST
              $INCLUDE(/INC/RMX86/ICRUSR.EXT)
          =   $SAVE NOLIST
              $INCLUDE(/INC/RMX86/ISTUSR.EXT)
          =   $SAVE NOLIST
              $INCLUDE(/INC/RMX86/IADLCN.EXT)
          =   $SAVE NOLIST
              $INCLUDE(/INC/RMX86/IAATFL.EXT)
          =   $SAVE NOLIST
              $INCLUDE(/INC/RMX86/IPATDV.EXT)
          =   $SAVE NOLIST
              $INCLUDE(/INC/RMX86/IPDTDV.EXT)
          =   $SAVE NOLIST
```

```
151   1       declare (disk_mailbox,th_in_mbx,th_out_mbx,read_in,read_out)      token;
152   1       declare (disk_connection,main_task_token,disk_sem)                token;
153   1       declare (status,gen_word,dummy_mbx,type_code,done)                 word;
154   1       declare wrt_buf_ptr                                             pointer;
155   1       declare com_seg_vals structure(offset word,base word) AT (@wrt_buf_ptr);
156   1       declare (wrt_buf based wrt_buf_ptr)(50)                            byte;
157   1       declare (read_buf based wrt_buf_ptr) structure(count byte,chars(50) byte);
158   1       declare disk_flag                                                  byte;
159   1       declare question(*) byte data(20,0,33,'DISK SUPERVISOR (Enter new disk).');
160   1       declare error(*)              byte data(22,0,15,'Incorrect disk.');

                $EJECT
```

```
161   1     SET_UP_USER: PROCEDURE;

            /* This routine creates a default user so that the disk may be connected
               properly. In future versions this routine may be modified so that the
               user must input the user number (instead of 65535 being the default)
               this number must then correspond to the user number of the disk for
               security checking.*/

162   2     declare (done,loop)                                    byte;
163   2     declare temp_id                                        word;
164   2     declare id_token                                       token;
165   2     declare ids structure (number word,count word,id_number word);

166   2     ids.id_number=65535;
167   2     ids.number,ids.count=one;
168   2     id_token=RQ$CREATE$USER(@ids,@status);
169   2     call RQ$SET$DEFAULT$USER(null,id_token,@status);

170   2     END SET_UP_USER;


171   1     DISK_ERROR: PROCEDURE;

            /* This routine uses the write task to display the incorrect disk error. */

172   2     call MOVB(@error,@wrt_buf,18);
173   2     call RQ$SEND$MESSAGE(th_in_mbx,com_seg_vals.base,th_out_mbx,@status);
174   2     com_seg_vals.base=RQ$RECEIVE$MESSAGE(th_out_mbx,wait_forever,@dummy_mbx,@status);

175   2     END DISK_ERROR;

            $EJECT
```

```
176   1     ATTACH_DISK: PROCEDURE;

            /* This routine physically attaches the disk drive AFD1, if a single unit is
               being used then the unit is AFD0.*/

177   2     call RQ$A$PHYSICAL$ATTACH$DEVICE(@(4,'AFD1'),named_driver,disk_mailbox,@status);
178   2     disk_connection=RQ$RECEIVE$MESSAGE(disk_mailbox,wait_forever,@dummy_mbx,@status);
179   2     type_code=RQ$GET$TYPE(disk_connection,@status);
180   2     if type_code<>connection_code then
181   2       do;
182   3          call RQ$DELETE$SEGMENT(disk_connection,@status);
183   3          disk_flag=bad;
184   3          call disk_error;
185   3       end;
186   2     else disk_flag=good;

187   2     END ATTACH_DISK;



188   1     DETACH_DISK: PROCEDURE;

            /* This routine detaches the disk when required wether it be when the disk is
               to be changed or the disk originally attached was not the correct one.*/

189   2     call RQ$A$PHYSICAL$DETACH$DEVICE(disk_connection,not_hard,disk_mailbox,@status);
190   2     call RQ$UNCATALOG$OBJECT(this_job,@(4,'DCON'),@status);
191   2     gen_word=RQ$RECEIVE$MESSAGE(disk_mailbox,wait_forever,@dummy_mbx,@status);
192   2     disk_flag=bad;
193   2     call RQ$DELETE$SEGMENT(gen_word,@status);

194   2     END DETACH_DISK;

            $EJECT
```

```
195   1       CHECK_DIR: PROCEDURE(name_ptr);

              /* This procedure checks that the file supplied exists at base level on the
                 floppy disk, if not an error is flagged on the return.*/

196   2       declare name_ptr                                    pointer;

197   2       call RQ$A$ATTACH$FILE(id,disk_connection,name_ptr,disk_mailbox,@status);
198   2       if status<>E$OK then
199   2         do;
200   3            call detach_disk;
201   3            call disk_error;
202   3         end;
203   2       else
              do;
204   3            gen_word=RQ$RECEIVE$MESSAGE(disk_mailbox,wait_forever,@dummy_mbx,@status);
205   3            type_code=RQ$GET$TYPE(gen_word,@status);
206   3            if type_code<>connection_code then
207   3              do;
208   4                 call RQ$DELETE$SEGMENT(gen_word,@status);
209   4                 call detach_disk;
210   4                 call disk_error;
211   4              end;
212   3            else call RQ$A$DELETE$CONNECTION(gen_word,no_response,@status);
213   3         end;

214   2       END CHECK_DIR;

              $EJECT
```

```
215   1     FLOPPY_SECURITY_TASK: PROCEDURE PUBLIC;

            /* The main routine in this task looks up the required tokens for reading
               and writing to the terminal and it's own semaphore token at which it
               waits for a single unit. It then suspends the main task and prompts for
               the correct disk to be inserted and a carriage return to be typed. The
               task then tries to attach the drive and if successful checks that the
               root data and path directories exist. If they do then the main task is
               resumed and the task waits for the next unit at the disk semaphore. If
               the disk is incorrect then the whole process is repeated.*/

216   2     th_in_mbx=RQ$LOOKUP$OBJECT(this_job,@(7,'THINMBX'),wait_forever,@status);
217   2     th_out_mbx=RQ$LOOKUP$OBJECT(this_job,@(8,'THOUTMBX'),wait_forever,@status);
218   2     read_in=RQ$LOOKUP$OBJECT(this_job,@(6,'READIN'),wait_forever,@status);
219   2     read_out=RQ$LOOKUP$OBJECT(this_job,@(7,'READOUT'),wait_forever,@status);
220   2     disk_sem=RQ$LOOKUP$OBJECT(this_job,@(7,'DISKSEM'),wait_forever,@status);
221   2     main_task_token=RQ$LOOKUP$OBJECT(this_job,@(4,'MTSK'),wait_forever,@status);
222   2     disk_mailbox=RQ$CREATE$MAILBOX(fifo,@status);
223   2     com_seg_vals.offset=null;
224   2     disk_flag=bad;
225   2     call set_up_user;
226   2     do forever;
227   3        gen_word=RQ$RECEIVE$UNITS(disk_sem,one,wait_forever,@status);
228   3        call RQ$SUSPEND$TASK(main_task_token,@status);
229   3        com_seg_vals.base=RQ$CREATE$SEGMENT(50,@status);
230   3        done=bad;
231   3        do while not done;
232   4           call MOVB(@question,@wrt_buf,36);
233   4           call RQ$SEND$MESSAGE(th_in_mbx,com_seg_vals.base,th_out_mbx,@status);
234   4           com_seg_vals.base=RQ$RECEIVE$MESSAGE(th_out_mbx,wait_forever,@dummy_mbx,@status);
235   4           call RQ$SEND$MESSAGE(read_in,com_seg_vals.base,read_out,@status);
236   4           com_seg_vals.base=RQ$RECEIVE$MESSAGE(read_out,wait_forever,@dummy_mbx,@status);
237   4           if disk_flag=good then call detach_disk;
239   4           call attach_disk;
240   4           if disk_flag=good then call check_dir(@(7,'DATADEF'));
242   4           if disk_flag=good then call check_dir(@(7,'PATHDEF'));
244   4           if disk_flag=good then call RQ$CATALOG$OBJECT(this_job,disk_connection,@(4,'DCON'),@status);
246   4           if disk_flag=good then done=good;
248   4        end;
249   3        call MOVB(clear_all_lines,@wrt_buf,6);
250   3        call RQ$SEND$MESSAGE(th_in_mbx,com_seg_vals.base,th_out_mbx,@status);
251   3        com_seg_vals.base=RQ$RECEIVE$MESSAGE(th_out_mbx,wait_forever,@dummy_mbx,@status);
252   3        call RQ$DELETE$SEGMENT(com_seg_vals.base,@status);
253   3        call RQ$RESUME$TASK(main_task_token,@status);
254   3     end;

255   2     END FLOPPY_SECURITY_TASK;

256   1     END FLOPSEC;

            $EJECT
```

MODULE INFORMATION:

```
     CODE AREA SIZE      = 03E5H    997D
     CONSTANT AREA SIZE  = 0088H    136D
     VARIABLE AREA SIZE  = 002BH     43D
     MAXIMUM STACK SIZE  = 0020H     32D
     559 LINES READ
     0 PROGRAM WARNINGS
     0 PROGRAM ERRORS
```

DICTIONARY SUMMARY:

```
     116KB MEMORY AVAILABLE
     12KB MEMORY USED   (10%)
     0KB DISK SPACE USED
```

END OF PL/M-86 COMPILATION

8. TERMINAL HANDLER WRITE TASK.

```
iRMX 86 PL/M-86 V2.3 COMPILATION OF MODULE TH_WRITE
OBJECT MODULE PLACED IN TH/WRITE.OBJ
COMPILER INVOKED BY:  :LANG:PLM86 TH/WRITE.P86


                $TITLE ('TERMINAL HANDLER WRITE VER 86/2.0')

                $DEBUG
                $LARGE
                $OPTIMIZE(3)

   1            TH_WRITE:DO;

                $INCLUDE(INC/LITERALS.P86)
                $SAVE NOLIST
                $INCLUDE(/INC/RMX86/NCRMBX.EXT)
                $SAVE NOLIST
                $INCLUDE(/INC/RMX86/NCFOBJ.EXT)
                $SAVE NOLIST
                $INCLUDE(/INC/RMX86/NSNMES.EXT)
                $SAVE NOLIST
                $INCLUDE(/INC/RMX86/NRCMES.EXT)
                $SAVE NOLIST
                $INCLUDE(/INC/RMX86/IAOPEN.EXT)
                $SAVE NOLIST
                $INCLUDE(/INC/RMX86/IPATOV.EXT)
                $SAVE NOLIST
                $INCLUDE(/INC/RMX86/IACREL.EXT)
                $SAVE NOLIST
                $INCLUDE(/INC/RMX86/IAWRIT.EXT)
                $SAVE NOLIST
                $INCLUDE(/INC/RMX86/ICRUSR.EXT)
                $SAVE NOLIST
                $INCLUDE(/INC/RMX86/IWTIO.EXT)
                $SAVE NOLIST

                /* This task simply  creates two  mailboxes, a segment of memory and a
                   connection  to the  terminal all of which are  placed in the object
                   directory  where other tasks  may look them up. The task then waits
                   at the first  mailbox (th_in_mbx ) until a  segment of memory which
                   contains an ASCII  string is  received.  The ASCII  string is to be
                   displayed on the  terminal  and  is  preceeded by X, Y  coordinates
                   describing the  position  on the  screen it is to  appear at. These
                   figures  are  converted to  ASCII digits and are then placed in the
                   escape  string which  positions the  cursor,  followed by the ASCII
                   string, all of  which is then written to the  terminal  connection.
                   The  segment is then  returned via the second mailbox (th_out_mbx )
                   at which the sending task may pick it up and the task loops to wait
                   for the next segment of memory to be sent. */

                $EJECT
```

```
127   1       WRITE: PROCEDURE PUBLIC;

128   2       declare string(88)                                              byte;
129   2       declare (status,dummy_mbx)                                      word;
130   2       declare count                                                   byte;
131   2       declare (response,th_in_mbx,th_out_mbx,co,wuser,term_token)     token;
132   2       declare th_seg_ptr                                              pointer;
133   2       declare th_seg_vals structure (offset word,base word) AT (@th_seg_ptr);
134   2       declare (th_seg based th_seg_ptr)(80)                           byte;
135   2       declare ids(3)          word          data          (one,one,WORLD);

136   2       th_in_mbx=RQ$CREATE$MAILBOX(fifo,@status);
137   2       call RQ$CATALOG$OBJECT(this_job,th_in_mbx,@(7,'THINMBX'),@status);
138   2       th_out_mbx=RQ$CREATE$MAILBOX(fifo,@status);
139   2       call RQ$CATALOG$OBJECT(this_job,th_out_mbx,@(8,'THOUTMBX'),@status);
140   2       wuser=RQ$CREATE$USER(@ids,@status);
141   2       call RQ$A$PHYSICAL$ATTACH$DEVICE(@(2,'T0'),one,th_in_mbx,@status);
142   2       term_token=RQ$RECEIVE$MESSAGE(th_in_mbx,wait_forever,@dummy_mbx,@status);
143   2       call RQ$A$CREATE$FILE(wuser,term_token,null,null,null,null,must_create,th_in_mbx,@status);
144   2       co=RQ$RECEIVE$MESSAGE(th_in_mbx,wait_forever,@dummy_mbx,@status);
145   2       call RQ$A$OPEN(co,update,share_mode,no_response,@status);
146   2       call RQ$CATALOG$OBJECT(this_job,co,@(2,'CO'),@status);
147   2       call RQ$A$WRITE(co,@(ESC,'='),2,th_in_mbx,@status);
148   2       count=RQ$WAIT$IO(co,th_in_mbx,wait_forever,@status);
149   2       call MOVB(@(ESC,'E',0,0,';',0,0,'H'),@string,8);
150   2       th_seg_vals.offset=null;
151   2       do forever;
152   3          th_seg_vals.base=RQ$RECEIVE$MESSAGE(th_in_mbx,wait_forever,@response,@status);
153   3          string(2)=th_seg(0)/10+30H;
154   3          string(3)=th_seg(0) MOD 10+30H;
155   3          string(5)=th_seg(1)/10+30H;
156   3          string(6)=th_seg(1) MOD 10+30H;
157   3          count=th_seg(2)+8;
158   3          call MOVB(@th_seg(3),@string(8),th_seg(2));
159   3          call RQ$A$WRITE(co,@string,count,th_in_mbx,@status);
160   3          count=RQ$WAIT$IO(co,th_in_mbx,wait_forever,@status);
161   3          call RQ$SEND$MESSAGE(response,th_seg_vals.base,no_response,@status);
162   3       end;

163   2       END WRITE;

164   1       END TH_WRITE;

                $EJECT
```

MODULE INFORMATION:

```
     CODE AREA SIZE      = 021DH    541D
     CONSTANT AREA SIZE  = 0027H     39D
     VARIABLE AREA SIZE  = 006DH    109D
     MAXIMUM STACK SIZE  = 0020H     32D
     364 LINES READ
     0 PROGRAM WARNINGS
     0 PROGRAM ERRORS
```

DICTIONARY SUMMARY:

```
     116KB MEMORY AVAILABLE
     9KB MEMORY USED    (7%)
     0KB DISK SPACE USED
```

END OF PL/M-86 COMPILATION

9.TERMINAL HANDLER READ TASK.

```
iRMX 86 PL/M-86 V2.3 COMPILATION OF MODULE TH_READ
OBJECT MODULE PLACED IN TH/READ.OBJ
COMPILER INVOKED BY:  :LANG:PLM86 TH/READ.P86


            $TITLE ('TERMINAL HANDLER READ TASK VER 86/2.0')

            $DEBUG
            $LARGE
            $OPTIMIZE(2)

  1         TH_READ: DO;

            $INCLUDE(INC/LITERALS.P86)
            $SAVE NOLIST
            $INCLUDE(/INC/RMX86/NUCOBJ.EXT)
            $SAVE NOLIST
            $INCLUDE(/INC/RMX86/NIOBJ.EXT)
            $SAVE NOLIST
            $INCLUDE(/INC/RMX86/NCFMBX.EXT)
            $SAVE NOLIST
            $INCLUDE(/INC/RMX86/NSNMES.EXT)
            $SAVE NOLIST
            $INCLUDE(/INC/RMX86/NRCMES.EXT)
            $SAVE NOLIST
            $INCLUDE(/INC/RMX86/IAREAD.EXT)
            $SAVE NOLIST
            $INCLUDE(/INC/RMX86/IAWRIT.EXT)
            $SAVE NOLIST
            $INCLUDE(/INC/RMX86/IWTIO.EXT)
            $SAVE NOLIST

            /* This task is similar to the  th/write task.  It creates two mailboxes
               and looks up the  terminal  connection created by the th/write task.
               The read task then waits at the read_io  mailbox until a  segment of
               memory is  received. The  terminal  connection is  then read and the
               carriage return and linefeed are removed before the string is copied
               into the  segment with the  character  count in the  first byte. The
               segment is then  returned to the  requesting  task via the  read_out
               mailbox. The task also writes to the console to clear the write line
               and to position the cursor over the answer line before the readund. */

            $EJECT
```

10. GET FILENAME UTILITY TASK.

```
iRMX 86 PL/M-86 V2.3 COMPILATION OF MODULE GNUT
OBJECT MODULE PLACED IN UTILS/GNAME.OBJ
COMPILER INVOKED BY:  :LANG:PLM86 UTILS/GNAME.P86


            $TITLE ('GET NAME UTILITY TASK VER 86/2.0')

            $DEBUG
            $LARGE
            $OPTIMIZE(3)

  1         GNUT: DO;

            $INCLUDE(:INC:LITERALS.P86)
            $SAVE NOLIST
            $INCLUDE(/INC/RMX86/NCTOBJ.EXT)
            $SAVE NOLIST
            $INCLUDE(/INC/RMX86/NLUOBJ.EXT)
            $SAVE NOLIST
            $INCLUDE(/INC/RMX86/NCRMSX.EXT)
            $SAVE NOLIST
            $INCLUDE(/INC/RMX86/NRCMES.EXT)
            $SAVE NOLIST
            $INCLUDE(/INC/RMX86/NSNMES.EXT)
            $SAVE NOLIST

            /* This task creates one  single  mailbox ('gname mbx' ) at which it waits
               for a  segment of memory. The task then uses the th/write and th/read
               tasks to display a prompt on the terminal and ask the user for a file
               name. Once the  response is  received it is checked for illegal chars
               etc, if the name is  acceptable it is returned to the requesting task
               via the response mailbox whose token is  received with the segment in
               the  initial request. If the  file name contains  illegal characters,
               is too short or long, then  another name is  requested until the task
               is satisfied. The gname task then waits for the next request. */

            $EJECT
```

```
112   1       GET_NAME: PROCEDURE PUBLIC;

113   2       declare status                                                  word;
114   2       declare (done,loop)                                             byte;
115   2       declare (th_in_mbx,th_out_mbx,read_in,read_out,gname_mbx,response)   token;
116   2       declare wrt_buf_ptr                                            pointer;
117   2       declare wrt_buf_vals structure(offset word,base word)  AT   (@wrt_buf_ptr);
118   2       declare (wrt_buf based wrt_buf_ptr)(80)                         byte;
119   2       declare (read_buf based wrt_buf_ptr) structure (count byte,chars(80) byte);
120   2       declare error(*) byte data (22,0,45,'Name contains illegal chars (Please correct).');

121   2       gname_mbx=RQ$CREATE$MAILBOX(fifo,@status);
122   2       call RQ$CATALOG$OBJECT(this_job,gname_mbx,@(5,'GNMBX'),@status);
123   2       read_out=RQ$LOOKUP$OBJECT(this_job,@(7,'READOUT'),wait_forever,@status);
124   2       read_in=RQ$LOOKUP$OBJECT(this_job,@(6,'READIN'),wait_forever,@status);
125   2       th_in_mbx=RQ$LOOKUP$OBJECT(this_job,@(7,'THINMBX'),wait_forever,@status);
126   2       th_out_mbx=RQ$LOOKUP$OBJECT(this_job,@(8,'THOUTMBX'),wait_forever,@status);
127   2       wrt_buf_vals.offset=null;
128   2       do forever;
129   3          wrt_buf_vals.base=RQ$RECEIVE$MESSAGE(gname_mbx,wait_forever,@response,@status);
130   3          done=bad;
131   3          do while not done;
132   4             done=good;
133   4             call RQ$SEND$MESSAGE(read_in,wrt_buf_vals.base,read_out,@status);
134   4             wrt_buf_vals.base=RQ$RECEIVE$MESSAGE(read_out,wait_forever,no_response,@status);
135   4             if ((read_buf.count<3)OR(read_buf.count>9)) then done=bad;
137   4             do loop=0 to read_buf.count-1;
138   5                if (read_buf.chars(loop)<'A' OR read_buf.chars(loop)>'Z') then
139   5                  if (read_buf.chars(loop)<'0' OR read_buf.chars(loop)>'9') then
140   5                    if read_buf.chars(loop)<>'.' then done=bad;
142   5             end;
143   4             if done=bad then
144   4               do;
145   5                 call MOVB(@error,@wrt_buf,47);
146   5                 call RQ$SEND$MESSAGE(th_in_mbx,wrt_buf_vals.base,th_out_mbx,@status);
147   5                 wrt_buf_vals.base=RQ$RECEIVE$MESSAGE(th_out_mbx,wait_forever,no_response,@status);
148   5               end;
149   4          end;
150   3          call RQ$SEND$MESSAGE(response,wrt_buf_vals.base,no_response,@status);
151   3       end;

152   2       END GET_NAME;

153   1       END GNUT;

                $EJECT
```

MODULE INFORMATION:

```
     CODE AREA SIZE      = 01D4H    468D
     CONSTANT AREA SIZE  = 0056H     86D
     VARIABLE AREA SIZE  = 0014H     20D
     MAXIMUM STACK SIZE  = 0014H     20D
     272 LINES READ
     0 PROGRAM WARNINGS
     0 PROGRAM ERRORS
```

DICTIONARY SUMMARY:

```
     116KB MEMORY AVAILABLE
     8KB MEMORY USED    (6%)
     0KB DISK SPACE USED
```

END OF PL/M-86 COMPILATION

11. CREATE FILE UTILITY TASK.

```
iRMX 86 PL/M-86 V2.3 COMPILATION OF MODULE CRFLUT
OBJECT MODULE PLACED IN UTILS/CRE.OBJ
COMPILER INVOKED BY:  :LANG:PLM86 UTILS/CRE.P86


            $TITLE ('CREATE FILE TASK UTILITY VER 86/2.0')

            $DEBUG
            $LARGE
            $OPTIMIZE(3)

  1         CRFLUT: DO;

            $INCLUDE(INC/LITERALS.P86)
            $SAVE NOLIST
            $INCLUDE(/INC/RMX86/NCRSEG.EXT)
            $SAVE NOLIST
            $INCLUDE(/INC/RMX86/NDLSEG.EXT)
            $SAVE NOLIST
            $INCLUDE(/INC/RMX86/NCRM8X.EXT)
            $SAVE NOLIST
            $INCLUDE(/INC/RMX86/NDLMBX.EXT)
            $SAVE NOLIST
            $INCLUDE(/INC/RMX86/NSNMES.EXT)
            $SAVE NOLIST
            $INCLUDE(/INC/RMX86/NPCMES.EXT)
            $SAVE NOLIST
            $INCLUDE(/INC/RMX86/NLUOBJ.EXT)
            $SAVE NOLIST
            $INCLUDE(/INC/RMX86/NCIOBJ.EXT)
            $SAVE NOLIST
            $INCLUDE(/INC/RMX86/IACRFL.EXT)
            $SAVE NOLIST

            /* This task is  responsible for creating all new files in the system. It
               creates one mailbox of it's own called  crw_file_mbx  and retrieves it
               in the object directory. If a task wishes to create a new file it must
               look up  this  mailbox and send the token of the directory in which the
               new file is to be placed. This task then  receives the directory token
               and sends it  straight to the list  directory task where it's contents
               are displayed on the terminal. A prompt is then displayed via the write
               task to ask for a file name and a  segment is then created and sent to
               the get file name task which  returns a valid file name. The task then
               tries to  create a file  with this file name  and returns the received
               token back to the requesting task. The  requesting task is responsible
               for checking  wether the  token  returned is a  token for a file or an
               error  segment. If it is an error  segment the task must delete it and
               if it wishes retry the creation routine. */

            $EJECT
```

```
124   1     CREATE_A_FILE: PROCEDURE PUBLIC;

125   2     declare wrt_buf_ptr                                          pointer;
126   2     declare wrt_buf_vals structure(offset word,base word)  AT   (@wrt_buf_ptr);
127   2     declare (wrt_buf based wrt_buf_ptr)(42)                        byte;
128   2     declare status                                                 word;
129   2     declare (dir_token,file_token,cre_file_mbx,gname_mbx)         token;
130   2     declare (th_in_mbx,th_out_mbx,list_dir_mbx,response)          token;
131   2     declare question(*) byte data (20,0,44,'Please enter the new file name (9 Chars Max)');

132   2     cre_file_mbx=RQ$CREATE$MAILBOX(fifo,@status);
133   2     call RQ$CATALOG$OBJECT(this_job,cre_file_mbx,@(6,'CREMBX'),@status);
134   2     list_dir_mbx=RQ$LOOKUP$OBJECT(this_job,@(5,'LDMBX'),wait_forever,@status);
135   2     gname_mbx=RQ$LOOKUP$OBJECT(this_job,@(5,'GNMBX'),wait_forever,@status);
136   2     th_in_mbx=RQ$LOOKUP$OBJECT(this_job,@(7,'THINMBX'),wait_forever,@status);
137   2     th_out_mbx=RQ$LOOKUP$OBJECT(this_job,@(8,'THOUTMBX'),wait_forever,@status);
138   2     wrt_buf_vals.offset=null;
139   2     do forever;
140   3        dir_token=RQ$RECEIVE$MESSAGE(cre_file_mbx,wait_forever,@response,@status);
141   3        wrt_buf_vals.base=RQ$CREATE$SEGMENT(50,@status);
142   3        call RQ$SEND$MESSAGE(list_dir_mbx,dir_token,cre_file_mbx,@status);
143   3        dir_token=RQ$RECEIVE$MESSAGE(cre_file_mbx,wait_forever,no_response,@status);
144   3        call MOVB(@question,@wrt_buf,47);
145   3        call RQ$SEND$MESSAGE(th_in_mbx,wrt_buf_vals.base,th_out_mbx,@status);
146   3        wrt_buf_vals.base=RQ$RECEIVE$MESSAGE(th_out_mbx,wait_forever,no_response,@status);
147   3        call RQ$SEND$MESSAGE(gname_mbx,wrt_buf_vals.base,cre_file_mbx,@status);
148   3        wrt_buf_vals.base=RQ$RECEIVE$MESSAGE(cre_file_mbx,wait_forever,no_response,@status);
149   3        call RQ$A$CREATE$FILE(id,dir_token,@wrt_buf,DRAW,null,size_1k_bytes,must_create,cre_file_mbx,@status);
150   3        file_token=RQ$RECEIVE$MESSAGE(cre_file_mbx,wait_forever,no_response,@status);
151   3        call RQ$DELETE$SEGMENT(wrt_buf_vals.base,@status);
152   3        call RQ$SEND$MESSAGE(response,file_token,no_response,@status);
153   3     end;

154   2     END CREATE_A_FILE;

155   1     END CRFLUT;

            $EJECT
```

```
MODULE INFORMATION:

     CODE AREA SIZE      = 01DBH    475D
     CONSTANT AREA SIZE  = 0053H     83D
     VARIABLE AREA SIZE  = 0016H     22D
     MAXIMUM STACK SIZE  = 0020H     32D
     331 LINES READ
     0 PROGRAM WARNINGS
     0 PROGRAM ERRORS

DICTIONARY SUMMARY:

     116KB MEMORY AVAILABLE
     9KB MEMORY USED   (7%)
     0KB DISK SPACE USED

END OF PL/M-96 COMPILATION
```

12. ATTACHFILE UTILITY TASK.

```
iRMX 86 PL/M-86 V2.3 COMPILATION OF MODULE ATFLUT
OBJECT MODULE PLACED IN UTILS/ATT.OBJ
COMPILER INVOKED BY:  :LANG:PLM86 UTILS/ATT.P86


                $TITLE ('CONNECT FILE UTILITY TASK VER 86/2.0')

                $DEBUG
                $LARGE
                $OPTIMIZE(3)

  1             ATFLUT: DO;

                $INCLUDE(INC/LITERALS.P86)
  2             $SAVE NOLIST
                $INCLUDE(/INC/RMX86/NDLSEG.EXT)
  3             $SAVE NOLIST
                $INCLUDE(/INC/RMX86/NCRSEG.EXT)
  4             $SAVE NOLIST
                $INCLUDE(/INC/RMX86/NCRMBX.EXT)
  5             $SAVE NOLIST
                $INCLUDE(/INC/RMX86/NDLMBX.EXT)
  6             $SAVE NOLIST
                $INCLUDE(/INC/RMX86/NRCMES.EXT)
  7             $SAVE NOLIST
                $INCLUDE(/INC/RMX86/NS  MES.EXT)
  8             $SAVE NOLIST
                $INCLUDE(/INC/RMX86/NLUOBJ.EXT)
  9             $SAVE NOLIST
                $INCLUDE(/INC/RMX86/NCTOBJ.EXT)
 10             $SAVE NOLIST
                $INCLUDE(/INC/RMX86/NGTTYP.EXT)
 11             $SAVE NOLIST
                $INCLUDE(/INC/RMX86/IAATFL.EXT)
 12             $SAVE NOLIST

                /* This tasks job is to  connect a file to the  system from a given  directory.
                   The task creates one  mailbox  called  att_file_mbx and  looks up the tokens
                   for the  th/write task, list  directory  mailbox and the set file name task.
                   It then  waits at the  attach  file  mailbox  until it receives a  directory
                   token, this token is  immediately passed to the list  directory task so that
                   the file names of the files in the directory are all listed on the terminal.
                   When the  token is  received back the task  creates a  temporary  segment of
                   memory and sends it to the set name task. The  segment will then be returned
                   containing a  valid  file name and the task  attempts to  attach the file of
                   that name. If the call is  unsuccessful the whole  process is repeated until
                   a file is attached. Once a valid file has been attched the screen is cleared
                   and the  file  token is  returned to the  requesting  task via the  response
                   mailbox which was  received along with the  directory  token in the  initial
                   request.The  task then  sleeps at the  attach file  mailbox  waiting for the
                   next directory token. */
```

```
127   1     CONNECT_FILE: PROCEDURE PUBLIC;

128   2     declare done                                                    byte;
129   2     declare wrt_buf_ptr                                          pointer;
130   2     declare wrt_buf_vals structure(offset word,base word)  AT   (@wrt_buf_ptr);
131   2     declare (wrt_buf based wrt_buf_ptr)(42)                         byte;
132   2     declare (status,type_code)                                      word;
133   2     declare (dir_token,file_token,att_file_mbx,response,sname_mbx)  token;
134   2     declare (th_in_mbx,th_out_mbx,list_dir_mbx)                     token;
135   2     declare question(*) byte data (20,0,40,'Please enter name of data file required.');

136   2     att_file_mbx=RQ$CREATE$MAILBOX(fifo,@status);
137   2     call RQ$CATALOG$OBJECT(this_job,att_file_mbx,@(6,'ATTMBX'),@status);
138   2     list_dir_mbx=RQ$LOOKUP$OBJECT(this_job,@(5,'LDMBX'),wait_forever,@status);
139   2     sname_mbx=RQ$LOOKUP$OBJECT(this_job,@(5,'GNMBX'),wait_forever,@status);
140   2     th_in_mbx=RQ$LOOKUP$OBJECT(this_job,@(7,'THINMBX'),wait_forever,@status);
141   2     th_out_mbx=RQ$LOOKUP$OBJECT(this_job,@(8,'THOUTMBX'),wait_forever,@status);
142   2     do forever;
143   3        dir_token=RQ$RECEIVE$MESSAGE(att_file_mbx,wait_forever,@response,@status);
144   3        call RQ$SEND$MESSAGE(list_dir_mbx,dir_token,att_file_mbx,@status);
145   3        dir_token=RQ$RECEIVE$MESSAGE(att_file_mbx,wait_forever,no_response,@status);
146   3        wrt_buf_vals.base=RQ$CREATE$SEGMENT(50,@status);
147   3        wrt_buf_vals.offset=null;
148   3        done=bad;
149   3        do while not done;
150   4           call MOVB(@question,@wrt_buf,43);
151   4           call RQ$SEND$MESSAGE(th_in_mbx,wrt_buf_vals.base,th_out_mbx,@status);
152   4           wrt_buf_vals.base=RQ$RECEIVE$MESSAGE(th_out_mbx,wait_forever,no_response,@status);
153   4           call RQ$SEND$MESSAGE(sname_mbx,wrt_buf_vals.base,att_file_mbx,@status);
154   4           wrt_buf_vals.base=RQ$RECEIVE$MESSAGE(att_file_mbx,wait_forever,no_response,@status);
155   4           call RQ$A$ATTACH$FILE(id,dir_token,@wrt_buf,att_file_mbx,@status);
156   4           if status=E$OK then
157   4             do;
158   5                file_token=RQ$RECEIVE$MESSAGE(att_file_mbx,wait_forever,no_response,@status);
159   5                type_code=RQ$GET$TYPE(file_token,@status);
160   5                if type_code=connection_code then done=good;
162   5                else call RQ$DELETE$SEGMENT(file_token,@status);
163   5             end;
164   4        end;
165   3        call MOVB(clear_screen,@wrt_buf,6);
166   3        call RQ$SEND$MESSAGE(th_in_mbx,wrt_buf_vals.base,th_out_mbx,@status);
167   3        wrt_buf_vals.base=RQ$RECEIVE$MESSAGE(th_out_mbx,wait_forever,no_response,@status);
168   3        call RQ$DELETE$SEGMENT(wrt_buf_vals.base,@status);
169   3        call RQ$SEND$MESSAGE(response,file_token,no_response,@status);
170   3     end;

171   2     END CONNECT_FILE;

172   1     END ATFLUT;

            EJECT
```

```
MODULE INFORMATION:

     CODE AREA SIZE     = 0248H    584D
     CONSTANT AREA SIZE = 0055H     85D
     VARIABLE AREA SIZE = 0019H     25D
     MAXIMUM STACK SIZE = 0016H     22D
     349 LINES READ
     0 PROGRAM WARNINGS
     0 PROGRAM ERRORS

DICTIONARY SUMMARY:

     116KB MEMORY AVAILABLE
     9KB MEMORY USED   (7%)
     0KB DISK SPACE USED

END OF PL/M-86 COMPILATION
```

13. LIST DIRECTORY TASK.

```
iRMX 86 PL/M-86 V2.3 COMPILATION OF MODULE DIRECTORY
OBJECT MODULE PLACED IN UTILS/LIST.OBJ
COMPILER INVOKED BY:  :LANG:PLM86 UTILS/LIST.P86


               $TITLE ('DIRECTORY LIST TASK VER 86/2.0')

               $DEBUG
               $LARGE
               $OPTIMIZE(3)

     1         DIRECTORY: DO;

               $INCLUDE(INC/LITERALS.P86)
               $SAVE NOLIST
               $INCLUDE(/INC/RMX86/NLUOBJ.EXT)
               $SAVE NOLIST
               $INCLUDE(/INC/RMX86/NCTOBJ.EXT)
               $SAVE NOLIST
               $INCLUDE(/INC/RMX86/NCRSEG.EXT)
               $SAVE NOLIST
               $INCLUDE(/INC/RMX86/NDLSEG.EXT)
               $SAVE NOLIST
               $INCLUDE(/INC/RMX86/NSNMES.EXT)
               $SAVE NOLIST
               $INCLUDE(/INC/RMX86/NRCMES.EXT)
               $SAVE NOLIST
               $INCLUDE(/INC/RMX86/NCRMBX.EXT)
               $SAVE NOLIST
               $INCLUDE(/INC/RMX86/IAGTDE.EXT)
               $SAVE NOLIST

               /* This task is to list the  contents of a  directory on the  terminal. It
                  creates a  single  mailbox called  list_dir_mbx, then  enters it in the
                  job  directory. If a task  wishes to list a  directory on the screen it
                  simply looks up the token of the mailbox and sends the directory token.
                  This  task then  receives the  token, creates a  segment of  memory and
                  clears the screen using the th/write task. Then using the get directory
                  entry  call it  sends the  segment of  memory to be  filled in with the
                  entries  file name and  again after  checking the  entry is valid sends
                  the file  string to the  th/write task to be displayed. Once the end of
                  directory  code is  received the  segment of  memory is deleted and the
                  directory  token is  returned  to the  requesting  task. The  task then
                  sleeps at the list_dir_mbx waiting for the next directory token.*/

               $EJECT
```

```
121   1     LIST: PROCEDURE PUBLIC;

122   2     declare  (entry_number,status,condition)                    word;
123   2     declare  (list_dir_mbx,dir_token,response,th_in_mbx,th_out_mbx)  token;
124   2     declare  (seg_ptr,wrt_buf_ptr)                             pointer;
125   2     declare  wrt_buf_vals structure (offset word,base word) AT (@wrt_buf_ptr);
126   2     declare  (wrt_buf based wrt_buf_ptr)(16)                    byte;
127   2     declare  seg_ptr_vals structure (offset word,base word) AT  (@seg_ptr);
128   2     declare  (entry based seg_ptr)  structure  (condition word,name(14) byte);

129   2     th_in_mbx=RQ$LOOK$UP$OBJECT(this_job,@(7,'THINMBX'),wait_forever,@status);
130   2     th_out_mbx=RQ$LOOK$UP$OBJECT(this_job,@(8,'THOUTMBX'),wait_forever,@status);
131   2     list_dir_mbx=RQ$CREATE$MAILBOX(fifo,@status);
132   2     call RQ$CATALOG$OBJECT(this_job,list_dir_mbx,@(5,'LDMBX'),@status);
133   2     seg_ptr_vals.offset=null;
134   2     wrt_buf_vals.offset=null;
135   2     do forever;
136   3        dir_token=RQ$RECEIVE$MESSAGE(list_dir_mbx,wait_forever,@response,@status);
137   3        wrt_buf_vals.base=RQ$CREATE$SEGMENT(32,@status);
138   3        call MOVB(clear_screen,@wrt_buf,6);
139   3        call RQ$SEND$MESSAGE(th_in_mbx,wrt_buf_vals.base,th_out_mbx,@status);
140   3        wrt_buf_vals.base=RQ$RECEIVE$MESSAGE(th_out_mbx,wait_forever,no_response,@status);
141   3        condition=E$OK;
142   3        entry_number=null;
143   3        wrt_buf(0)=4;
144   3        wrt_buf(1)=0;
145   3        wrt_buf(2)=14;
146   3        do while ((condition=E$OK)OR(condition=E$EMPTY$ENTRY));
147   4           call RQ$A$GET$DIRECTORY$ENTRY(dir_token,entry_number,list_dir_mbx,@status);
148   4           seg_ptr_vals.base=RQ$RECEIVE$MESSAGE(list_dir_mbx,wait_forever,no_response,@status);
149   4           entry_number=entry_number+1;
150   4           condition=entry.condition;
151   4           if condition=E$OK then
152   4             do;
153   5                call MOVB(@entry.name,@wrt_buf(3),14);
154   5                call RQ$SEND$MESSAGE(th_in_mbx,wrt_buf_vals.base,th_out_mbx,@status);
155   5                wrt_buf_vals.base=RQ$RECEIVE$MESSAGE(th_out_mbx,wait_forever,no_response,@status);
156   5                wrt_buf(1)=wrt_buf(1)+20;
157   5                if wrt_buf(1)>60 then
158   5                  do;
159   6                     wrt_buf(1)=0;
160   6                     wrt_buf(0)=wrt_buf(0)+1;
161   6                  end;
162   5             end;
163   4           call RQ$DELETE$SEGMENT(seg_ptr_vals.base,@status);
164   4        end;
165   3     call RQ$DELETE$SEGMENT(wrt_buf_vals.base,@status);
166   3     call RQ$SEND$MESSAGE(response,dir_token,no_response,@status);
167   3     end;

168   2     END LIST;
```

```
 169   1       END DIRECTORY;

MODULE INFORMATION:

     CODE AREA SIZE     = 01E3H     483D
     CONSTANT AREA SIZE = 001DH      29D
     VARIABLE AREA SIZE = 0018H      24D
     MAXIMUM STACK SIZE = 0014H      20D
     322 LINES READ
     0 PROGRAM WARNINGS
     0 PROGRAM ERRORS

DICTIONARY SUMMARY:

     116KB MEMORY AVAILABLE
     9KB MEMORY USED    (7%)
     0KB DISK SPACE USED

END OF PL/M-86 COMPILATION
```

```
 121   1       READ: PROCEDURE PUBLIC;

 122   2       declare (status,bytes_read)                                word;
 123   2       declare (read_in,read_out,co,response)                     token;
 124   2       declare rd_buf_ptr                                         pointer;
 125   2       declare rd_buf_vals structure (offset word,base word) AT (@rd_buf_ptr);
 126   2       declare (read_buf based rd_buf_ptr) structure(count byte,chars(80) byte);

 127   2       read_in=RQ$CREATE$MAILBOX(fifo,@status);
 128   2       call RQ$CATALOG$OBJECT(this_job,read_in,@(6,'READIN'),@status);
 129   2       read_out=RQ$CREATE$MAILBOX(fifo,@status);
 130   2       call RQ$CATALOG$OBJECT(this_job,read_out,@(7,'READOUT'),@status);
 131   2       co=RQ$LOOK$UP$OBJECT(this_job,@(2,'CO'),wait_forever,@status);
 132   2       rd_buf_vals.offset=null;
 133   2       do forever;
 134   3          rd_buf_vals.base=RQ$RECEIVE$MESSAGE(read_in,wait_forever,@response,@status);
 135   3          call RQ$A$WRITE(co,@(ESC,'[21;00H'),8,read_in,@status);
 136   3          bytes_read=RQ$WAIT$IO(co,read_in,wait_forever,@status);
 137   3          call RQ$A$READ(co,@read_buf.chars,one_set,read_in,@status);
 138   3          bytes_read=RQ$WAIT$IO(co,read_in,wait_forever,@status);
 139   3          read_buf.count=LOW(bytes_read-2);
 140   3          call RQ$A$WRITE(co,@(ESC,'[21;00H',ESC,'[J'),11,read_in,@status);
 141   3          bytes_read=RQ$WAIT$IO(co,read_in,wait_forever,@status);
 142   3          call RQ$SEND$MESSAGE(response,rd_buf_vals.base,no_response,@status);
 143   3       end;

 144   2       END READ;

 145   1       END TH_READ;


MODULE INFORMATION:

     CODE AREA SIZE      = 0152H    338D
     CONSTANT AREA SIZE  = 0025H     37D
     VARIABLE AREA SIZE  = 0010H     16D
     MAXIMUM STACK SIZE  = 0016H     22D
     347 LINES READ
     0 PROGRAM WARNINGS
     0 PROGRAM ERRORS

DICTIONARY SUMMARY:

     114KB MEMORY AVAILABLE
     9KB MEMORY USED   (7%)
     0KB DISK SPACE USED

END OF PL/M-86 COMPILATION
```

14. ASM86 EQUATES DICTIONARY.

```
$SAVE NOLIST

        /************************************************
         *                                              *
         *               ASM86 DICTIONARY               *
         *               ****************               *
         *                                              *
         * This file is the equate dictionary for all   *
         * the ASM86 modules in the application and is  *
         *     'included' in all such modules           *
         ************************************************/

BIT0MASK        EQU     00000001B               ;MASKS FOR
BIT1MASK        EQU     00000010B               ;SPECIFIC
BIT2MASK        EQU     00000100B               ;BYTE BITS
BIT5MASK        EQU     11011111B               ;
BIT7MASK        EQU     01111111B               ;
BIT6MASK        EQU     10111111B               ;
BIT76MASK       EQU     00111111B               ;

CNTRNTRL        EQU     00A0H                   ;CONTROL PORT FOR D-A

DAWORD          EQU     08H                     ;D-A INITIALIZING WORD
DAMASK          EQU     00000010B               ;D-A I/P BUFF FULL MASK
DEFSPEED        EQU     0CH                     ;DEFAULT MOVE SPEED
DIRPRTADDR      EQU     0400H                   ;ADDR OF DIR & ENABLE PORT

ENBPRTADDR      EQU     0408H                   ;TIMER ENABLE BITS

FLAGMASK        EQU     7FFFH                   ;REMOVES TOP FLAG

GATEPORT        EQU     00C9H                   ;GATE FOR 86/30 TIMER
GENABLE         EQU     09H                     ;STATUS REG BIT 1 = 1

INPUT_MASK      EQU     3FH                     ;SAME AS INPUT_ROUTINES IN PLM

KEYROW          EQU     0CCH                    ;KEYBOARD OUTPUT PORT
KEYCOL          EQU     0C8H                    ;KEYBOARD INPUT PORT

LEVEL4          EQU     0048H                   ;MASTER LEVEL 4 INTERRUPT CODE

MODEWORD        EQU     70H                     ;COUNTER 1,11,MODE 0
MOVE_MASK       EQU     23H                     ;SAME AS SPEED_EXIT_ONLY IN PLM

ONE             EQU     0001H                   ;NUMBER ONE

ROTONLY         EQU     00111000B               ;JUST ROTATION ENABLES
```

```
START           EQU     0FF01H          ;STARTWELD CONTROL WORD
STOP            EQU     0FF02H          ;STOPWELD CONTROL WORD
SWITCHPRT       EQU     00C0H           ;PORT ADDRESESS
STATPRT         EQU     00A2H           ;

TIMER0ms        EQU     5FB7H           ;
TMRCNTRL        EQU     00D6H           ;TIMER ON 86/30 CONTROL PORT
TMRINPUT        EQU     00D2H           ;PORT ADDRESS OF TIMER ON 86/30
TIMER1          EQU     0410H           ;ADDRESESS OF TIMERS ON
TIMER2          EQU     0412H           ;INTERFACE BOARD FOR THREE
TIMER3          EQU     0414H           ;AXIS MACHINE
TIMER4          EQU     0418H
TIMER5          EQU     041AH
TIMER6          EQU     041CH

XYZONLY         EQU     00000111B       ;XYZ MOTORS ONLY

FREESTORE
```

15. PL/M LITERALS DICTIONARY.

```
$NOPAGE NOLIST

        /*************************************************
        *                                                *
        *                PL/M DICTIONARY                 *
        *                ***************                 *
        *                                                *
        * This file is the literal dictionary for all    *
        * the PL/M modules in the application and is     *
        *    'included' in all such modules              *
        *************************************************/

declare         address_mode            literally       '02';
declare         auto_ret_no_echo        literally       '07';
declare         answer_line             literally       '@(21,0,0)';

declare         backwards               literally       '01H';
declare         bad                     literally       '00H';
declare         bell3                   literally       '@(0,0,3,7,7,7)';
declare         bell                    literally       '@(0,0,1,7)';

declare         col                     literally       '01';
declare         char_count              literally       '02';
declare         clear_screen            literally       '@(3,0,3,1BH,5BH,4AH)';
declare         clear_all_screen        literally       '@(0,0,3,1BH,5BH,4AH)';
declare         clear_all_lines         literally       '@(20,0,3,1BH,5BH,4AH)';
declare         clear_question_line     literally       '@(20,0,3,1BH,5BH,4BH)';
declare         clear_answer_line       literally       '@(21,0,3,1BH,5BH,4BH)';
declare         clear_error_line        literally       '@(22,0,3,1BH,5BH,4BH)';
declare         connection_code         literally       '101H';
declare         control_point           literally       '8000H';
declare         CR                      literally       '0DH';

declare         DDAC                    literally       '0FH';
declare         DRAU                    literally       '0FH';
declare         data_priority           literally       '235';

declare         error_flag              literally       '0FFH';
declare         error_line              literally       '@(22,0,0)';
declare         edit_keys               literally       '43H';
declare         ESC                     literally       '1BH';

declare         forward_mode            literally       '03';
declare         false                   literally       '00H';
declare         file_write              literally       '02';
declare         file_read               literally       '01';
declare         fifo                    literally       '00H';
declare         forever                 literally       'WHILE 1';

declare         dn_priority             literally       '225';
declare         get_terminal_data       literally       '04';
declare         good                    literally       '0FFH';
```

```
declare     id                  literally   '00';
declare     input_routines      literally   '3FH';
declare     inc_8087            literally   '01';

declare     level_four          literally   '001001000B';

declare     match               literally   '0FFFFH';
declare     must_create         literally   '0FFH';

declare     not_hard            literally   '00';
declare     no_waiting          literally   '0000H';
declare     no_int_task         literally   '00';
declare     no_response         literally   '0000H';
declare     null                literally   '00H';
declare     not_E$OK            literally   '0FFH';
declare     no_data_sed         literally   '00';
declare     nuc_allocated_stk   literally   '00';
declare     no_8087             literally   '00';
declare     normal_terminal     literally   '02';
declare     named_driver        literally   '04';

declare     one_sec             literally   '10000';
declare     one_set             literally   '80';
declare     one                 literally   '01';
declare     on                  literally   '0FFH';
declare     off                 literally   '00H';

declare     path_priority       literally   '235';
declare     physical_driver     literally   '1';

declare     question_line       literally   '@(20,0,0)';

declare     row                 literally   '00';
declare     root_job            literally   '03H';
declare     read_priority       literally   '210';
declare     real_error          literally   '0FFH';
```

```
declare     segment_code            literally     '06H';
declare     sec_disk_priority       literally     '220';
declare     space                   literally     '20H';
declare     supervisor_priority     literally     '240';
declare     size_1K_bytes           literally     '0400H';
declare     share_mode              literally     '03H';
declare     set_fix_towards_0       literally     '0FBFH';
declare     set_fix_nearest_int     literally     '03BFH';
declare     set_terminal_data       literally     '05';
declare     speed_exit_only         literally     '023H';
declare     stk_600_bytes           literally     '600';

declare     token                   literally     'word';
declare     true                    literally     '0FFH';
declare     this_job                literally     '00H';
declare     this_task               literally     '00H';
declare     trace_keys              literally     '0FFH';
declare     task_flags              literally     '00H';

declare     update                  literally     '03';
declare     utils_priority          literally     '230';
declare     unused                  literally     '00';

declare     wait_forever            literally     '0FFFFH';
declare     WORLD                   literally     '0FFFFH';
declare     world_security          literally     '@(5,36,35,35,33,35)';
declare     write_priority          literally     '200';
declare     weld_priority           literally     '235';
declare     while_weld_priority     literally     '67';
declare     while_move_priority     literally     '67';

declare     E$OK                    literally     '00H';
declare     E$FEXIST                literally     '20H';
declare     E$FNEXIST               literally     '21H';
declare     E$EMPTY$ENTRY           literally     '24H';
declare     E$SPACE                 literally     '29H';

$RESTORE
```

16. MACROS.

A). PL/M FILE REPAIR MACRO

B). ASM86 FILE REPAIR MACRO

C). LINK MACRO FOR APPLICATION CODE.

D). LOCATE MACRO FOR APPLICATION CODE.

E). REPAIR MACRO TO REPLACE APPLICATION CODE IN FULL SYSTEM.

```
;*****************************************************************
;*                                                               *
;*    THIS FILE REPAIRS THE  SUPERVISOR MODULE BY ASSEMBLING     *
;*    THE FILE  INPUT  AND  THEN  USES THE LINK  AND  LOCATE     *
;*    MACROS WITH THE REPAIR MACRO TO COMPLETE THE OPERATION     *
;*     ALL OUTPUT IS SENT TO THE FILE LOG                        *
;*                                                               *
;*           USE: SUBMIT CSD/REPA(ASSEMBLER_FILE)                *
;*                ********************************               *
;*                                                               *
;*****************************************************************
;
DELETE LOG
ASM86 %0.A86 EP NOMR
SUBMIT CSD/ALINK TO LOG
SUBMIT CSD/ALOC AFTER LOG
SUBMIT CSD/AREP AFTER LOG
```

```
;*****************************************************************
;*                                                               *
;*    THIS FILE  REPAIRS THE  SUPERVISOR  MODULE BY COMPLING     *
;*    THE FILE  INPUT  AND  THEN  USES THE LINK  AND  LOCATE     *
;*    MACROS WITH THE REPAIR MACRO TO COMPLETE THE OPERATION     *
;*    ALL OUTPUT IS SENT TO THE FILE LOG                         *
;*                                                               *
;*           USE: SUBMIT CSD/REPA(PL/M_FILE)                     *
;*                ***************************                    *
;*                                                               *
;*****************************************************************
;
DELETE LOG
PLM86 %0.P86
SUBMIT CSD/ALINK TO LOG
SUBMIT CSD/ALOC AFTER LOG
SUBMIT CSD/AREP AFTER LOG
```

```
;*****************************************************************
;*                                                               *
;*      THIS FILE TAKES ALL THE PROGRAM MODULES AND              *
;*      LINKS THEM WITH THE NUCLEUS, BASIC I/O AND               *
;*      8087 LIBRARIES. THE RESULTING OBJECT FILE IS             *
;*      PROG/APPL.LNK, WITH THE LINK MAP IN PROG/APPL.MP1        *
;*                                                               *
;*****************************************************************
#
LINK86                                  &
       MAIN/CLI.OBJ,                    &
       TH/WRITE.OBJ,                    &
       TH/READ.OBJ,                     &
       UTILS/LIST.OBJ,                  &
       UTILS/ATI.OBJ,                   &
       UTILS/CRE.OBJ,                   &
       UTILS/GNAME.OBJ,                 &
       DISK/SUP.OBJ,                    &
       PATH/LIB.OBJ,                    &
       DRIVERS/MOIS.OBJ,                &
       DATA/LIB.OBJ,                    &
       WELD/PROG.OBJ,                   &
       DRIVERS/WELD.OBJ,                &
       /LIB/RMX86/RPIFL.LIB,            &
       /LIB/RMX86/IPIFL.LIB,            &
       /LIB/NDP87/8087.LIB              &
TO PROG/APPL.LNK MAP PRINT(PROG/APPL.MP1)
```

```
;*****************************************************************
;*                                                               *
;*      THIS FILE TAKES THE FILE PROG/APPL.LNK FROM LINK86       *
;*      AND LOCATES IT ABSOLUTELY. THE STACK SIZE IS ZERO        *
;*      AS THE NUCLEUS TAKES CARE OF STACK ALLOCATION. THE       *
;*      CODE IS LOCATED AT 1C000H WITH THE DATA ETC ABOVE.       *
;*      THE LOCATOR MAP IS AT PROG/APPL.MP2, CONTAINING          *
;*      ALL THE SYMBOL AND PUBLIC ADDRESSES.                     *
;*                                                               *
;*****************************************************************
;
LOC86                                           &
PROG/APPL.LNK TO PROG/APPL                      &
SEGSIZE    (STACK (0))                          &
ORDER      (CLASSES (CODE,DATA,STACK,MEMORY))   &
ADDRESSES (CLASSES (CODE (1C000H)))             &
MAP PRINT (PROG/APPL.MP2)                       &
NOINITCODE                                      &
PRINTCONTROLS(SYMBOLS,PUBLICS,NOLINES)
```

```
;*****************************************************************
;*                                                               *
;*     THIS FILE DELETES THE SUPERVISOR MODULE FROM              *
;*     THE APPLICATION SYSTEM AND REPLACES IT WITH               *
;*     THE NEW ONE IN PROG/APPL.                                 *
;*     IF A DISK IS IN :F1: IT ATTEMPTS TO REPLACE ANY           *
;*     FILE ON IT CALLED :F1:SYSTEM/RMX86.                       *
;*                      ******************                       *
;*                                                               *
;*****************************************************************
;
LI586
DELETE PROG/APPL.SYS(SUPERVISOR)
ADD PROG/APPL TO PROG/APPL.SYS
LIST PROG/APPL.SYS
EXIT
COPY PROG/APPL.SYS OVER :F1:SYSTEM/RMX86
```