Computer Aided Flow-Turning

by

ALA HASSOON MOHAMAD B.Sc.

A thesis submitted to the COUNCIL FOR NATIONAL ACADEMIC AWARDS in partial fulfilment of the degree of MASTER OF PHILOSOPHY

Sponsoring Establishment: Department of Mechanical and Production Engineering
Sheffield City Polytechnic
Sheffield 1

Collaborating Establishment: C.W. Fletcher & Sons Limited
Sterling Works, Arundel Street
Sheffield S1 1DP

July 1988

ACKNOWLEDGEMENTS

I should like to thank Dr.E.A.Vallis Head of Mechanical and Production Engineering Department and the Authorities of Sheffield City Polytechnic for allowing this work to be carried out.

I should like to take this opportunity to acknowledge the invaluable guidance and continual assistance given by Prof.M. SARWAR under whose supervision this work was carried out.

The author gratefully acknowledges his appreciation to Prof. M.S.J.HASHMI on the overall context of this research and preparation of the final report.

The author also wishes to thank Dr.J.R.TRAVIS from the Electric and the Electronic Department for his part on the electronic design and software development side.

Mr.R.RODDIS for his assistance, encouragement and patience in the program design, SDK-85 board modification and in the understanding of the Intel development system.

Mr.J.ASHBY from the Automation Advisory Service Department for his advice and assistance in the hydraulic circuit design, component selection and testing procedure.

Mr.L.EVANS for constructing the hydraulic rig and his useful comments.

Mr.J.STANLEY for his assistance in the CAD/CAM system drawings and his useful suggestions.

The technical assistance offered by Mr.R.Teasdale and his staff was much appreciated and particular thanks go to Mr.S.Leigh, Mr.T. O'Hara and Mr.R.Wilkinson for their assistance in manufacturing and setting up the experimental equipment.

ABSTRACT

Computer Aided Flow-turning

A. H. Mohamad

The work undertaken in this research is concerned with the flow-turning process and its control using microprocessor technology. The research centres on the design of a suitable flow-turning process controller in which hardware and software are integrated together leading to a successful realisation.

Microprocessor software has been developed to provide a user friendly interface with the operator. This was written in PL/M 80 which was subsequently compiled into machine code for execution on a modified commercially available single board computer. Interface circuits were designed and transducers and actuators selected to enable this computer to be linked to a flow-turning rig which itself was custom designed to facilitate automatic control. Considerable development work was devoted to the integrated system to produce a working controller.

The endeavour was rewarded with success and a working controller has been accomplished. Experimentation and testing of the real specimens followed and the results obtained are tabulated.

CONTENTS PAGE

CONTENTS (cont) PAGE

CONTENTS (cont) PAGE

LIST OF FIGURES PAGE

CHAPTER 1

CHAPTER 2

CHAPTER 3

CHAPTER 4

CHAPTER 5

CHAPTER 6

LIST OF PLATES PAGE

Chapter 1: Flow-turning process definition and history

1.1 - The flow-turning process

Flow-turning is a process by which a metal blank is deformed to a final axisymmetrical component shape using a tool (usually a roller) to press it against a former. A reduction in wall thickness occurs.

Briefly it can be likened to the cold rolling of sheet metal, with the difference that the roller is in single point contact instead of in line contact(1).

The pre-requisite for flow-turning is the need for a final axi-symmetrical component, provision of a mating mandrel, a forming roller and a means by which the movements of this roller can be made to describe the required profile(2).

As no metal is removed, it follows that the volume of material remains unaltered, so in order to calculate for a conventional blank size, the change in wall thickness is the only important information required. The process is illustrated in fig 1.1 and plate 1.1.

There are two types of flow-turning:-

1- Tube spinning (flow forming)

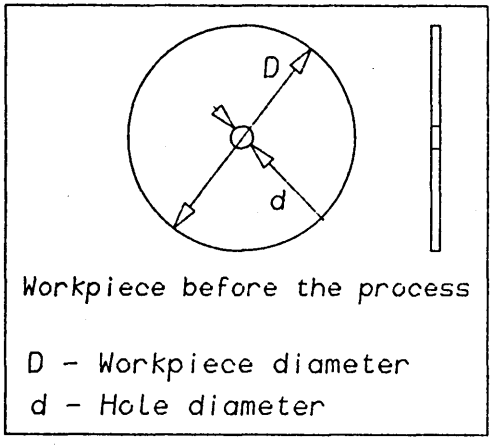
2- Shear spinning (shear forming)

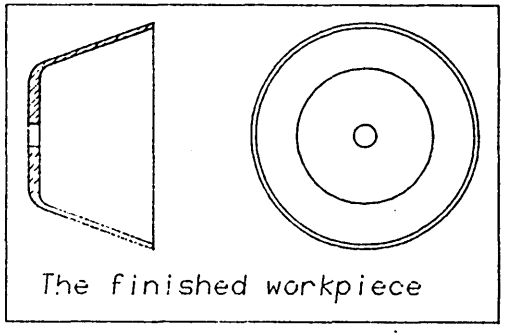
In tube spinning, as the name implies, the pre-form is tubular and is flow-turned to the required dimension by continuous point extrusion which increases the length of the tubular section by reducing the wall thickness.

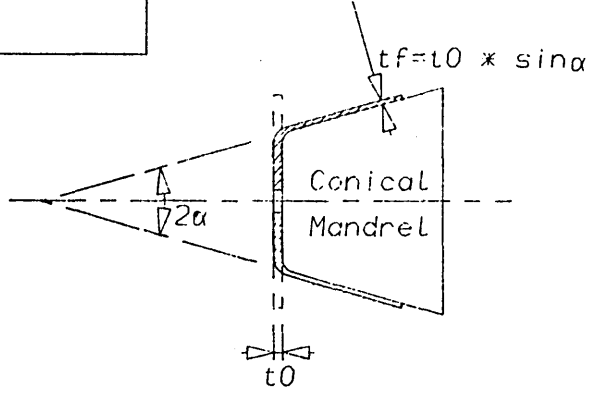
In shear spinning the pre-form is a flat disc which is formed into progressive cone shapes by displacing the metal along the shear planes running parallel to the centre line of rotation.

THE AIM:

To develop a working controller for the flow-turning process and then produce an axisymmetric components out of a circular blank







t0 - Original thickness
tf - Final thickness
α Cone angle

FIG 1.1: Workpiece before and after the process

The relationship between the length of the cone and the wall thickness follows a sine law; that is, the wall thickness Tl of the finished part is equal to the thickness T of the flat blank multiplied by the sine of one half of the included cone angle. It is, therefore, essential to start a pre-form at a specific thickness according to the thickness of the wall required in the finished cone(3).

It is impossible, in conical flow-turning, to reduce the wall thickness by more than shown by the sine-formula, since the blank would offer too much resistance as the displaced material flowed into the increasing diameter. An increase in the roller pressure can cause the 'ring' of piled up metal to impede the tool and the strength of the blank material can be exceeded, thus leading to a tear(1).

Since in most cases a single roller-pass only is required, complex systems to control the roller path are not usually necessary. Two methods of controlling the roller are in common use:

1- Through variation of roller in-feed cylinder pressure.
2- Through a hydraulically operated copying device and template.

Of the two systems, the first is the simpler to set up but requires a certain measure of skill in operation, particularly in judging the starting point at which the required set pressure is introduced to the roller in-feed cylinder to commence the forming operation(4).

The technique can be used to produce angles of between 12 and 80 degrees from the centre line of mandrel rotation. Below 12 degrees the thinning is excessive and above 80 degrees the amount of working is insufficient to ensure metallurgical stability(5).

With shear forming, in contrast, the metal is stretched beyond its elastic limit, thereby introducing new properties. This procedure

involves stresses being applied to the material to stretch it beyond its elastic limit, but below its yield point, and produces plastic deformation, which can be recognised in the characteristic features of shear-formed material(4).

This process can be used for components where the cone angle lies between 8 and 80 degrees. However, for single-pass forming, the limiting minimum angle lies around 13 degrees. For components with an angle between 8 and 13 degrees, a two-pass forming operation is necessary(4).

1.2 - The advantages and disadvantages

In general, it can be stated that whenever the cone angle lies in the range 13 to 80 degrees it is worth investigating the feasibility of the shear-forming technique(4).

Most metals can be formed by this process and although heat is sometimes applied throughout the cycle, it is not required for steel alloys and most nonferrous metals(6).

One main advantage of this process, which makes it suitable for many applications, is the improvement obtained in the mechanical properties of the workpiece material, namely an increase in hardness and ultimate tensile stress. However this is usually accompanied by a decrease in ductility. Other advantages are speed of operation, little waste and economy, especially for smaller batch manufacture. Moreover, the tooling and setting requirements are comparatively simple and relatively inexpensive. Recent applications of the spin-forging process are the manufacture of conical parts to a high degree of dimensional accuracy which include, for example, radar reflectors, components for jet and turbine engines, satellite nose cones, rocket components, truck wheel rims and parts for nuclear reactors(7).

Tolerances in thickness and inside diameter of between +0.0508 mm are possible and very smooth surface finishes are produced, so that in many cases finishing or polishing operations can be reduced or eliminated(3).

Tooling for flow-turning is remarkably low in cost, only about 10% that of deep-drawing dies, and long tool life is obtained. Flow-turning is especially suitable, therefore, for the production of small quantities of components and these can be produced more economically than by deep-drawing, forging or machining(3).

Most metals can be formed by flow-turning provided that sufficient pressure can be applied. These include aluminium alloys, copper alloys, mild steels, many stainless steels, high temperature alloys, Monel, Iconel and Nimonics. Some metals, such as molybdenum and magnesium, are difficult to spin unless they are heated and most titanium alloys require to be heated(3).

Since the actual deformation of the material takes place only at the point of contact between the roller and the blank, the remaining material remains free of any stresses. This characteristic of shear forming allows a very much greater degree of deformation of the material to be achieved than is possible with other processes. In many cases, finished components can be produced in a single operation, when the use of other techniques would involve several(8).

The shapes that can be produced successfully and economically by flow-turning fall into four classes:-

1- Straight-wall cones

2- Curvilinear-wall shapes

3- Hemispherical or elliptical shapes

4- Straight-wall shapes

Straight-wall cones can be made from flat blanks, the maximum thickness depending on the material and the power available.

Curvilinear shapes can be flow-turned from either flat blanks resulting in a gradual thinning of the walls, or from tapered blanks which give an almost constant thickness of wall of the thinnest component.

Hemispherical shaped parts with a constant wall thickness can be flow-turned from tapered flat plates. When the machine is fitted with contour tracing equipment, tube and curvilinear wall sections with multiple diameters can be formed(3).

Parts having a cone angle of less than 60 degrees require a conical preform. A reduction in wall thickness of up to 80% is possible although, in most cases, the reductions are much smaller(6).

Standard machines are typically made to spin parts 127 cm long and 106.68 cm in diameter. However, there is no upper size limit, capital investment and requirements being the only limiting factors(9).

Flow-turning is generally practicable only for components that are of hollow, conical or cylindrical shape. Another limitation is the thickness of the material that can be formed. Power requirements for flow-turning depend on the type of material or alloy, the included angle, diameter and thickness of the workpiece, the percentage reduction in wall thickness and the rate of feed of radius of the roller. Stainless steel and Nimonics up to 19.05 mm have been successfully shaped as well as aluminium and brass alloys up to 38.1 mm thick. The smallest angle considered practicable from the blank is generally considered to be 30 degrees, but smaller angles down to 15 degrees have been turned. Ductility or elongation of the material is reduced substantially after flow-turning(3).

The dimensional accuracy obtained will depend on a number of factors including

1- Variation in the thickness of the original material.

2- The severity of the operation.

3- The rigidity of the machine - no machine is absolutely rigid and will deflect under the application of considerable power.

There is a limiting point at which the molecular structure of the material will break down and reduction passes can only be taken below this limit, the corresponding angle being referred to as 'critical'. If a shape beyond the critical angle is required, further passes become essential(10).

1.3 - Historical background of flow-turning

The process of flow-turning was probably first used in the United States of America to produce cream separator cones. Subsequently the famous television cone was produced in great quantities in this manner(9).

Flow-turning is said to have been invented and patented in the early years of this century. The process has been developed since 1947 and development was particularly intensive at the time of American rearmament during the Korean War, when speed of production was of prime importance(3).

During the spinning of cones, the initial wall thickness of the material remains unchanged whereas, in the spin-forging operation, the final thickness is dictated by the sine of the cone semi-angle. Also in spin-forging, there is no appreciable change in the radial position of each element from its original position in the circular blank. The maximum diameter of the formed cone thus remains approximately the same as the original blank diameter(7). (See the illustration in fig 1.2.)

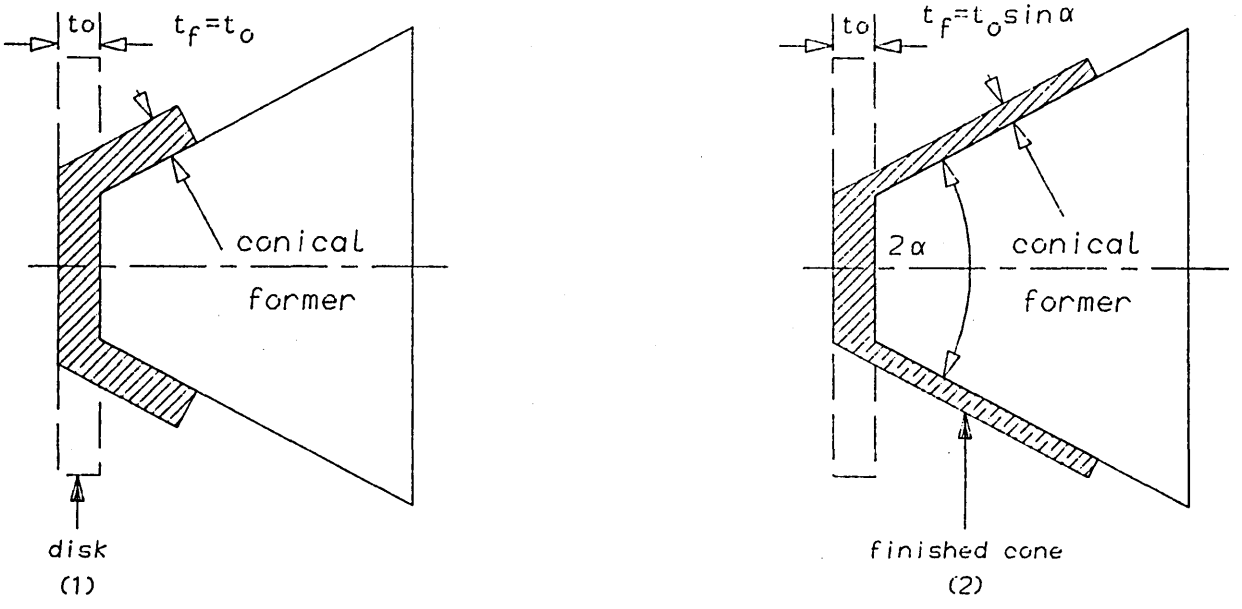


FIG 1.2: Illustrating the forming of a sheet metal cone by

(1) the spinning process
(2) the flow-turning process

In metal spinning, the metal can normally be reduced only slightly in thickness and, as the reduction cannot be predetermined exactly, the finished parts are not always exactly uniform and close tolerances cannot be maintained consistently. Flow-turning is, however, a much more exact process. As no metal is detached during the process and the volume of the actual material remains the same, very large reductions in thickness are accomplished with corresponding increases in total area. The metal undergoes partial shear deformation under the high pressures employed as it is squeezed ahead of the roller and displaced parallel to the centre line of the part being formed. The metal is obtained from the thickness of the blank and not the diameter of the blank, as in metal spinning(3).

The first development from simple hand spinning was the provision of mechanical assistance either to reduce the physical effort required by the spinner or to increase the gauge diameter and range of metals spun.

Mechanical spinning always uses roller tools mounted on a cross-slide carried on a longitudinal slide mounted alongside the machine. The simplest method of moving the tool is by means of a handwheel operating on screw threads. A recently reported variant of this system is the attachment of stepping motors to drive the feed screws.

The more usual method of providing mechanical assistance to the tool is to move the compound slide carrying the tool by hydraulic means.

For automatic spinning, the objective is to program the roller tool in pre-determined paths to convert the blank into a spinning. Several benefits can accrue from such automatic sequencing:

First, there is product reproductibility which, always a problem with a manually produced article, should be much better. Similarly with correct adjustment, surface finish may also be improved.

Second, with the increasing scarcity of skilled and experienced spinners, routine work can be performed by less-skilled personnel.

There have been three main lines of development of automatic spinning, viz

1- Self-learning systems

2- Template systems

3- Fully CNC systems

These are detailed below:

1- This system entails a spinner producing a component using a mechanically or hydraulically-assisted lathe. In doing so, he exercises his normal skills. When he is satisfied with his procedure, his operations are recorded. The play-back of this record then actuates the lathe into reproducing the same movements.

2- A pre-cut template can also be used to predetermine the path of the roller. For the final spinning passes, this template will normally have the profile of the required component. A succession of pre-form passes can be achieved by using either a series of templates or a swivelled template system.

When a series is used the sheet-metal templates are individually cut and checked on the lathe before the next in succession is cut. Either the template carrier or the template sensor is advanced to the next position after each traverse.

In the swivel template system a template shaped to the final form of the work-piece is fixed to the template carrier. The wheeled template tracer can be adjusted to give a predetermined gap between spinning roller and workpiece on the final passes. A second pivoting template is mounted below the fixed template and is retracted by a hydraulic cylinder, a series of cams providing the necessary steps. When the tracer is fully retracted, these follow the final form template. Operation is through electrical sequential control actuated by micro-switches.

3- The next logical development in automatic spinning was to produce an 'electronic' template machine with the path data and information for auxiliary actions stored on punched tape(11).

With a semi-automatic machine, it is only necessary to remove the finished workpiece and to put in the new blank. A machine can properly be called automatic when all manipulations are mechanized. Completely automatic machines feed the blanks from magazines and finished components are also removed. The development of spinning and flow-turning lathes to automatic machines has been made possible through the application of hydraulic actuation and this development can be considered as having taken place in stages.

First the centre was tightened up hydraulically. The next step was the hydraulic movement of the roller forming lever, operated by a hydraulic cross slide(12).

A lathe which can do both spinning and flow-forming has been developed by Joshua Bigwood & son Ltd, Engineers, Wolverhampton. It has the capability of making a wider range of items than was

previously possible by flow-forming, and can produce cylindrical shapes from a single blank of metal in one operation. The lathe can also produce items such as stainless steel sinks, buckets, bowls and is particularly suitable for the manufacture of stainless steel vessels and storage containers for chemicals, pharmaceuticals, bakery and dairy products. Particular success has been achieved with stainless steel, which has hitherto proved particularly difficult for this type of work. However, mild steel, aluminium, copper and brass can also be formed with complete satisfaction(13).

Today, with the assistance of electronic circuits and switching, all spinning and flow-turning manipulations can be initiated(12).

The number of applications this process can be used for is so wide that further rapid development seems likely.

1.4 - Machines for flow-turning

For flow-turning, a number of makes of machine in a wide range of sizes is available, and most of these resemble heavy duty lathes. What is believed to be the most powerful flow-turning machine in Europe is the Cincinnati "Hydrospin" installed in the machine shop of Bristol Siddeley Engines Ltd at Patchway. It will accommodate components more than 121.92 cm long and up to 106.68 cm in diameter. The headstock is driven by a 200 hp dc motor and a force of 28 tons at each tool ring makes possible the spinning of pre-forms 20.63 mm thickness in high tensile heat-resisting steels and similar materials. With shear spinning, however, the pre-form thickness has been limited to approximately 15.875 mm. The machine was supplied with a hydraulic system working at 3,000 p.s.i., as this was considered to be the minimum pressure to handle the thick section components to be formed.

Much bigger and more powerful machines for flow-turning can be made(3).

In practice, the longitudinal slide is fixed approximately parallel to the contour of the component and the pressure of the roller in-feed cylinder (or top side) set to give the required degree of material deformation(4).

Standard machines have a force on the cross-slide of 50,000 lb and the tailstock a force of 30,000 lb. The total load against the headstock and its bearing is 130,000 lb. The speed of the headstock is currently from 100 to 450 rev/min. It is becoming evident that higher speeds will be advantageous, which will of course increase the bearing design problem in the headstock and rollers. The rollers and tool rings are driven by contact with the work and the mandrel(9).

1.5 - Coolants and lubricants

It is generally necessary to use both a coolant and a lubricant in flow-turning. A soluble oil may be used for both purposes in some cases (such as for low pressure spinning of thin ductile materials) but when the material is difficult to spin and high pressures are necessary, it is usually better to use separate cooling and lubricating media(3).

Lubricants are used on the mandrel to prevent scoring of the part when it is removed from the mandrel. Any type of paste lubricant is satisfactory. The lubricant used on the outside of the part is a different matter: the type employed depends on the type of material being spun. The lubricants established for most drawing or rolling applications are also suitable for spinning. The other requirement for the lubricant is that it must not be dissolved by any coolant fluid used.

It must be noted that the material used for spinning must be clean, as inclusions of dirt or slag in the blank will cause cracking or splitting of the part as it is formed. Normally, tool marks or scratches will not cause difficulty provided that the forming operation is not too severe. However, if the preform is of critical shape and the spinning operation is very difficult, tool marks or scratches may cause cracking of the part during spinning, due to stress concentration set up by sharp corners(9).

1.6 - Speed and feed rates

The surface speed can affect the metallurgical properties of the part produced. Best results are usually obtained at speeds from (1000-2000) surface feet per minute. Although this range of speed requires more power than lower speeds, the properties and results obtained at lower speeds are not desirable.

The rate of feed influences the finish of the part: the finer the feed, the smoother surface obtained. Usually the best results have been obtained with feed rates from 0.0508 - 1.27 mm per revolution. Where smoothness is not the most important factor, feed rates of 0.762 - 1.27 mm per revolution are often preferred(3).

1.7 - Background to the design of the rig and scope of the present work

Whilst normal practice is to control the roller movement in flow-turning either by templates or NC machines (see section 1.4), a more reliable and cheaper control could be achieved by introducing micros into the flow-turning. Microprocessor control is becoming less expensive and is well within the budget of small companies. This is a

trend which will undoubtedly accelerate as software and hardware are becoming cheaper and cheaper.

This work is aimed at developing a suitable microprocessor based controller to replace the existing control techniques.

The principal objectives of the work programme were to

(i) Highlight microprocessor application areas, and recommend an appropriate microcomputer. Assess the capabilities of the software development systems available.

(ii) Develop and implement the flow-turning process rig.

(iii) Formulate microcomputer programs for the control of the required contours (conical and parabolic shapes are implemented).

(iv) Test the program on the rig and ascertain that it performed its intended function.

(v) Assess the overall capabilities of flow-turning and the potential practical applicability of the rig.

Chapter 2: Microprocessor applications and selection

2.1 - Introduction

This chapter is concerned with the selection, application and implementation of the microprocessor element in the control system.

The first task was to choose a microprocessor suitable for controlling the flow-turning process. Also in this chapter, some of the application areas are highlighted to illustrate how the microprocessor has become an essential ingredient in almost all sectors of life. Microprocessor-aided flow-turning is contrasted with other manufacturing techniques in terms of efficiency, initial cost and degree of accuracy after which it was decided to adopt the new technology as an alternative to the existing methods. The choice of microprocessor was carefully considered in this research in view of availability and associated expertise.

Some of the tools used in developing the flow-turning process software are described briefly to demonstrate how considerable time and cost can be saved. Also, the language used (PL/M 80) was underlined and explained in some detail.

The use of a microprocessor may in itself promote a change in present manufacturing techniques for this process or it may pave the way for other further research. This research was directed towards implementation of a microprocessor in the flow-turning process.

2.2 - Examples of common microprocessor uses

There are many current application areas for microprocessor, and to some extent these indicate the likely course of future developments. It is useful for manufacturers and users to know in what fields investment will be fruitful and what are the practical application

opportunities. Major areas in which microprocessors were likely to make an impact are listed

1- production monitoring / recording

2- automatic warehousing

3- distribution

4- retail trade (point of sale / stock recording)

5- banking transactions

6- tickets /reservations / passenger movements

7- hotel / restaurant booking

8- hospitals (patient records / monitoring/ analysis)

9- road control (traffic lights / hazards)

10- building control (heat / light / lifts / fires)

11- office systems (word processing / automatic filing)

There are many processes and phases in industrial activity from the handling of basic raw materials, through the design and manufacture of products. All the associated tasks, processes and production control lend themselves to computerisation.

As far as the engineering industry is concerned, processes such as flow-turning, metal-spinning and tube-bending are amenable to microprocessor control. Micros can also help to optimise forging, cold-forming and extrusion operations(14).

2.3 - Using the microprocessor as an alternative method of tool movement control

As mentioned earlier in chapter 1 section 1.3, there are various well-known methods of controlling tool movement which can be used satisfactorily and efficiently. The techniques use specialised

flow-turning machines (with their supporting hardware and software) which are usually expensive. Some of these techniques involved using a medi computer or a CNC lathe to store the tool movements, as mentioned previously in section 1.4.

In this research, a lathe was converted to accommodate the flow-turning process and can also perform spinning. Certain modifications were carried out, which included the hardware (which is elaborated on in chapters 3 and 4) and software (as explicated in chapter 5).

In this study, a controller was conceived, fabricated and installed on the lathe to perform the flow-turning process. A single board microcomputer system (SDK-85) was utilized with the necessary interface to communicate with the designed rig. A program was written to enable the roller to follow the required shape(s) (conical or parabolic).

The rig design was simplified by using a single hydraulic cylinder to actuate the tool (in the y-axis) while employing the screw threads on the lathe to move in the x-axis. In this way, the added features can be implemented on a second hand lathe of appropriate size and cost, thus keeping the overheads to a minimum. The price of the added features was around £1300 in total and, assuming an estimated lathe value of £8000, thus the sum total is £9300 which is probably less than 1/7 the cost of a new flow-turning machine.

2.4 - Selection of the microprocessor

The microprocessor selected should be able to perform the following

1- Fulfil the arithematic manipulations which involve addition, subtraction, multiplication and division with adequate accuracy (whether it is 8 or 16 bit processor).

2- Possess adequate speed to process data which include input/output operations (the input of transducer movement i.e ADC, the shaft encoder input, which is the roller tool position, and the output of digital values to the DAC).

3- Meet good language facilities, with floating point variables if possible.

The two possible available models are the SDK-85 or the SDK-86 microcomputer boards.

On the one hand, the SDK-85 board has the 8085 as a CPU, which is an 8-bit general-purpose microcomputer that is very cost-effective in small systems because it has a low hardware overhead requirement. At the same time it is capable of accessing up to 64K bytes of memory and has status lines for controlling large systems. The SDK-85 microcomputer is used mainly is small control applications. It has enough I/O ports, yet has some language limitations (there are no floating point variables, only integers, which limits the accuracy).

On the other hand, the SDK-86 board has a newer, more powerful 16-bit processor with good language facilities (floating point variables). However the supporting hardware and software were not available to the researcher within the Mechanical and Production Department of Sheffield Polytechnic.

The microprocessor to be selected must fulfil the three requirements mentioned above for the flow-turning process. In addition, other factors have to be considered, such as expertise. Both the SDK-85 microcomputer systems satisfy the three criteria. However, because the former has been accessible for several years, expertise is already obtainable, an important incentive when undertaking such research. Moreover, all the difficulties associated with using the

hardware and software are well known. Thus the SDK-85 was chosen as the most suitable device.

Some of the development aids available were

1- Intellec series II microcomputer development system.

2- In-circuit emulator (ICE-85).

3- PL/M 80 programming language.

4- Universal PROM programmer.

5- ISIS-II diskette operating system.

These aids are discussed in the following section

2.4.1 - Intellec series II microcomputer development system

The Intellec series (model iMDX 225) is a complete microcomputer development system integrated in one compact package. It includes a CPU with 64K of RAM memory, 4K bytes of ROM memory, a 2000 character CRT, detachable full ASCII keyboard with cursor controls and upper/lower case capability, an integrated 250K byte disk drive, plus an iMDX 721 dual disk drive system. Plate 2.1 shows the Intellec development system.

2.4.2 - In-circuit emulator (ICE-85)

This is a debugging device which is available with the development system. It is invaluable because of its ability to test software without 'blowing' an EPROM. This works by allowing a program that exists in object code on a disk file, to be loaded and run in the development system's CPU, under the control of ICE. The ICE software

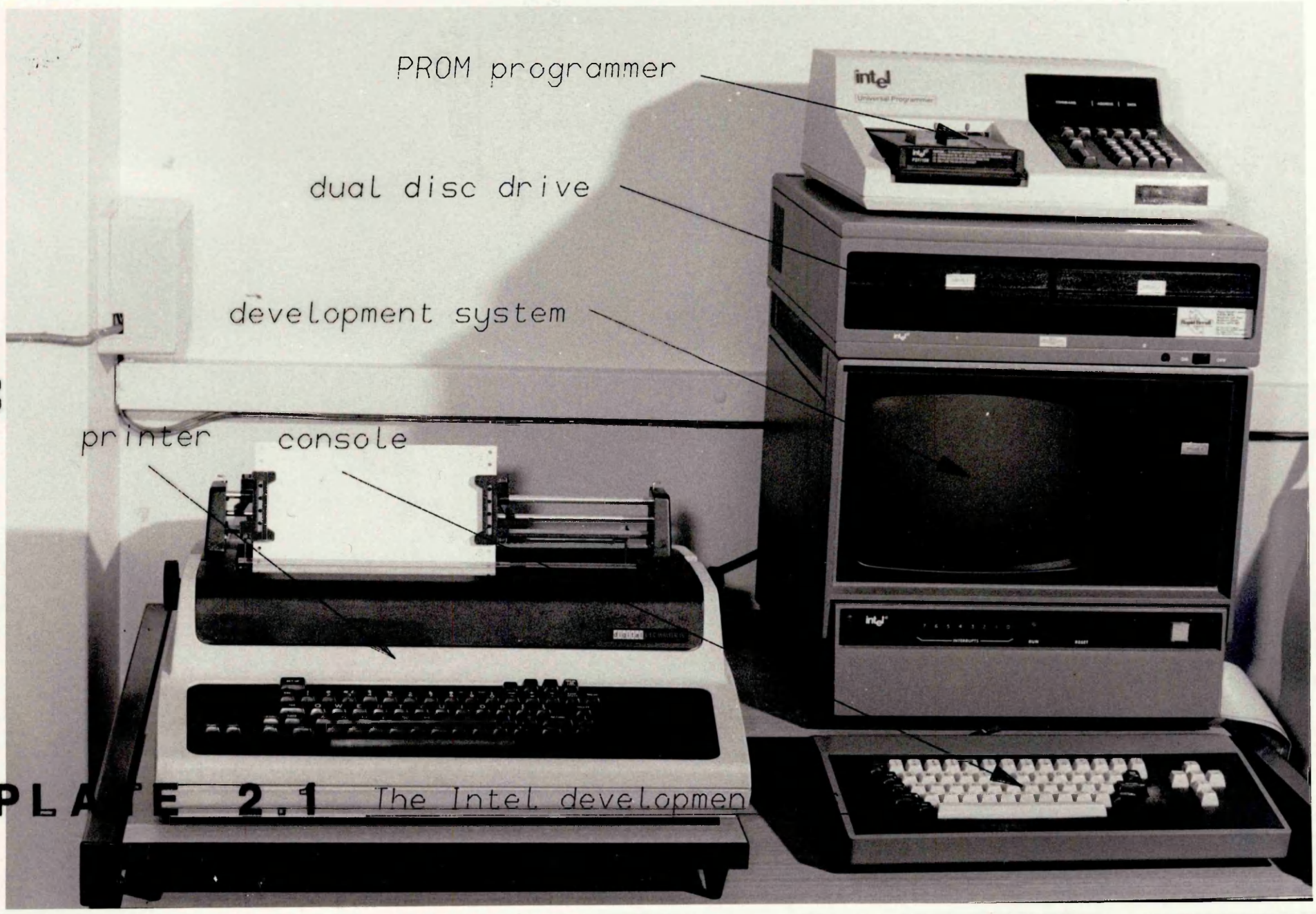


PLATE 2.1 The Intel developmen

is a sophisticated monitor which controls the execution of a program, with a set of commands to facilitate debugging of the program under development. These commands permit such functions as insertion of 'break points' into the program.

The break points can be specified as either addresses or symbolic addresses, or the break point could be specified as a certain command or operation, for example, the calling of a sub-routine.

When the execution is terminated, the ICE permits investigation as to why the program has stopped. The ICE will produce an output of the last 'n' instructions executed, or a listing of the status registers and memory locations; this also applies to controlled terminations(15).

2.4.3 - PL/M 80 programming language

PL/M 80 is a high level programming language for use on the Intellec microprocessor development system. It is easy to learn, facilitates rapid program development and debugging, and significantly reduces maintenance costs.

PL/M 80 is an algorithmic language in which program statements can naturally express the algorithm to be programmed.

The PL/M 80 compiler converts 'free-form' PL/M programs into equivalent instructions for the 8085 processor. Fewer PL/M 80 statements are required for a given application than would be required with assembly language or machine code.

The major benefits of using PL/M 80 for the development of the flow-turning include the following

(i) Low learning effort.

(ii) Earlier project completion - PL/M 80 increases programmer productivity.

(iii) Lower software development costs - because of (ii) above.

(iv) Increased reliability - because a simply stated program is more likely to perform its intended function.

(v) Easier enhancement and maintenance - because it is readable and understandable.

The PL/M compiler accepts source programs, translates them into object code, and produces listings. After compilation, the object program may be linked to other modules, located to a specific area of memory, then executed. Fig 2.1 illustrates a program development cycle where the program consists of two different types of modules (PL/M and assembly language) see (16), (17) and (18).

2.4.4 - Universal PROM programmer

The UPP-103 universal PROM programmer is an Intellec system peripheral capable of programming and verifying various EPROMs.

Programming and verification operations are initiated from development system console and are controlled by the universal PROM mapper (UPM) program(19). Plate 2.1 shows the PROM programmer.

2.4.5 - ISIS-II Disk Operating System

This is a well established operating system very similar to CP/M. Various application packages are available to run under the system such as assemblers, screen editors and compilers. The disk operating system can be seen in plate 2.1.

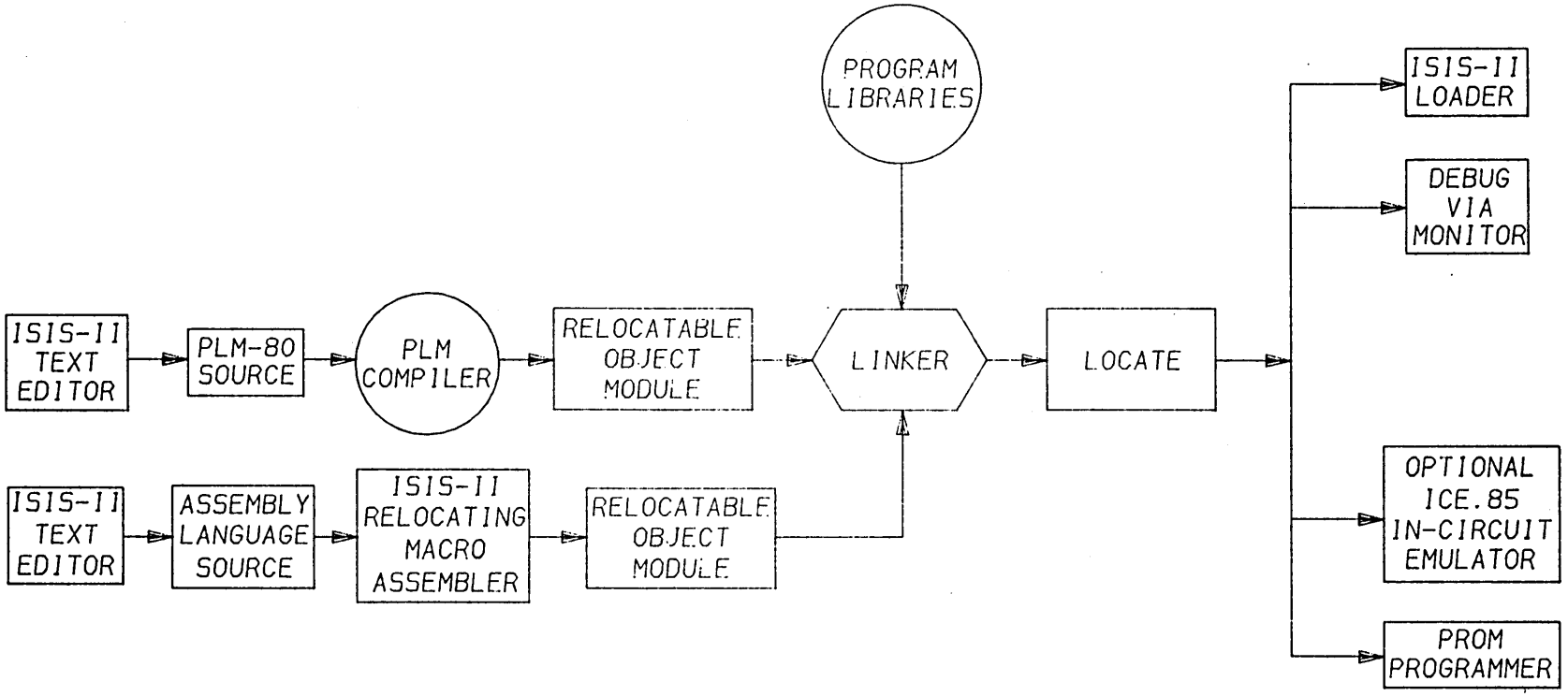


FIG 2.1: programme development flow using ISIS-II disk operating system

2.4.6 - Intel SDK-85 (System Design Kit)

This is a single board microcomputer, based on the 8085 microprocessor with:-

2K bytes of ROM

2K bytes monitor/ROM adaptability

512 bytes of RAM

76 bits of Parallel I/O

serial I/O - limited to 110 bauds

See plate 2.2 for the SDK-85 board.

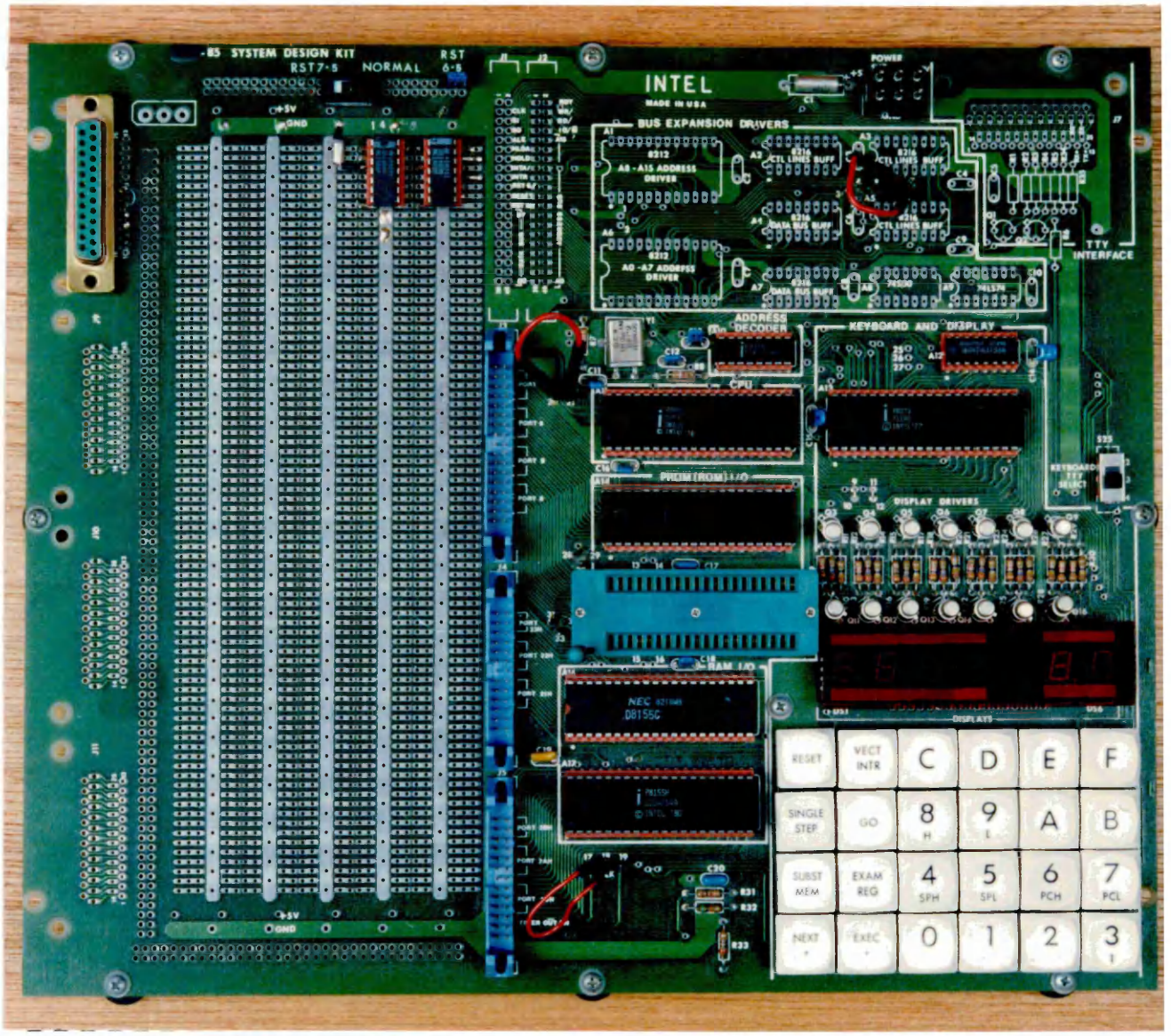


PLATE 2.2 *The SDK-85 board*

Chapter 3: Equipment description and preliminary testing

3.1 - Introduction

In this chapter a detailed description of the apparatus used is given and modifications to the existing rig are explained.

Also, the preliminary testing carried out during the onset of the research is summarised and the results obtained are shown in the form of graphs.

The parts made in the workshop including the roller and formers are explained in detail with drawings.

Later, in chapter 6, the ranges of the suitable spindle speeds and carriage feeds were selected after consideration of software requirements.

3.2 - Description of the experimental equipment

A lathe was needed to perform the flow-turning process. The one chosen in the workshop was manufactured by Dean Grace & Smith with 33 cm swing. There were 12 spindle speeds available, ranging from 16.8 to 750 rpm and 48 carriage feed rates, ranging from 0.226 to 12.7 mm per revolution. Plate 3.1 shows a general view of the equipment used. It was assumed that with this type of lathe the process can be carried out although the lathe is not specialised for this job but can be converted to handle this task.

3.3 - Modification to the existing rig

In order to make flow-turning possible, some components had to be made and added to the lathe. Alterations included the following:-

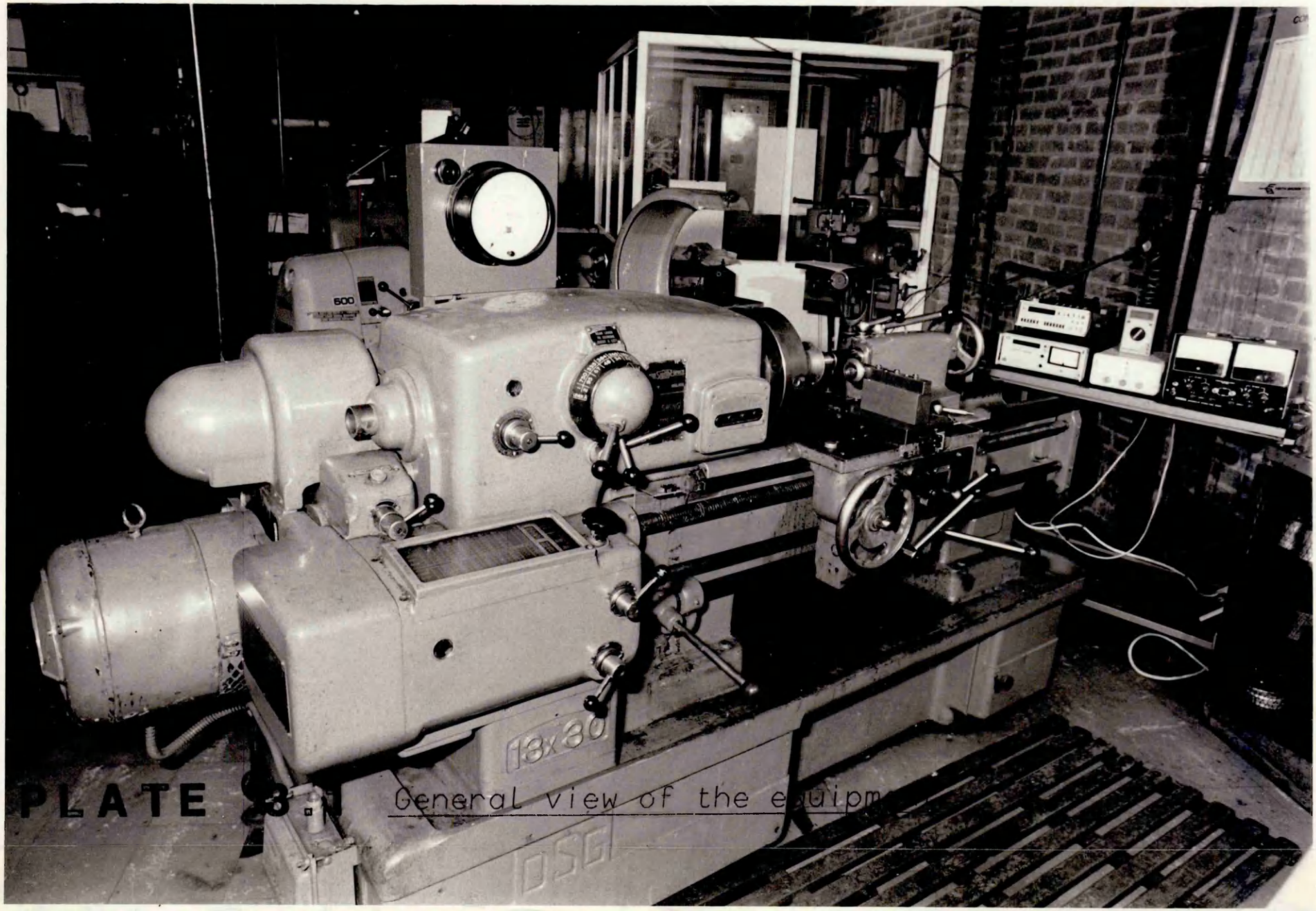


PLATE 3.1 General view of the equipm

1- Modifications to accommodate the workpiece and the roller (these will be explained in this chapter).

2- Other modifications, which include attaching the transducer to the cross-slide so as to measure roller movement, selecting and fabricating the ADC and the DAC boards, mounting the shaft encoder on the leadscrew end and making the counter board; installing a relay and the necessary circuitry on the lathe electric starter so that the microcomputer SDK-85 could remotely control the ON/OFF starter. These will be dealt with in chapter 4 (the hardware).

The tooling for shear spinning consists of the mandrel, tool rings or rollers, and the tracing templates. As the mandrels must be harder than the material in the finished part, they must have a compression strength of 200,000 lb/in square. Tool steel that can be hardened to this or a higher strength is easily obtained, but for large mandrels, the cost is significant. A very satisfactory material at lower cost is high-strength nodular iron which may be cast to shape and hardened to the required strength.

The tool rings or rollers must be of high-grade tool steel in order to obtain mimimum hardness of 62 Rockwell C. This hardness is necessary to resist wear and scuffing(9).

To position the workpiece in place, it was considered best to fix by using the tailstock as a support. The tailstock was provided with a dummy rotating centre which was made specifically for this purpose to hold the blank against the mandrel end by exerting pressure on it. The centre was provided with a pig in the middle, which passes through the centre hole of the workpiece.

One thickness of commercially pure aluminium sheet (BS 1470/SIC) of 1.6 mm was used. The aluminium sheets were cut into discs of 100 mm

diameter, each with a hole drilled at the centre. High pressure grease was used as the lubricant throughout the study.

Chucks or formers are often made of cast iron, but when forming high-tensile materials or producing large quantities, chucks of hardened and ground steel may prove more economical. Generally the quality of surface finish obtainable is related to the surface finish on the chuck itself, imperfections in the chuck frequently being transferred to the work under the very heavy pressure applied(10).

The former was made in the shape of a cone with 10 degrees semi-angle (see fig 3.1), and was used in the initial testing. It was made of steel EN8, which can withstand the compressive pressure imposed by the roller. Another former with 30 degrees semi-angle was made at a later stage of the research for the final testing (see fig 3.2). This was however made from Meehanite.

Meehanite castings satisfy the requirements for a good mandrel material. Such castings are noted for their fine grained structure, which ensures dependability and freedom from casting defects. A uniform structure provides good machinability with a compression strength of 80 ton/in square in the as-cast condition. Moreover, Meehanite type 'GA' is able to resist the extreme high level of external forces applied to the surface of the mandrel during spinning operations(20).

At a later stage of the research it was decided to implement the parabola contour (see figs 3.3 and 3.4). A parabolic former was made in a later stage of the work on a CNC lathe, retro-fitted with an AUDIT M268 controller. The program was generated using software package called 'path turn' supplied by 'path trace' Ltd. The parabola was generated with the loop command within the geometry section, this was then processed by the machining section to give tool movements in order to cut the parabola. This data was then post-processed to suit

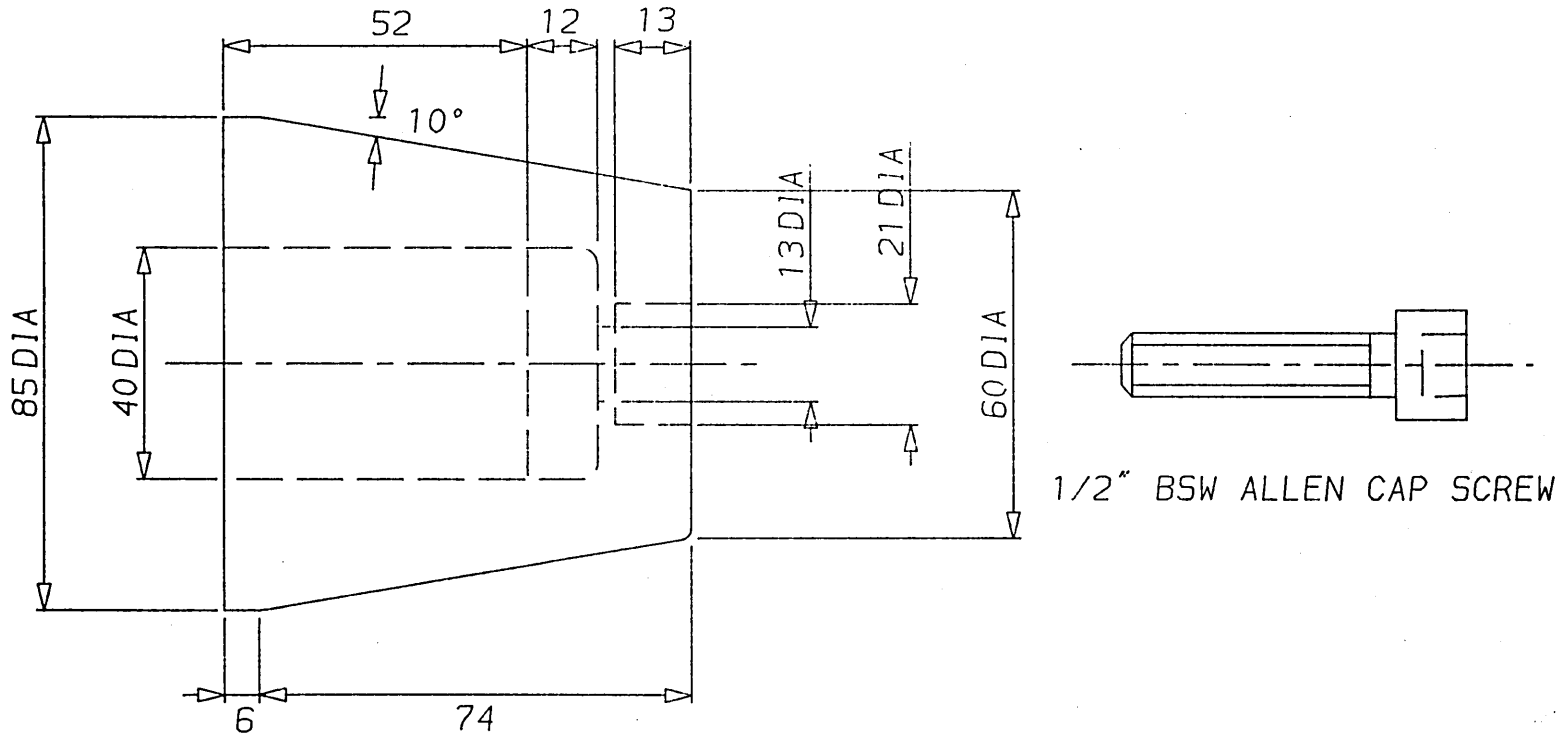


FIG 3.1: Preliminary testing former

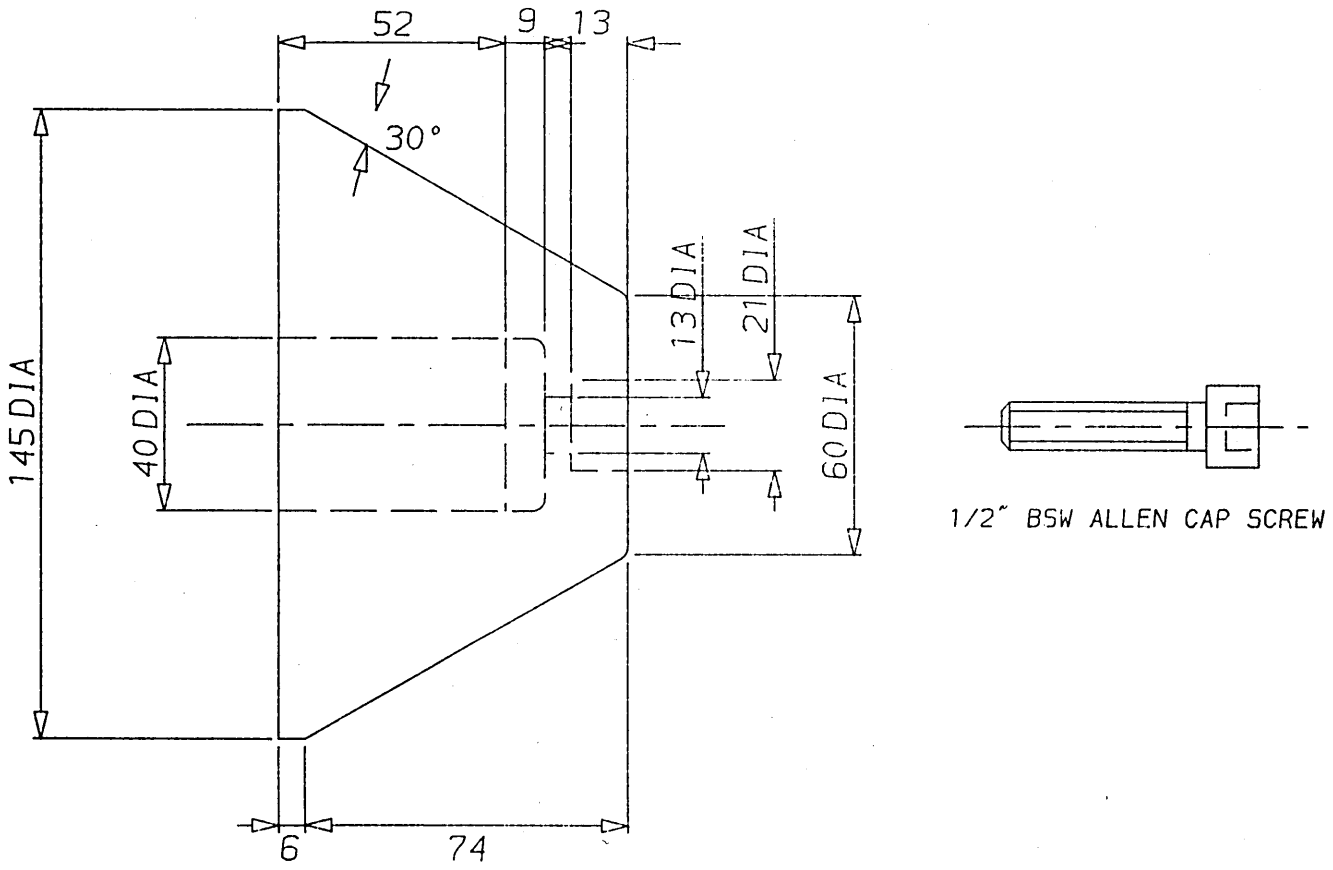


FIG 3.2: Final testing former

$(y-y0)^2 = 4a(x-x0)$ --- Parabola equation

with $(x1, y1) = (30, 0)$

and $(x2, y2) = (50, 74)$

substitute to get the value of (a)

$(74-0)^2 = 4a(50-30)$

therefore $a = 68.45$

the equation becomes with (0,0)

$$y^2 = 273.8\ (x)$$

$$x = \frac{y^2}{273.8}$$

with y values from 0 to 74

calculate x values at 1.016 intervals

Y value	X value
0	0
1.016	0.0037
2.032	0.01508
.	.
.	.
73.151	19.544

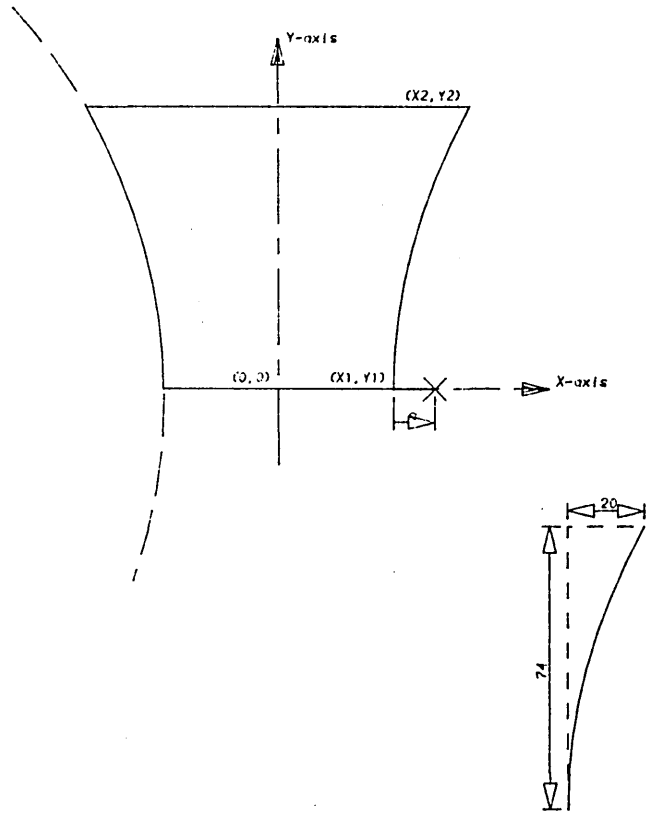


FIG 3.3 : Parabolia equation

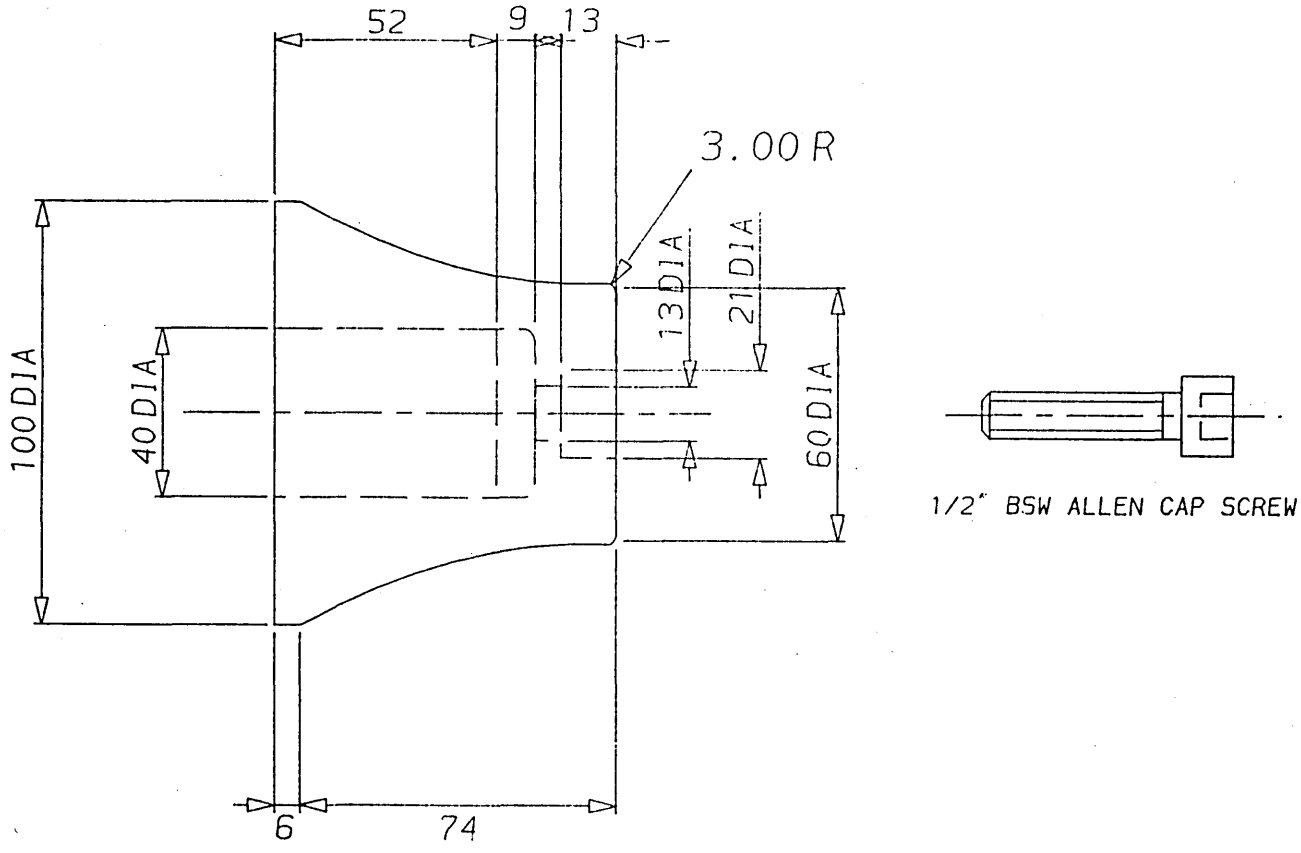


FIG 3.4: Parabola former

ISO work address format of the AUDIT lathe. Finally, punched tape was produced to load into AUDIT.

The former was mounted on a stub arbor which itself was then mounted in a four jaw chuck and trued using a dial test indicator so as to be concentric. The former was mounted on the stub arbor by means of a bolt This was so designed that mounting and dismounting (i.e replacing a former) can be done easily, quickly and efficiently.

All that was required was to slaken the bolt, then remove (dismount) the first former, install (mount) the second former and fix with the bolt again. This arrangement allowed different formers to be used with minimum set-up time.

Some modifications to the lathe had to be carried out in order to accommodate the roller tool. The compound slide was removed and replaced with a fixed tool post on which the roller forming tool was mounted. The fixed tool post was connected to the piston rod instead, so that movement could be controlled hydraulically and the necessary power would be provided to form the workpiece. In order to prevent damage to the cross-slide transducer, the metal back stop was fixed to the top surface of the cross-slide, which prevented the cross-slide from compressing the transducer beyond its working travel, (see fig 3.5 also see chapter 4 section 4.6.2).

Forming rollers must be well supported, free running without play to minimize surface friction, and at the same time capable of producing a high degree of finish. The contour of the roller has to be designed to allow for the correct flow of the material during operation. Accordingly to satisfy these criteria, a roller was made of EN9 steel with a semi circular profile. Rollers are made of a variety of materials. Where heavy pressures are exterted, they are usually of steel, hardened and ground(10).

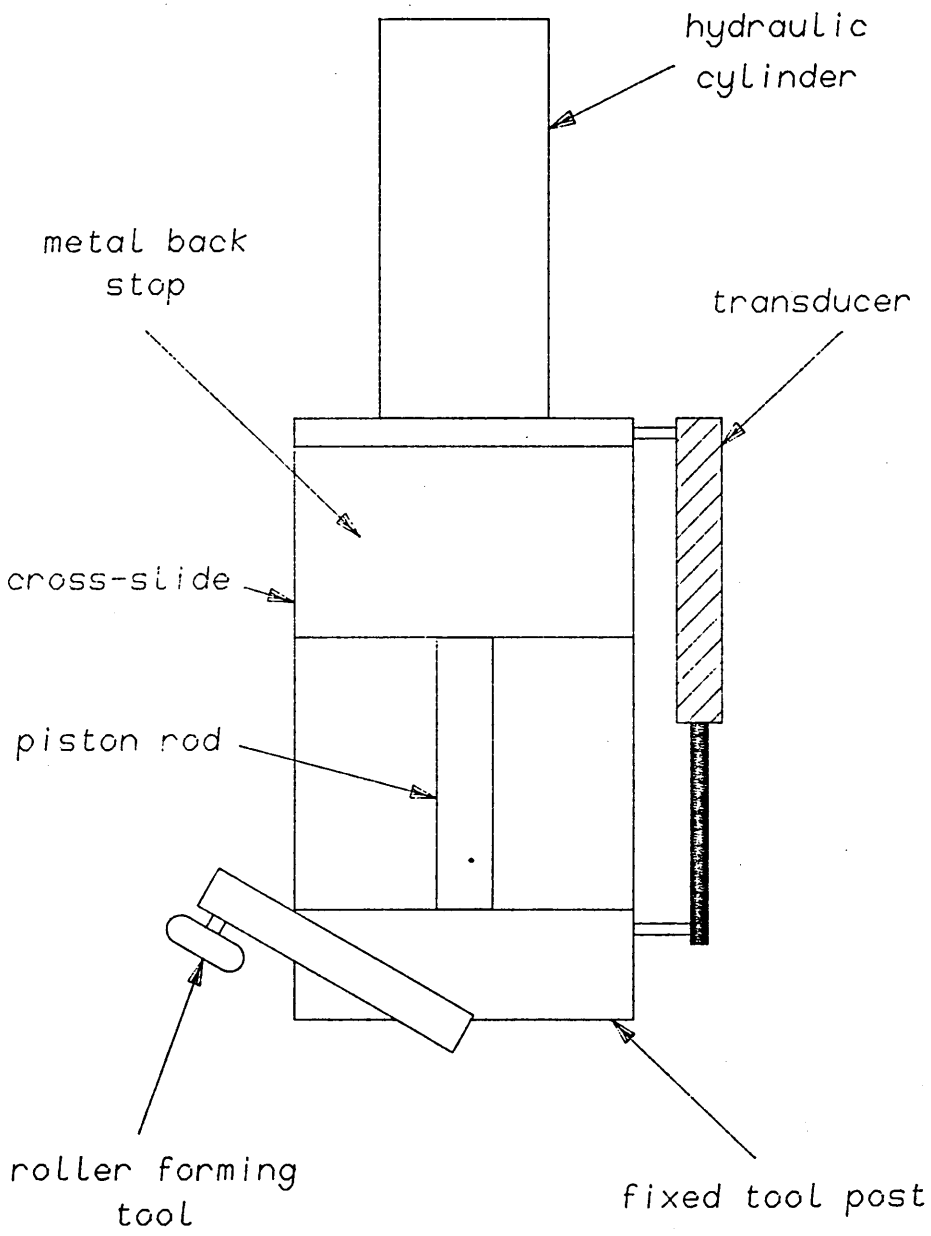


FIG 3.5: A schematic diagram showing the tool fixing arrangement

The whole roller assembly including a support bar, spindle, ball bearing and a roller was designed and fabricated. The roller assembly can be seen in fig 3.6.

3.4 - Preliminary testing

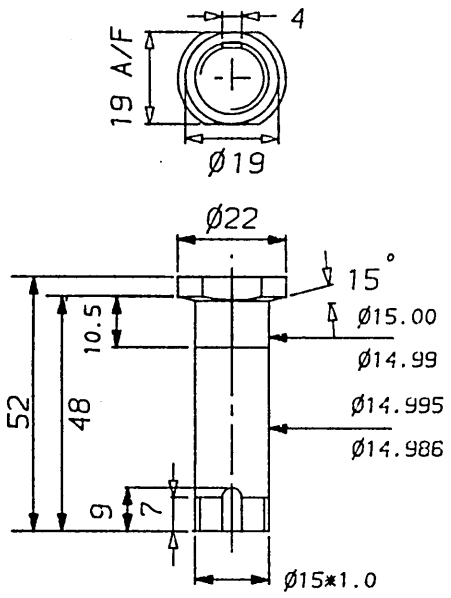
As mentioned in chapter 1, the next step was to devise a method of moving the roller to follow the contour.

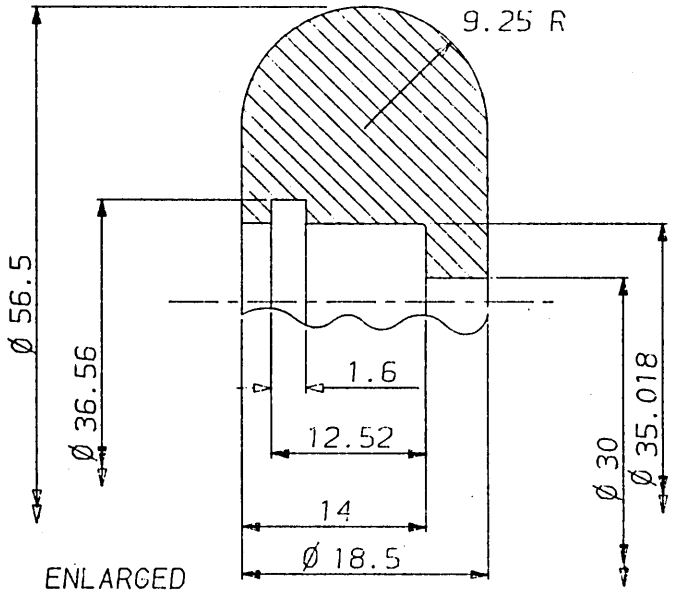
It was decided to use the taper attachment on the lathe to configure the conical contour with a 10 degree taper. The experiments were performed with the feed engauged.

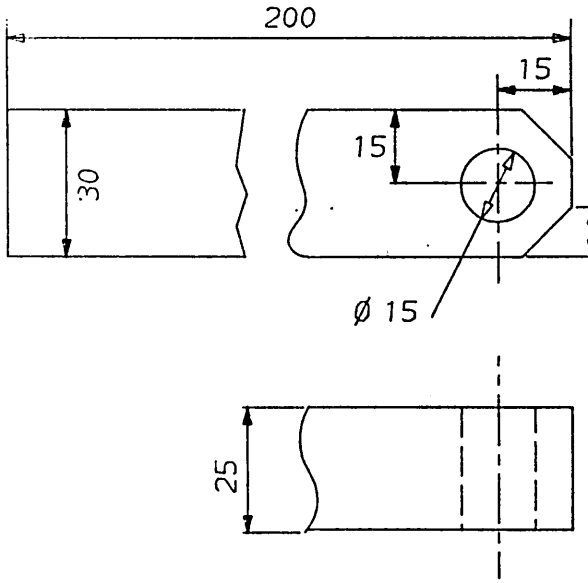
Deformations were recorded on graph paper with peaks observed at the commencement of the bending of the disc. Uneven wall thickness was obtained as it was found that the taper attachment contour did not fit onto the former profile precisely. The outer wall thickness was almost the same as the original blank thickness, whereas the inner parts were reduced by a different extent.

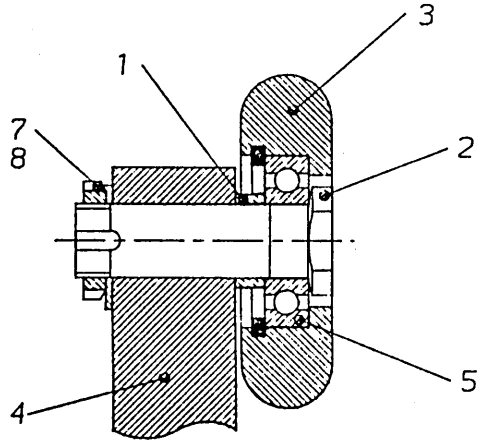
It was informative to measure the forces' magnitudes during initial testing. The axial and the radial force components imposed by the roller were measured by means of a dynamometer installed on the roller pedestal and the values plotted on a graph using a PL 2000 plotter. These can be seen in fig 3.7 and 3.8.

Results showed that the axial force had the maximum values all the time. These results corresponded with the results obtained by C.F.Noble & K.S.Lee(2) and contradicted with R.A.C.Slater & A.Joorabchian(7).









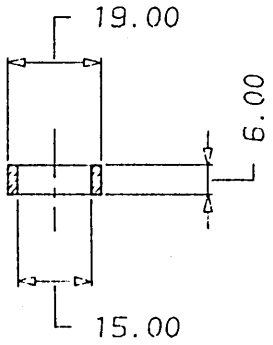


FIG 3.6: Roller assembly

8	2		LOCK WASHER MB2 (FAG)
7	2		NUT. KM2 (FAG)
6	2		INTERNAL CIRCLIP Ø 35*1.5
5	2		BALL BEARING 6202 2Z
4	1	EN1	SUPPORT BAR
3	1	EN9	ROLLER
2	1	EN8	SPINDLE
1	1	EN1	SPACER

ROLLER FORMING TOOL

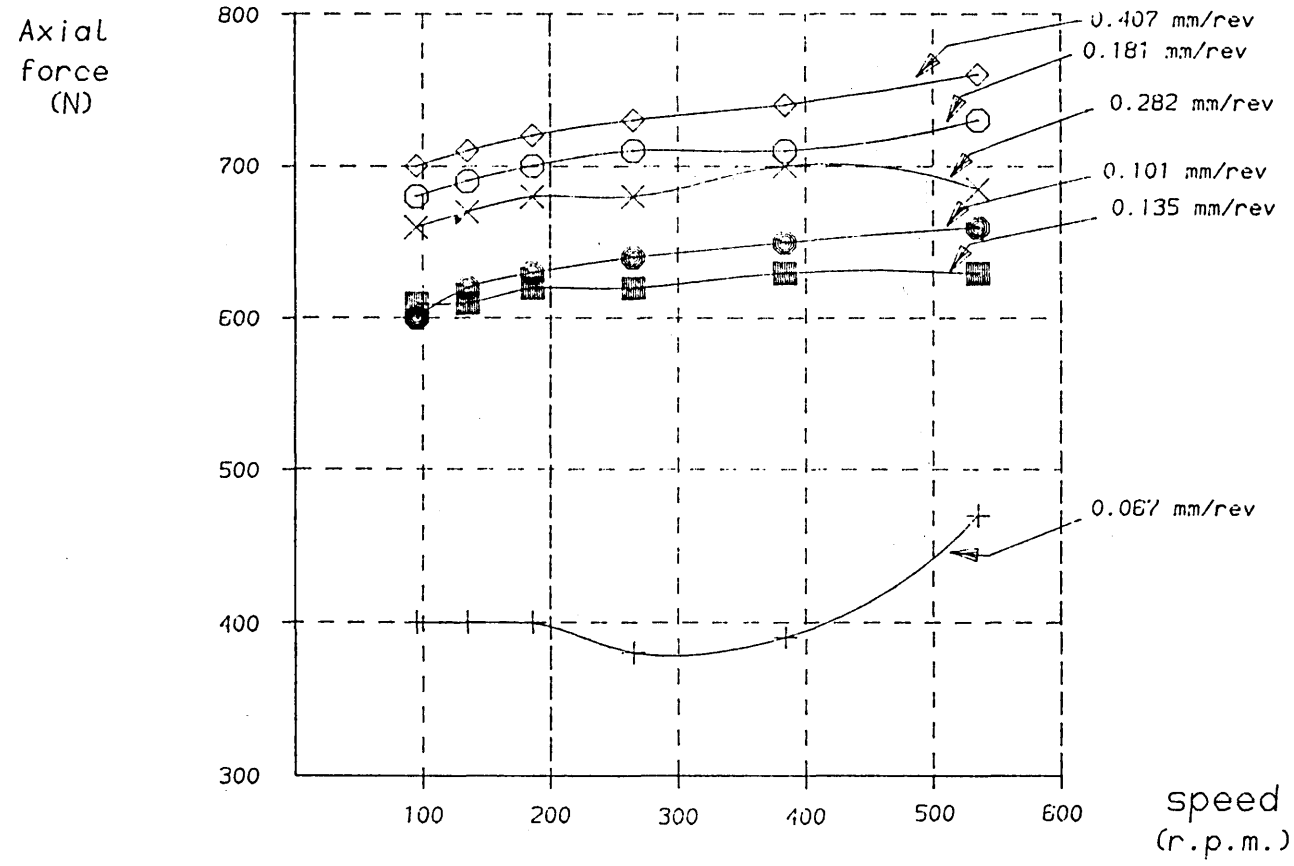


FIG 3.7: Axial force with speed (different feeds)

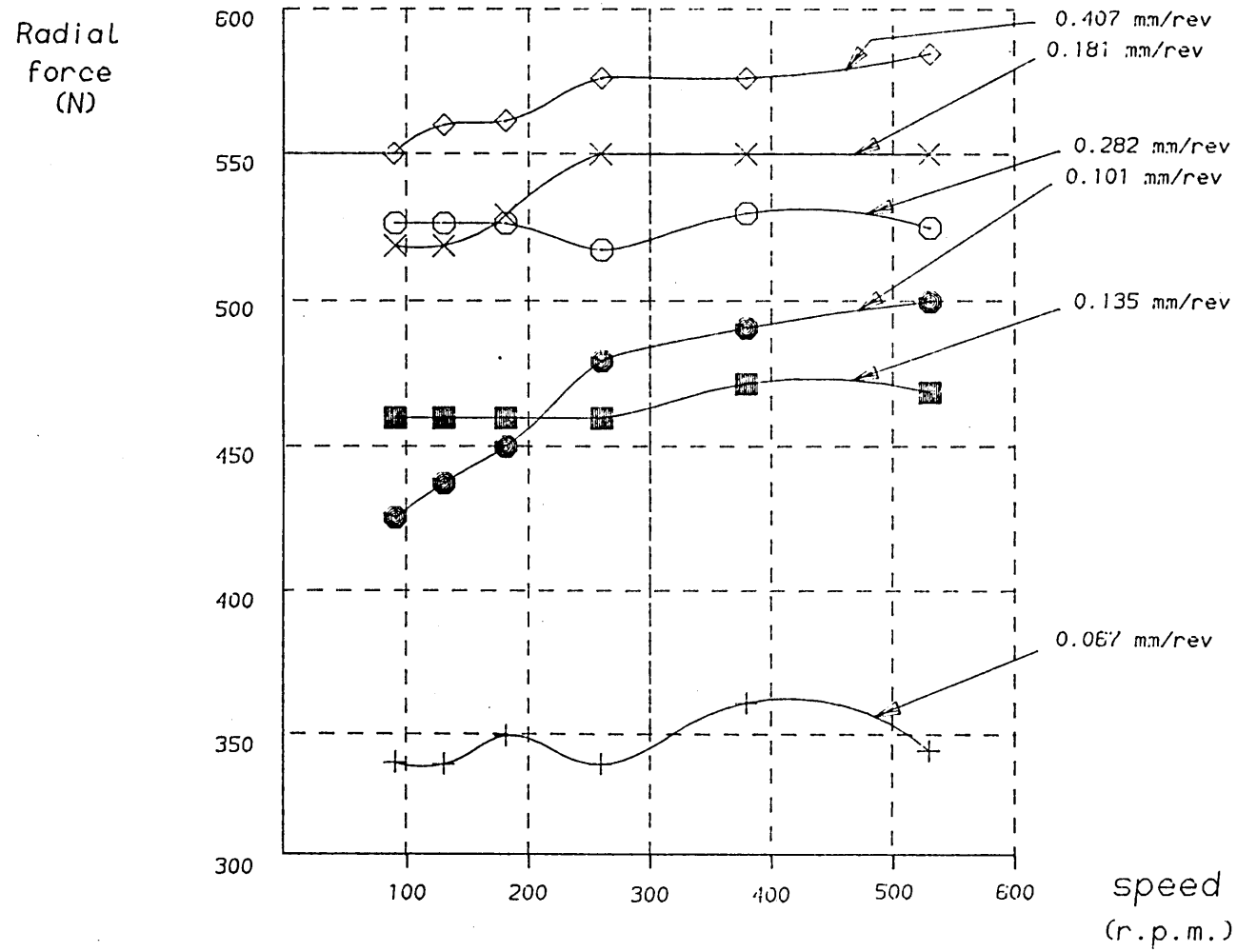


FIG 3.8: Radial force with speed (different feeds)

Chapter 4: Hardware design and testing

4.1 - Introduction

In this chapter, a full description of the hardware components included to build the flow-turning process controller is presented along with the component associated test procedures.

These components can be broadly divided into the hydraulic circuit and the electric circuits which can be subdivided further as follows:

The hydraulic circuit consists of a proportional directional valve, a cylinder, a relief valve, a filter, a gear pump, an electric motor, and an oil reservoir. The design, selection, fabrication and inspection of the hydraulic components are described in detail.

The electrical circuits comprised ADC, DAC and counter boards. Also, their integral parts, namely the linear transducer and the shaft encoder, which complement the ADC and the counter boards functions, are listed. The various components and boards were devised, decided on, assembled and then examined afterwards.

4.2 - Hydraulic components

A hydraulic power pack was built to drive the cylinder controlling the roller. Part of the rig function was to provide the necessary force for the flow-turning to be performed. The circuit was devised and fabricated with the help of the Automation Advisory Service Department. A schematic diagram for the circuit is shown in fig 4.1. The circuit consists of the following parts:-

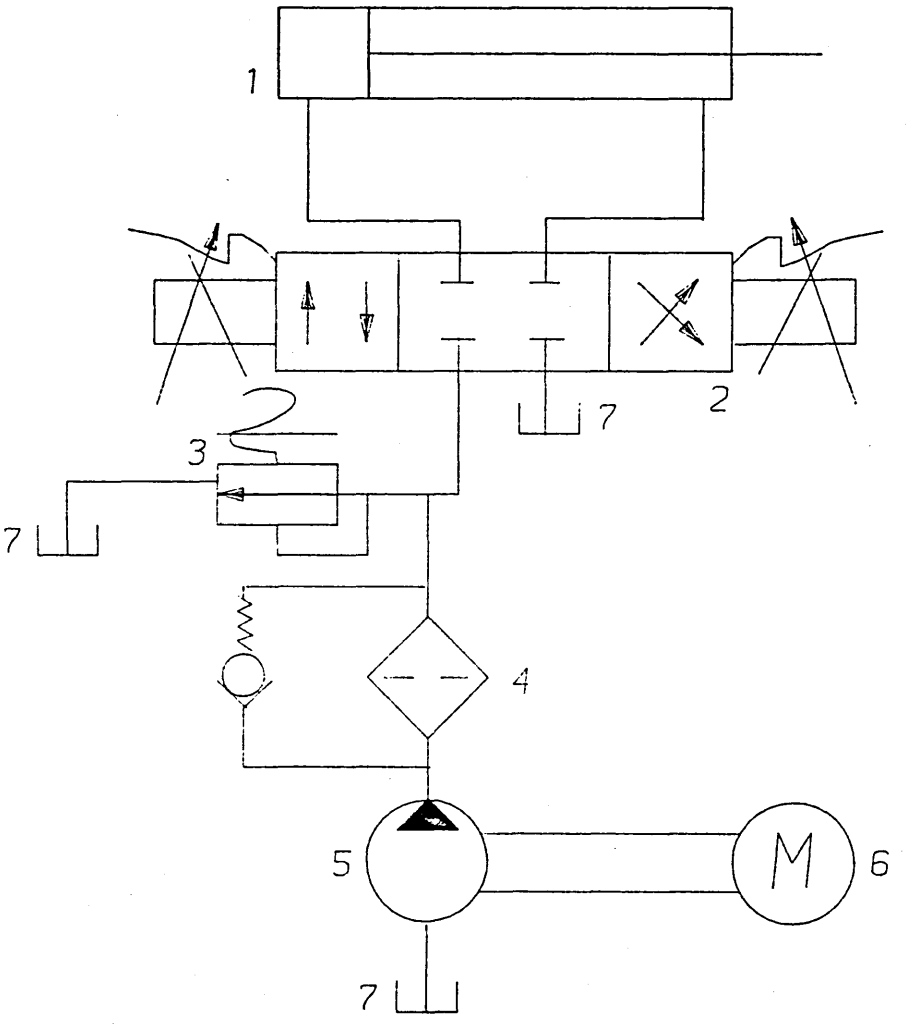
1- A cylinder

2- A proportional directional control valve

3- A relief valve

4- A filter

5- A gear pump



1	HYDRAULIC CYLINDER
2	PROPORTIONAL VALVE
3	RELIEF VALVE
4	FILTER
5	A GEAR PUMP
6	AN ELECTRIC MOTOR
7	RESERVOIR

FIG 4.1: The hydraulic circuit

6- An electric motor

7- An oil reservoir

The cylinder was used indirectly to move the roller. Instead of mounting the roller on the cylinder rod, it was found technically easier to connect the piston rod to the cross-slide over which the roller was fixed.

The oil flow rate control and direction were accomplished by means of a proportional directional control valve. This type of valve provides flow control together with directional control in the same manner as the normal type of directional control valve. Thus a single proportional valve can fulfil the functions of flow control, directional control and braking valves. It offers a simple method of electrical control of the working speed of hydraulic units, such as cylinders and motors. The hydraulic components can be seen in plate 4.1.

4.3 - The hydraulic circuit design

An approximate calculation of the hydraulic circuit components needed were as follows

Roller force on disc (radial) = 4.7 kN

Cylinder speed (maximum) = 70 mm/s

we assume a pressure of = 70 bar

$$F = P * A \quad -- (1)$$

where

P= cylinder pressure (N/m^2)

F= roller force (N)

A= cylinder area (m^2)

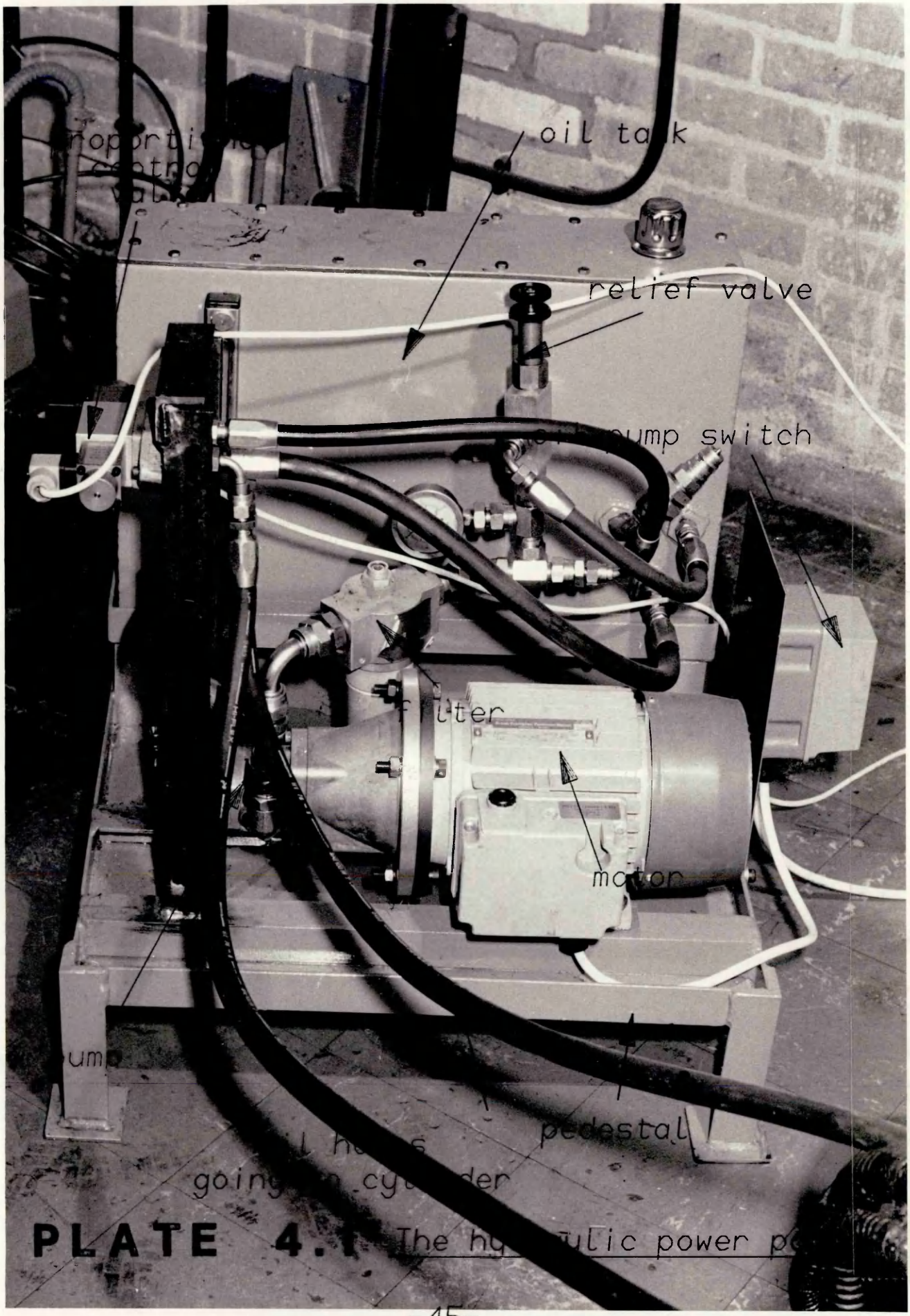


PLATE 4.1 The hydraulic power pack

From equation (1) is derived the cylinder area and then the diameter was determined to be 2.92 cm. The nearest cylinder size is 3.81 cm. The correct pressure is calculated corresponding to the force of 4.7kN from (1) to be 41 bar.

$$Q = V * A \quad \text{------ (2)}$$

where

Q = oil flow rate (m^3/min)

V = cylinder velocity (m/s)

A = cylinder area (m^2)

The oil flowrate from equation (2) was calculated to be 4.78 ℓ/min. The pump capacity will be about 6 ℓ/min. The motor power is to be determined from

$$\text{Theoretical power} = \frac{P * Q}{600} \quad \text{------ (3)}$$

where

P = power in (kW)

Q = flowrate in (litre/min)

The theoretical power was calculated from (3) to be 0.327 kW. A 1.1 kW motor was chosen in order to compensate for the hydraulic losses in the different parts of the circuit. The proportional valve size should be selected to accommodate about 6 ℓ/min. The relief valve should be able to bypass the calculated flowrate when the piston reaches one of the cylinder ends, so the capacity should be near 6 ℓ/min. The valves, filter and pipework must be capable of handling a flow of approximately 6 ℓ/min.

4.3.1 - Proportional directional valve

The oil control valve used was manufactured by Integrated Hydraulics (Wandfluh) with a valve number NG6 (VWS4D61-10-TF). Plate 4.1 shows the proportional valve.

4.3.2 - Hydraulic cylinder

The cylinder chosen was the Carter model BBJ 1.5 in Bore * 6 in Stroke and with a rod diameter 5/8 in, style MF1. Plate 4.2 shows the hydraulic cylinder.

4.3.3 - Relief valve

The relief valve chosen was manufactured by Integrated Hydraulics (Wandfluh) with a valve number 1G11-R2W-10S. Plate 4.1 shows the relief valve.

4.3.4 - Filter

The oil filter used was designed by Pall Industrial Hydraulics with a filter assembly part number HH9020UPRBD. Plate 4.1 shows the filter.

4.3.5 - Gear pump

It was decided to choose GMM gear pump with 4.4 ℓ/min at 1500 rpm delivery, motor speed of 1420 rpm so that the pump output is 4.16 ℓ/min. Plate 4.1 shows the pump.

4.3.6 - Electric motor

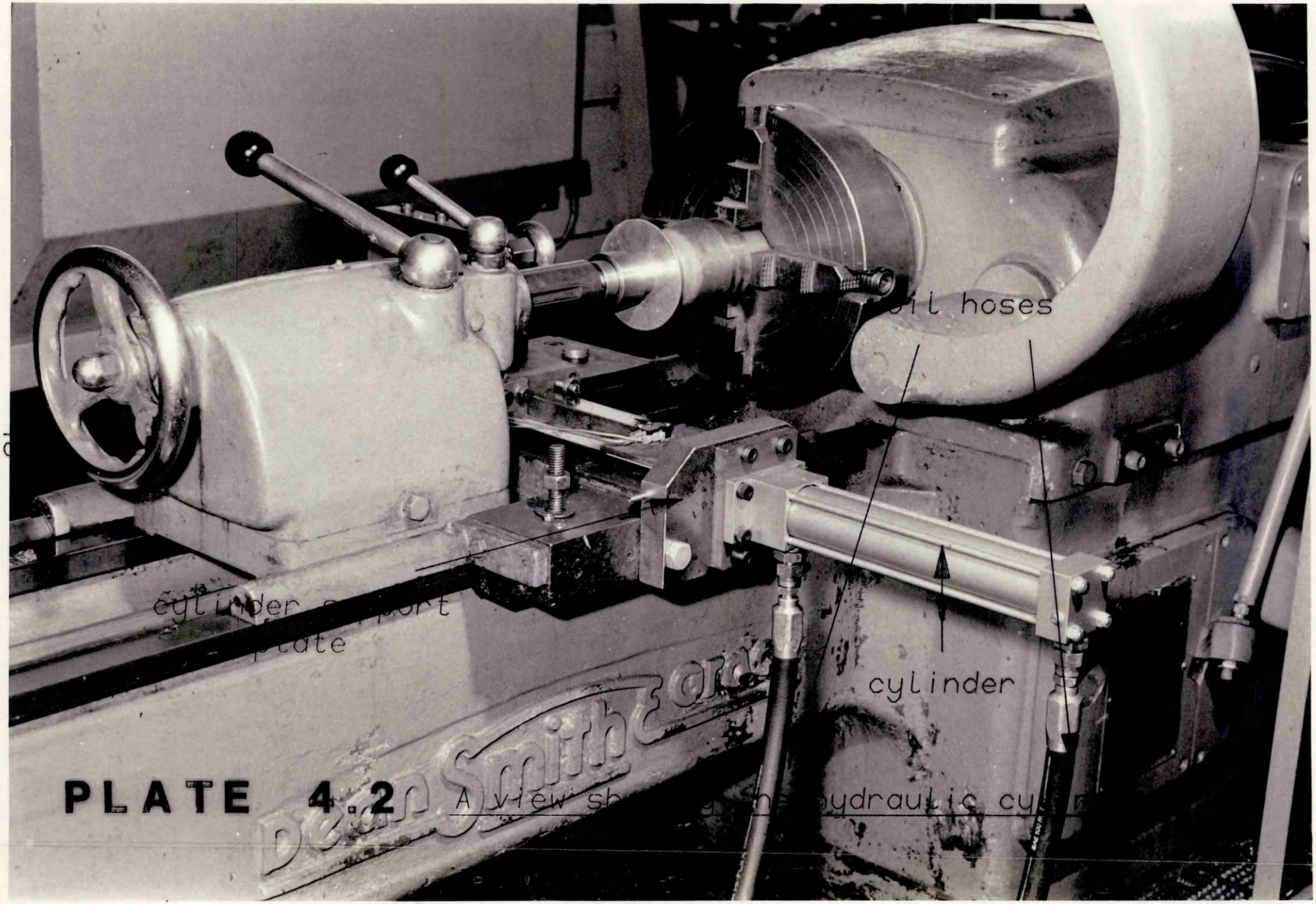


PLATE 4.2 A view sh[illegible] hydraulic cy[illegible]

The gear pump was driven by a Brook Crompton Parkinson Motor (BCPM) with ED90L frame size, 1.1 kW, 1.5 hp and 1420 rpm at full load speed. Plate 4.1 shows the motor.

4.3.7 - Oil reservoir

A five gallon oil storage tank was built for the hydraulic circuit. The oil level could be readily observed through the sight glass. The tank was made of mild steel sheet 16 gauge and also served as a heat sink and dissipator.

It was provided with 2 internal baffles, a suction line strainer, a return line defuser to prevent turbulence and a filler cap air breather (Plate 4.1).

4.4 - The hydraulic circuit test

The principal objectives were to check

1- Motor wiring connections

2- Oil filtration

3- The regulation of the relief valve lift pressure

4- Possible leaks

Before using the system, it was necessary to purge it of all dirt particles and fragments trapped inside piping, joints and parts. This was accomplished by operating the pump and circulating oil through the system, without the proportional control valve or the cylinder in the circuit so as to avoid any contamination of these sensitive components.

Next, the proportional control valve was added downstream of the filter and the output ports of the valve connected. The purging of dirt particles was repeated for both positions of the valve.

Finally the cylinder was connected to the output ports of the valve and the piston advanced and retracted repeatedly to release the air trapped inside the cylinder. This was done for about 15 minutes until the piston movement was smooth.

4.5- Electronics

A variety of electronic components and circuit boards was used during this research. The equipment comprised the

1- Transducer

2- ADC board

3- DAC board

4- Shaft encoder

5- Counter board

6- Modified SDK-85 board

These fulfil different functions in the control of the process, each of which is described below:-

4.5.1 - The transducer

Actual tool movement was measured by means of a hybrid track rectilinear potentiometer connected to the cross-slide upon which the roller was fixed. Whenever the tool was advanced or retracted, the transducer would be displaced by the same amount, thus the actual distance moved was known.

The stroke of the transducer was chosen to be slightly longer than the cylinder stroke, which was 150 mm. The resolution was virtually infinite and linearity was within 0.1%.

Mounting was accomplished with two studs, one of which was clamped on the cross-slide (the moving part) and the other end fastened on the saddle far end (the stationary part).
Fine adjustment of the transducer slider was done by rotation of the slider eyelet fixing.

To protect the transducer, another thicker stud was fixed on the side of the saddle opposite the tailstock to guard it from possible damage inflicted by accidently hitting the tailstock.

Further protection, when the transducer was fully retracted, was provided by a rectangular metal block fixed on the far side of the saddle.

Monitoring of the roller (tool) movement feed control in the y-direction was achieved with this device. The required feed for the flow-turning process is specified in the software, the minimum increment being 1/8 of a millimeter (see section 4.6.2.).
The choice of transducer was a Penny & Giles type HLP 190SA1/150/6K.

4.5.2 - ADC board

The analogue output of the transducer required conversion to a digital signal to enable the position input to be fed to the microcomputer. A 12-bit unipolar ADC board was constructed to feed the actual roller position measured by the transducer to the microcomputer. Each ADC conversion would be initiated by a start command coming from the second bit on port (2AH); 40 microseconds later the data could be read from the ports (21H) and (23H).

The following is an approximate assessment of tool traverse position measurement for purposes of ADC selection. A more accurate assessment of tool traverse resolution is given in section 4.6.2.

$$\text{ADC resolution} = \frac{2^N}{\text{transducer stroke (mm)}}$$

where N = Number of bits

$$= \frac{2^{12}}{150} = \frac{4096}{150}$$

$$= 27.3 \text{ steps/mm}$$

Using the 10 most significant bits the resolution becomes:-

$$= \frac{2^{10}}{150} = \frac{1024}{150}$$

$$= 6.8 \text{ steps/mm}$$

which was sufficently accurate for this application. The choice of ADC was an Analogue Devices type AD574. The layout of the circuit board is shown in fig 4.2.

4.5.3 - DAC board

The digital output of the roller control commands from the microcomputer required conversion to an analogue to enable cylinder movement. An 8-bit bipolar ±5 volt DAC board was constructed to convert the digital control signal values output from the microcomputer to an analogue voltage, which was then fed to the valve controller (voltage-current converter).

The roller control commands, (advance, retract or stop) were sent from the SDK-85 to the valve controller. As the proportional directional valve worked on current rather than voltage, the valve controller converted the control signals into current signals. The digital values were output through port (22H). The chosen DAC was a National Semiconductor type DAC 0800. The circuit board layout is shown in fig 4.3.

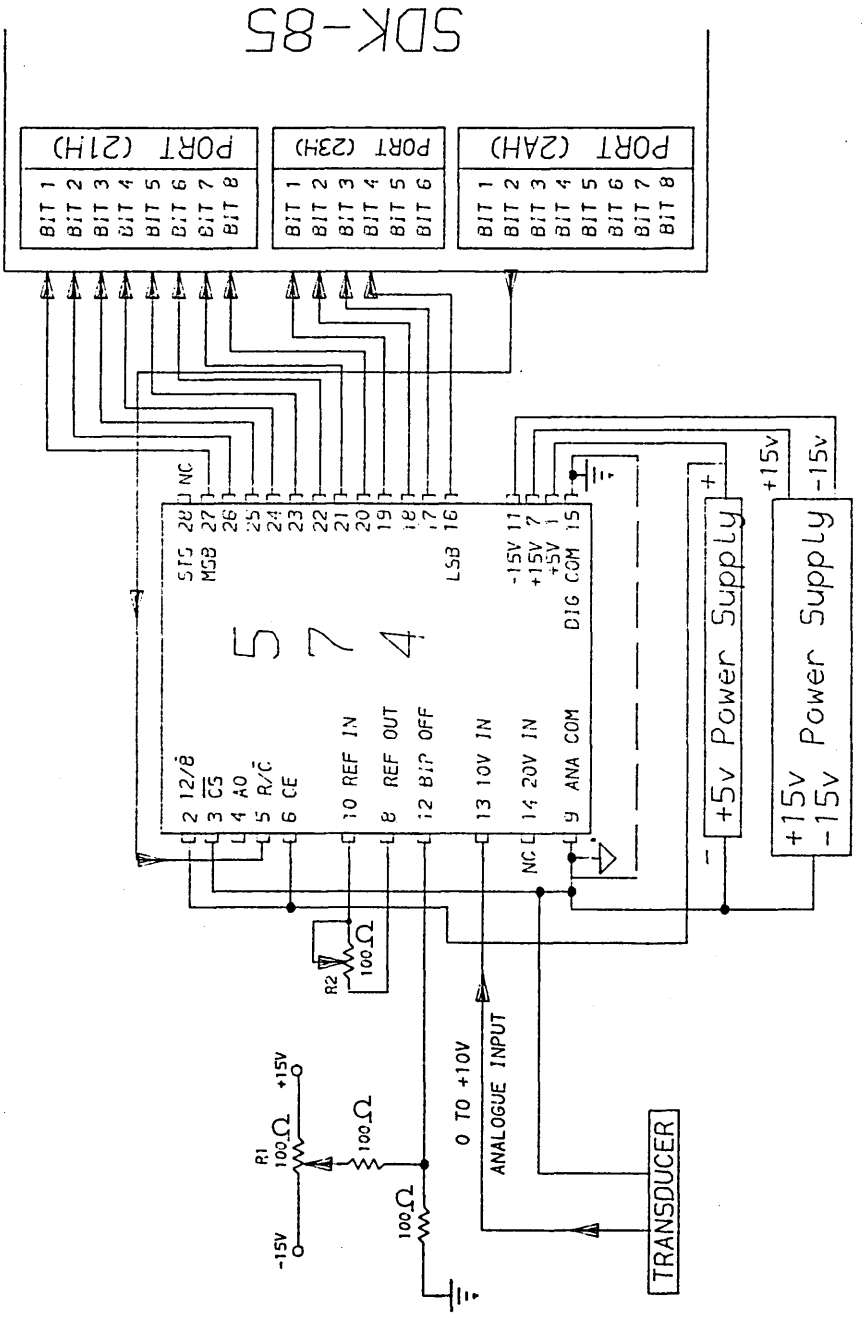


FIG 4.2: Unipolar AD574 ADC circuit

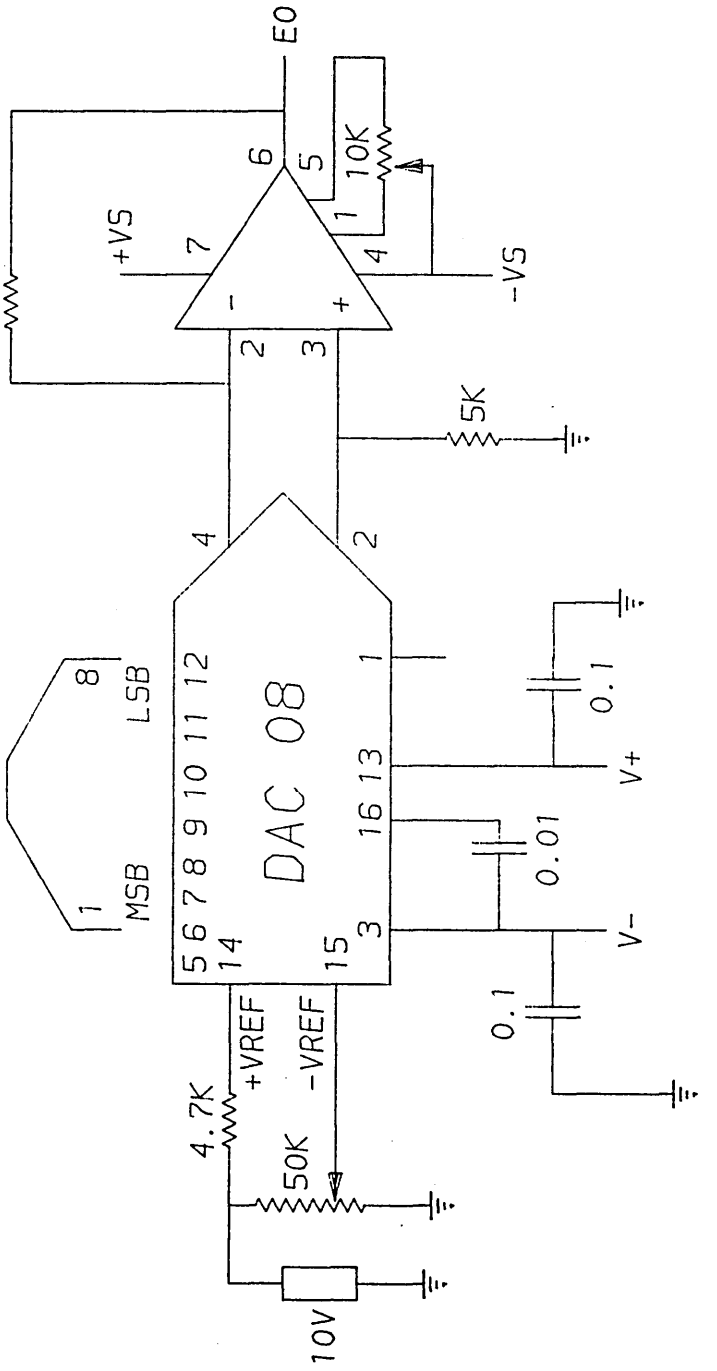


FIG 4.3: Bipolar DAC0800 ±5 volts

4.5.4 - Shaft encoder

The roller traverse position was measured along the axis of the rotating workpiece. A rotary encoder was used to read the carriage position from when it starts moving towards the former until it stops after forming the workpiece, see fig 4.4. A rubber coupling was used to connect the encoder shaft to an adapter head provided on the leadscrew.

The selected encoder was an OMRON E6A-CW100 rotary encoder having 100 pulses per rev with lathe leadscrew pitch of 6.35 mm (1/4 in). Each encoder pulse was equivalent to 0.0635 mm i.e shaft encoder resolution equal to 0.0635 mm.

4.5.5 - Counter board

The output pulses from the shaft encoder were fed to the counter board containing a circuit which provided the binary value of the leadscrew position. This was taken into port 29H on the microcomputer.

The first prototype of this board (shown in fig 4.5) failed to satisfy the operating requirements (see section 4.6.2) and a second circuit was designed and developed, which proved to be satisfactory upon testing. The new board is shown in fig 4.6.

Calculation of maximum counter pulses

In order to determine the total number of encoder pulses stored in the counter variable (in the software), the encoder rotation number has to be calculated first as follows

The total distance envisaged to be moved by the saddle

= Distance before forming (counter disabled) + Form length

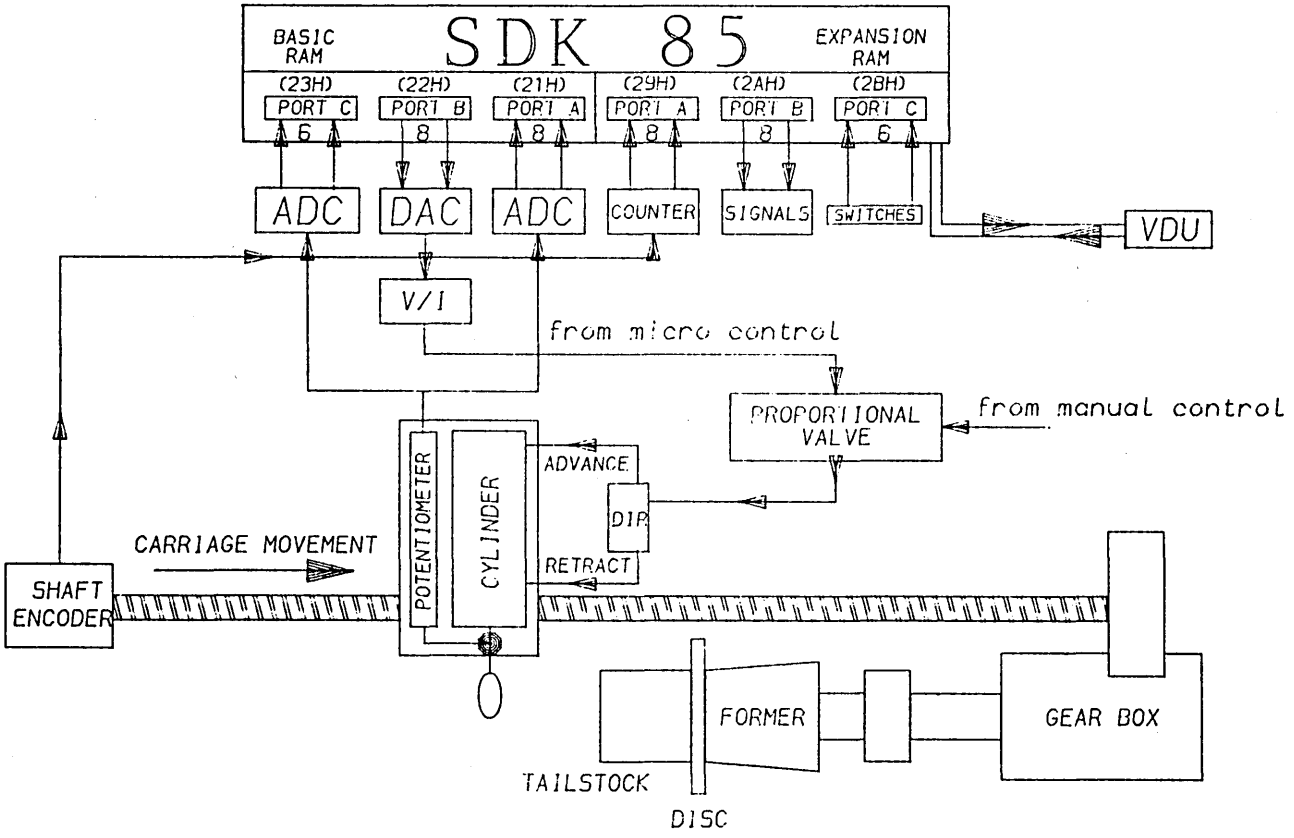


FIG 4.4: An overall schematic diagram

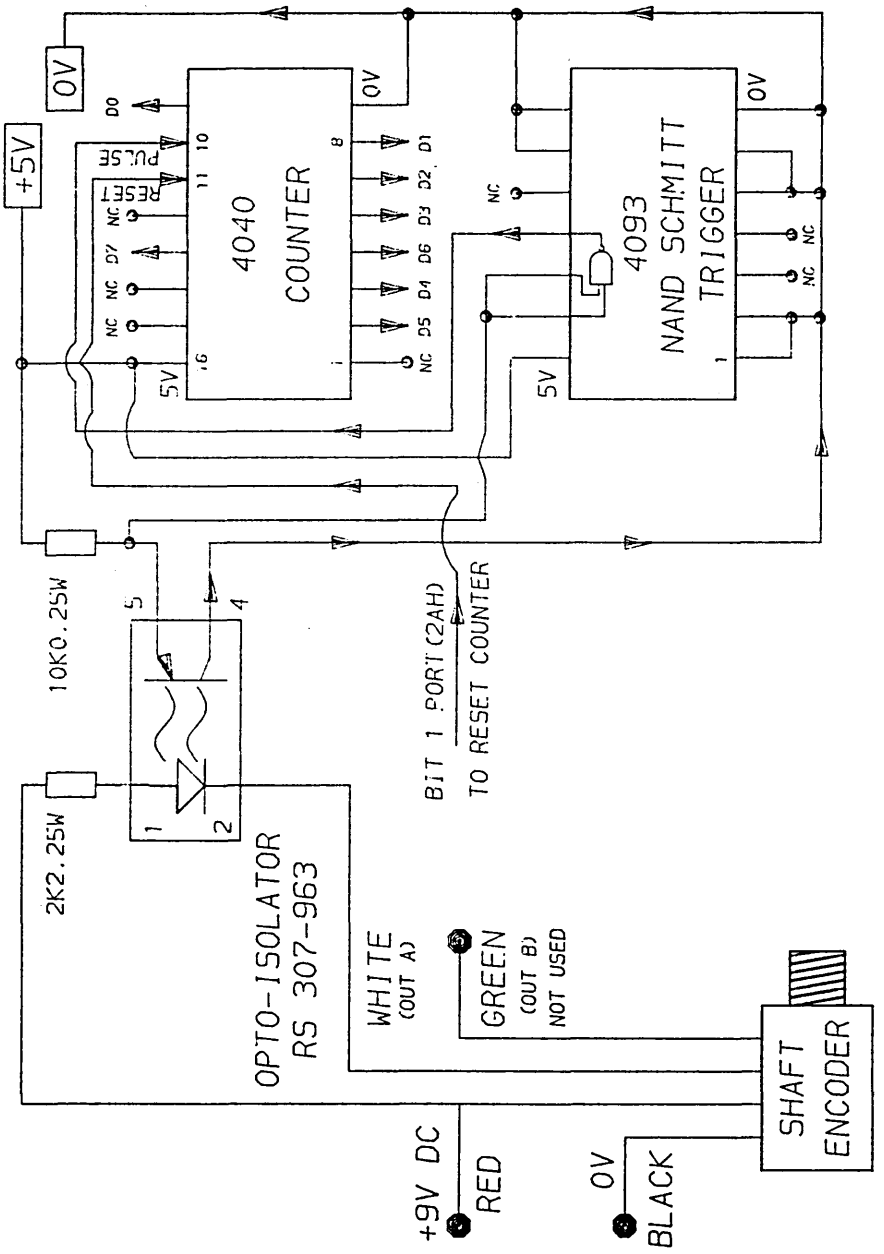


FIG 4.5: Counter circuit

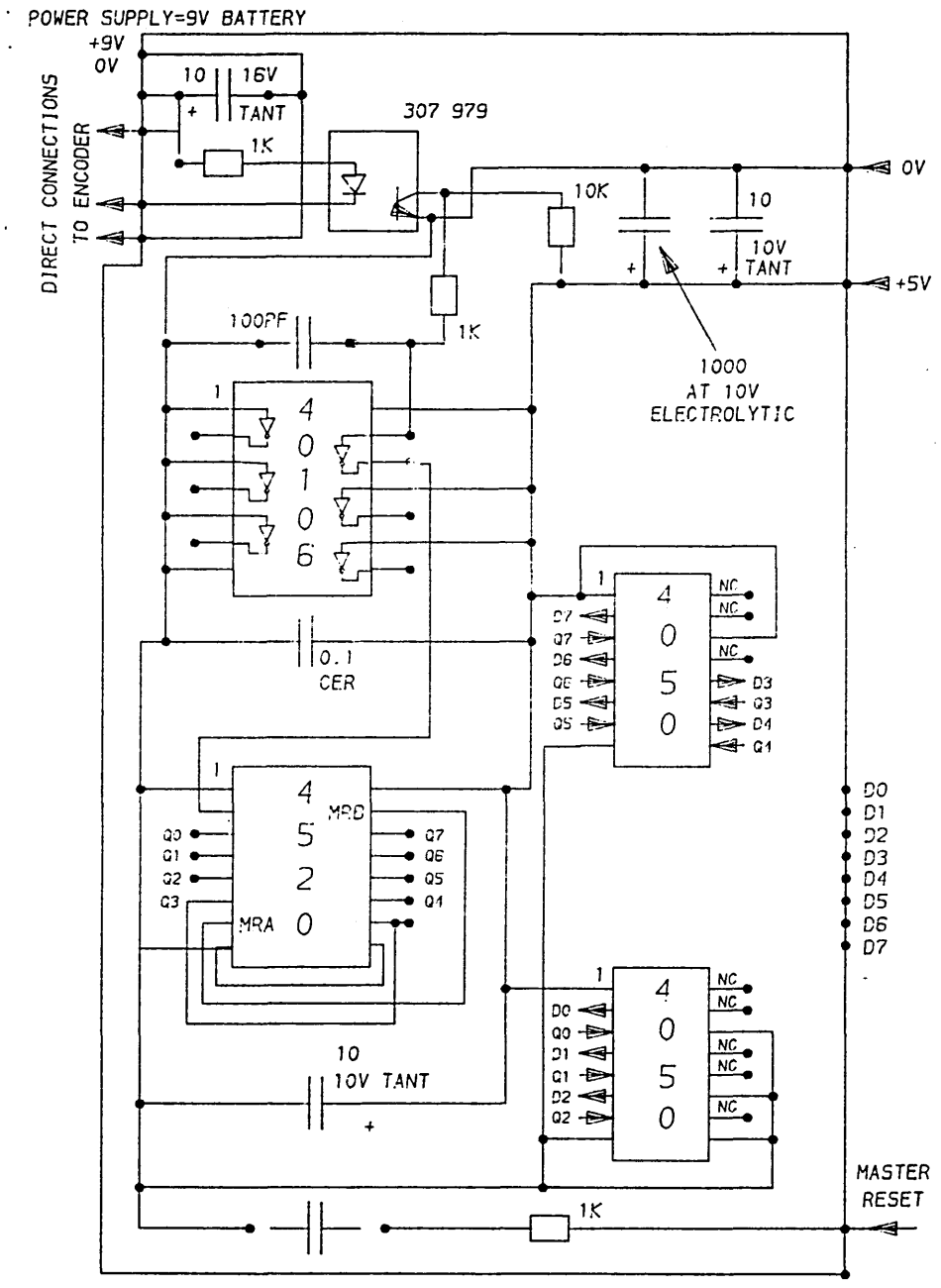


FIG 4.6: Second counter circuit

= 0 + 50

= 50 mm

The leadscrew has 4 threads/in

1 in= 25.4 mm

lathe pitch= 25.4/4

= 6.35 mm

number of rev= The total distance/lathe pitch

= 50/6.35

= 7.87 rev

Therefore the maximum encoder pulse count is of the order of 7.87*100 = 787 pulses. The limitation of the 8 bit counter (255 pulses maximum) was overcome by treating the counter output as an incremental count and accumulating the total count in a sixteen bit variable.

4.5.6 - Modified SDK-85 board

The flow-turning process program had to be stored in the microcomputer memory before the process could actually start, i.e the purpose was to be finally independent of the development system.

After program development, it was found that the program memory requirement was about 10K bytes. This was far greater than that provided on the SDK-85. The memory was therefore expanded by adding three contiguous EPROM chips having a total memory capacity of 12K. It was useful to have some extra memory for further expansion or modifications and for future implementations.

There was another problem in outputting the text to the VDU. It was thought that the SDK-85's original serial interface operating at 110 baud was far too slow for the present application. For this reason a serial interface capable of operating at up to 9600 baud was developed

to replace the SDK-85's serial interface. For this, an ACIA chip MC6850 was used in conjunction with an 8 position baudrate switch and other necessary circuitry (see figs 4.7 - 4.10). The ACIA (asynchronous communications interface adapter) permits data to be transmitted or received in a serial format i.e as a stream of pulses. The output speeds are shown in fig 4.11. Any of these could be selected by simply switching to that baud rate.

4.6 - The electronic circuit test

All of the above mentioned electrical components were to be tested before use. The first task was to modify the SDK-85 board, after which the ADC and DAC and finally the counter board were assembled. The last three were housed in a cabinet for convenience and compactness along with the necessary sockets and switches on the front and back covers. Circuit testing includes:

4.6.1 - Modified SDK-85 board

After the extra ACIA line and the EPROM sockets were added with the necessary circuitry, a careful examination of the additional operating features was carried out.

The sockets were checked by inserting programmed chips into them, and by stepping through known programs (using the SDK-85 keyboard) it was possible to ascertain the functionability of the EPROMs. The ACIA line was readily inspected by outputting a piece of text to the VDU. This was repeated for all the eight different baudrates available.

4.6.2 - Transducer and ADC board

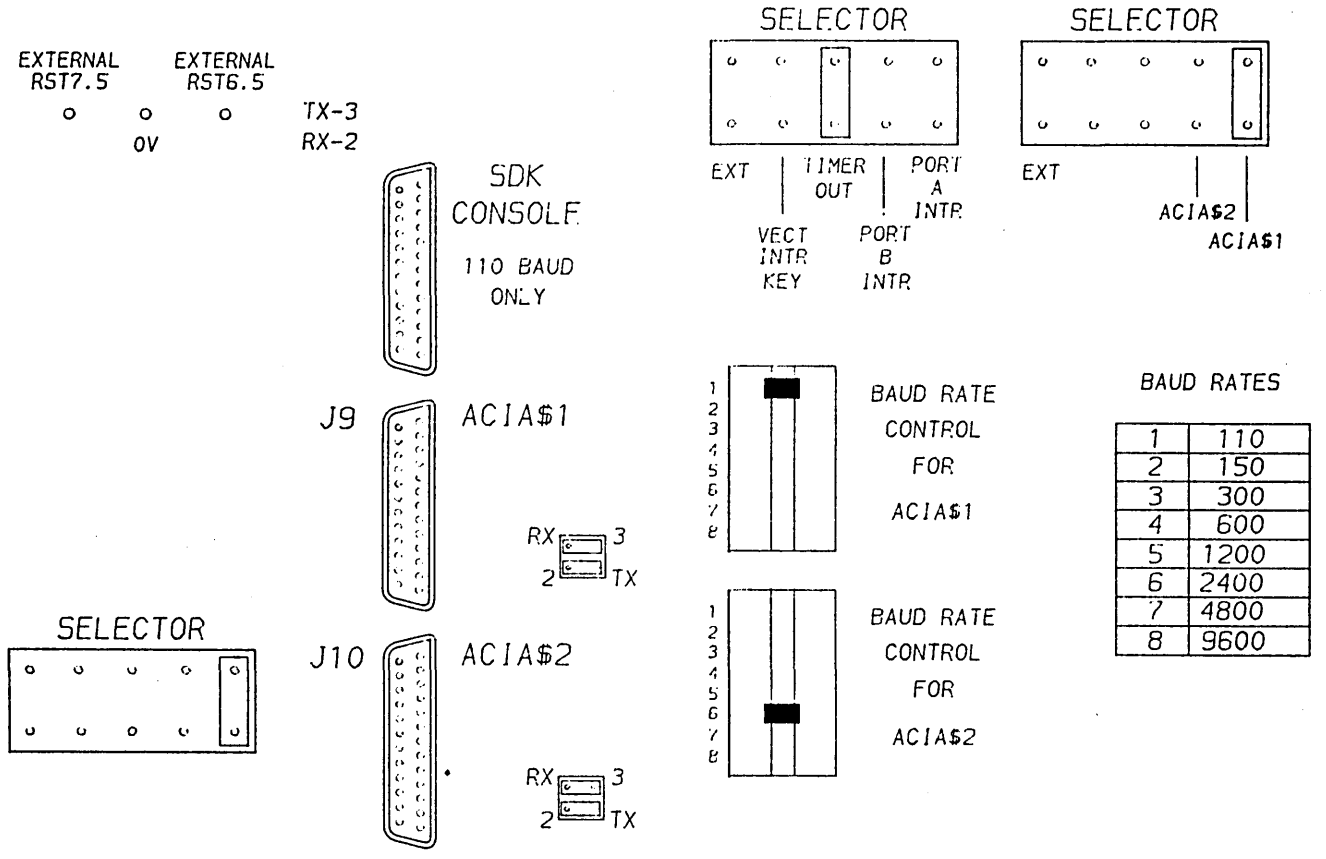


FIG 4.7: Connections for ACIA$1

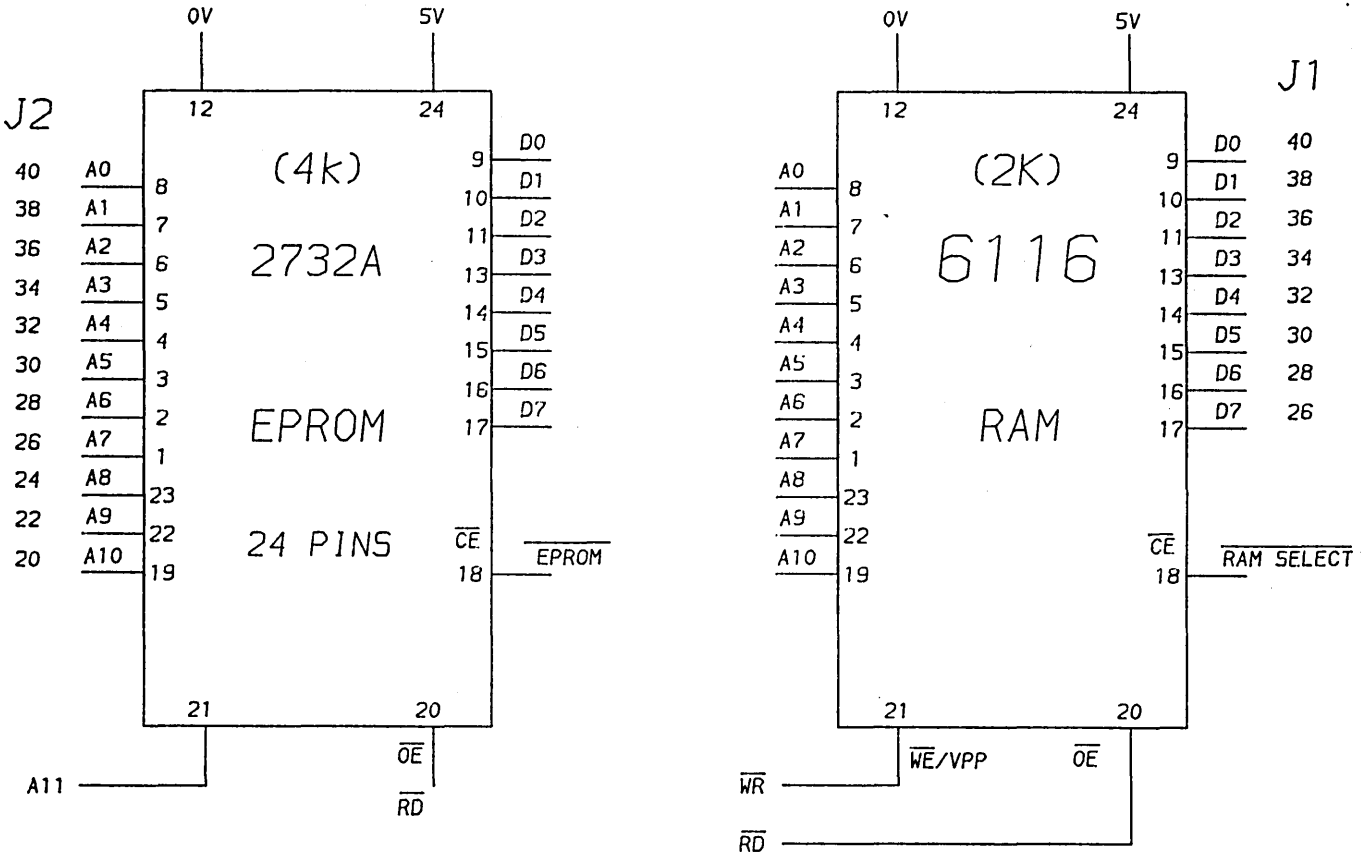


FIG 4.8: Memory devices

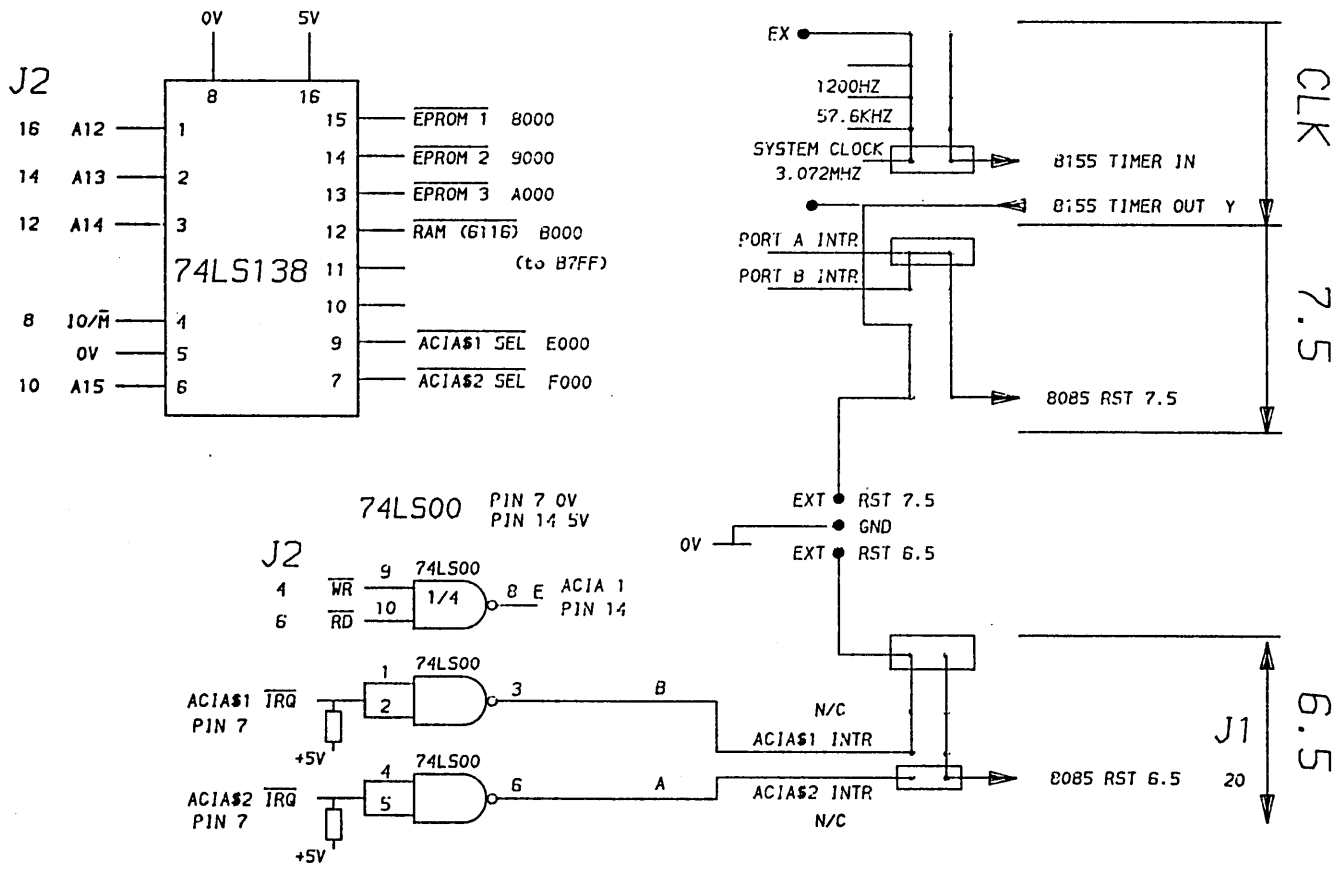


FIG 4.9: Decoding, clock generation, timer in and interrupt selectors

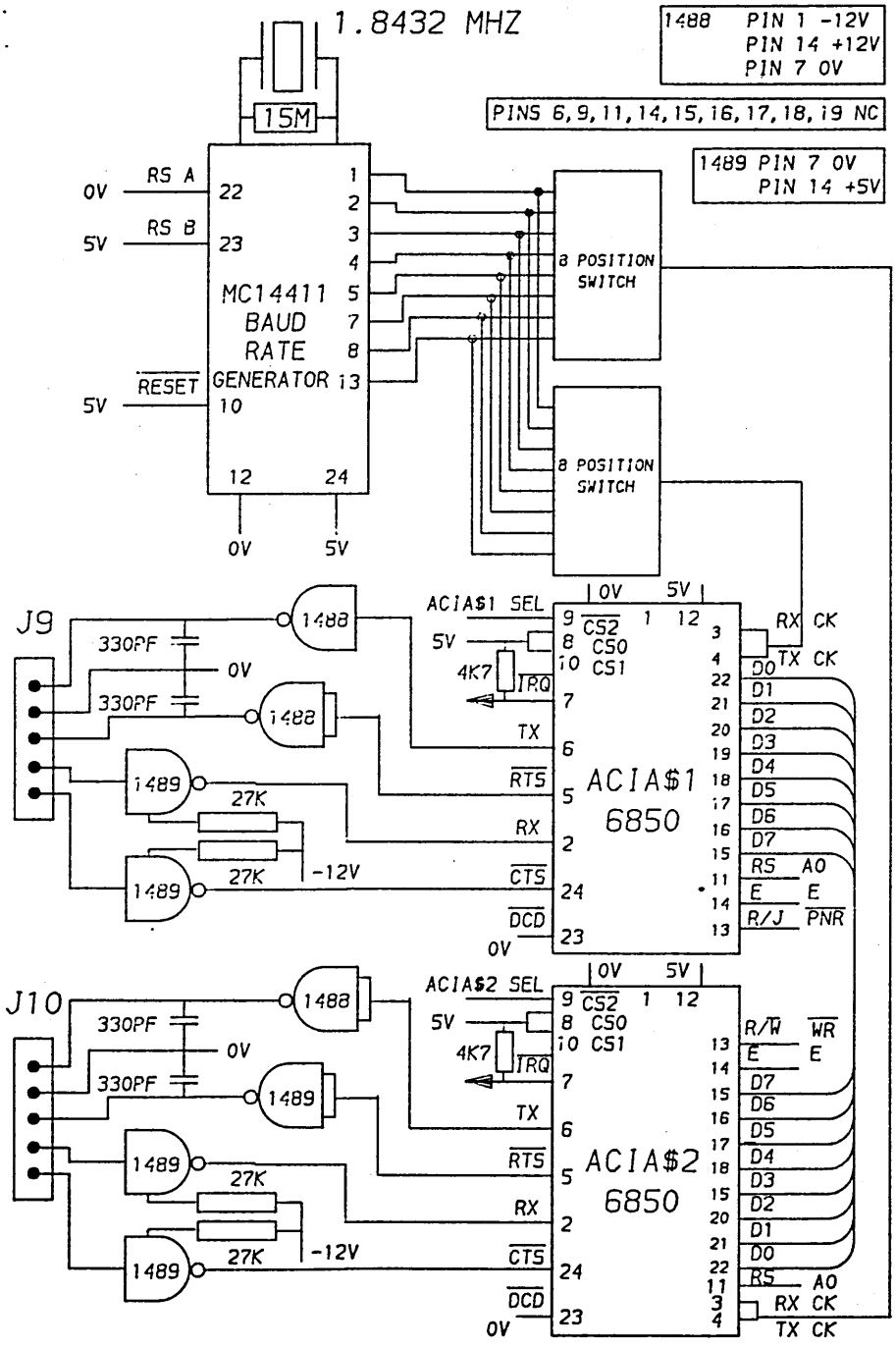


FIG 4.10: Baud rate generator & serial I/O

switch	baud rate
1	110
2	150
3	300
4	600
5	1200
6	4800
7	4800
8	9600

memory map

EPROM 1	8000 - 8FFFH	4K
EPROM 2	9000 - 9FFFH	4K
EPROM 3	A000 - AFFFH	4K
RAM	B000 - B7FFH	2K
ACIA$1	E000 - E001H	

FIG 4.11: Baud rates and memory address map

The ADC board was tested first using a Limrose PB 100 digital circuit patching panel to supply the conversion start command and monitor the digital output of the ADC and a variable power supply to supply the analogue input. Thus a known d.c. voltage between 0-5 volts and measured with a digital multimeter was input to the ADC and the corresponding digital value from the ADC was displayed on the PB 100 LED panel. Input voltages according to the ADC manufacturer's calibration data were applied and adjustment of the gain control made so that the digital value was in accordance with the data. The clock on the PB 100 was used to control the start conversion on the ADC.

The next part was to test the displacement transducer. To ensure the transducer met specifications, a measured d.c. voltage was connected and its output voltage was taken to the ADC input. Both the transducer output and the digital output of the ADC were then monitored when the transducer was moved. It was found that the relationship was not linear due to the ADC causing a loading effect on the transducer output.

This difficulty was overcome by incorporating a d.c. voltage follower board in the circuit.

The transducer was also examined for linearity over its whole range to ensure accurate performance. A further test was carried out with the transducer connected to the cross-slide of another lathe enabling accurate movement of the transducer by the cross-slide traverse control. Readings were recorded at a given interval and two different input voltages were used, results were tabulated and then presented in graphical form, fig 4.12.

It was obvious by observing these figures that the transducer was linear and within specification.

After checking linearity, the resolution was determined. As only 10-bit of the 12-bit ADC were being used and assuming an active

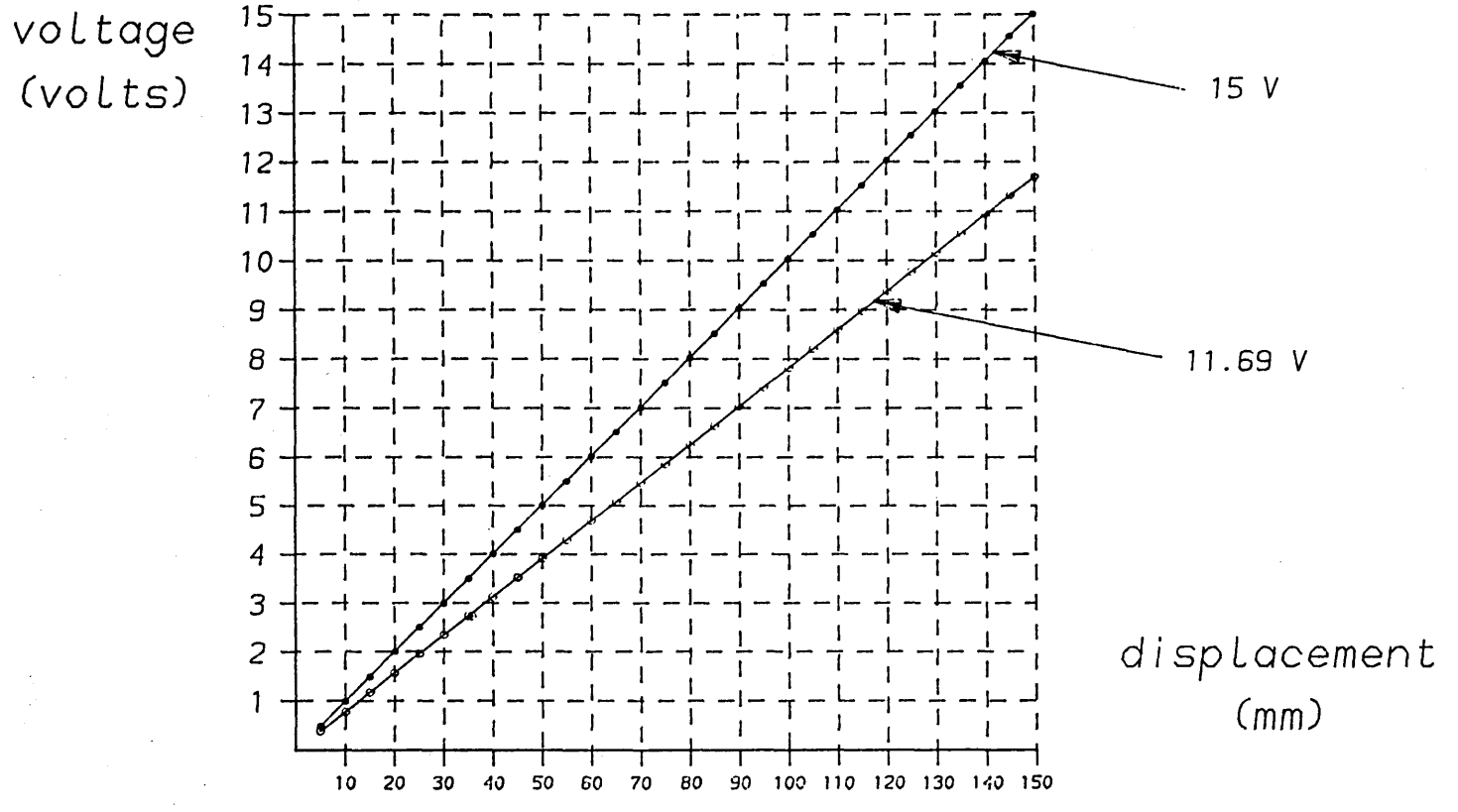


FIG 4.12: Transducer voltage-displacement relation

transducer length of 128 mm from the 150 mm total length, it was calculated that the total voltage needed across the transducer would be 11.719 volts to give 1/8 mm resolution of the tool traverse.

4.6.3 - DAC board

The apparatus used for this test was similar to that for the ADC, which included the limrose PB 100 test board and the digital multimeter.

The digital values were input from eight switches on the Limrose panel and the corresponding analogue voltage values were shown on the DVM display. As the switch combinations were altered, the DAC output changed symmetrically between ± 5 volts.

To see the DAC output values on the screen, a small program was written. The software was run with the Intellec development system, an ICE-85, an oscilloscope and the SDK-85 board. The symmetrical values in the form of a continuous series of inclined straight lines (i.e a sawtooth waveform) across the oscilloscope screen were seen. The testing program PROG 3 is shown in appendix 1.

The cylinder speeds during advancing and retracting had to be calculated in order to be incorporated in the software. This was done by measuring the time required to move the roller tool a constant distance of 148 mm, which is the cross-slide traverse, and then dividing this distance by the measured time to obtain the speed.

A variable power supply was connected to the DAC board during the test while the voltage was displayed on a DVM. The value corresponding to a given speed was recorded. The test was repeated three times and the average was taken for the recorded times.

The equivalent decimal numbers corresponding to the recorded voltages were calculated for use later in the program. Values were

obtained for advancing and retracting, and are illustrated in graphical form in figs 4.13, 4.14 and 4.15.

4.6.4 - Shaft encoder and counter board

Prior to using the encoder on the rig, it was essential to ensure that an accurate number of pulses would be sent to the SDK-85.

It was important to measure the exact number of pulses from the encoder to the counter board, the transfer of the eight bit count through port 29H on the SDK-85 and to obtain the correct computation of the sixteen bit accumulated count. For this purpose a small test program was run on the Intellec development system and the output was displayed on the SDK-85 4 address LEDs as before.

The number of rotations has to correspond with the value listed for the encoder specifications, thus for these tests an encoder of 100 pulses/rev was used. Upon running the program, the number initially displayed was 00H. As the encoder was slowly rotated, the display started counting up accordingly. When it was rotated about half a revolution clockwise, the figure was 032H i.e 50 decimal as expected from the encoder specifications. With continuous rotation, the number was steadily increased until it occupied the four LEDs. The testing program PROG 1 can be seen in appendix 1.

The lathe ON/OFF remote control drive from the microcomputer was accomplished with a Darlington relay connected to the lathe relay. The limit microswitches used for the carriage end and the cylinder were connected to port 2BH, as shown in fig 4.16.

Useful Negative Range

Voltage (volts)	Decimal Equivalent
-2.6	75
-2.8	71
-3	67
-3.2	63
-3.4	59

Calculation Formula

$$d = \frac{(v+6.37)*255}{12.74}$$

Simple calculation

v = 2.4 volts

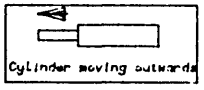
$$d = \frac{(2.4+6.37)*255}{12.74}$$

= 175

Useful Positive Range

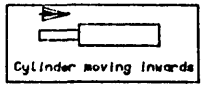
Voltage (volts)	Decimal Equivalent
2.4	175
2.6	179
2.8	183
3	187
3.2	191
3.4	195
3.6	199
3.8	203

Voltage (volts)	Time (s)			Time (s)	Distance (mm)	Speed (mm/s)
	Reading 1	Reading 2	Reading 3			
-2	0	0	0	0	0	0
-2.2	0	0	0	0	0	0
-2.4	0	0	0	0	0	0
-2.6	5.69	5.45	5.47	5.47	148	27
-2.8	3.04	3.34	3.16	3.16		46.83
-3	2.43	2.43	2.42	2.43		60.9
-3.2	2.2	2.21	2.2	2.2		67.27
-3.4	2.2	2.21	2.21	2.21		66.96
-3.6	2.21	2.21	2.21	2.21		66.96
-3.8	2.21	2.21	2.21	2.21		66.96
-4	2.21	2.21	2.21	2.21		66.96
-5	2.21	2.21	2.21	2.21		66.96
-6	2.21	2.21	2.21	2.21		66.96
-7	2.21	2.21	2.21	2.21		66.96



* Velocities less than 27 mm/s are unobtainable

Voltage (volts)	Time (s)			Time (s)	Distance (mm)	Speed (mm/s)
	Reading 1	Reading 2	Reading 3			
2	0	0	0	0	0	0
2.2	0	0	0	0	0	0
2.4	11.65	11.82	11.75	11.75	148	12.6
2.6	5.24	5.23	5.24	5.24		28.24
2.8	3.45	3.12	3.27	3.27		45.25
3	2.69	2.53	2.57	2.57		57.58
3.2	2.06	2.03	2.05	2.05		72.20
3.4	1.87	1.94	1.93	1.93		76.68
3.6	1.85	1.85	1.85	1.85		80
3.8	1.87	1.86	1.86	1.86		79.56
4	1.87	1.86	1.87	1.87		79.14
5	1.87	1.87	1.86	1.87		79.14
6	1.87	1.87	1.87	1.87		79.14
7	1.87	1.87	1.87	1.87		79.14



* Velocities less than 12.6 mm/s are unobtainable

FIG 4.13: Cylinder speed calculation and conversion

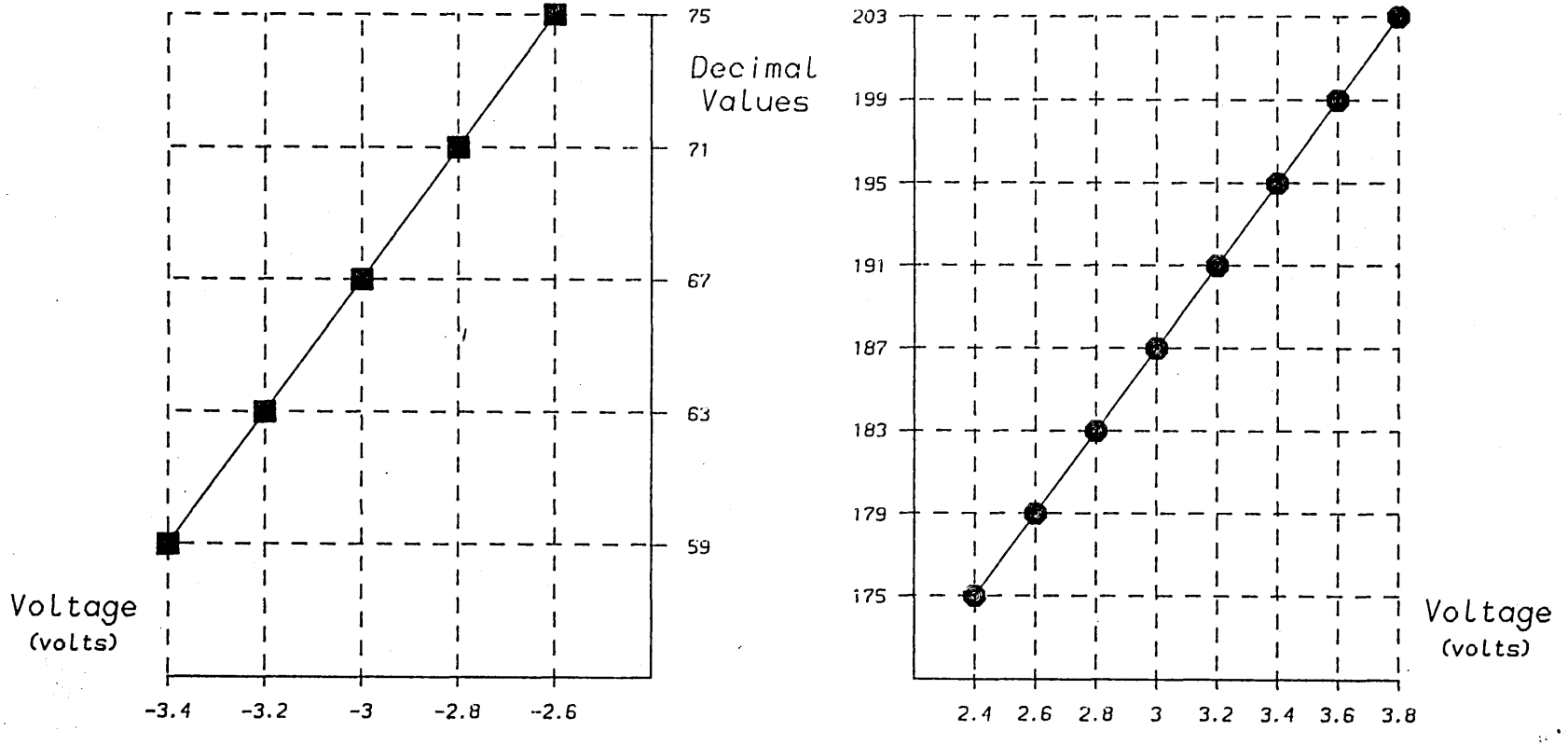


FIG 4.14: A graphical representation of the useful DAC values

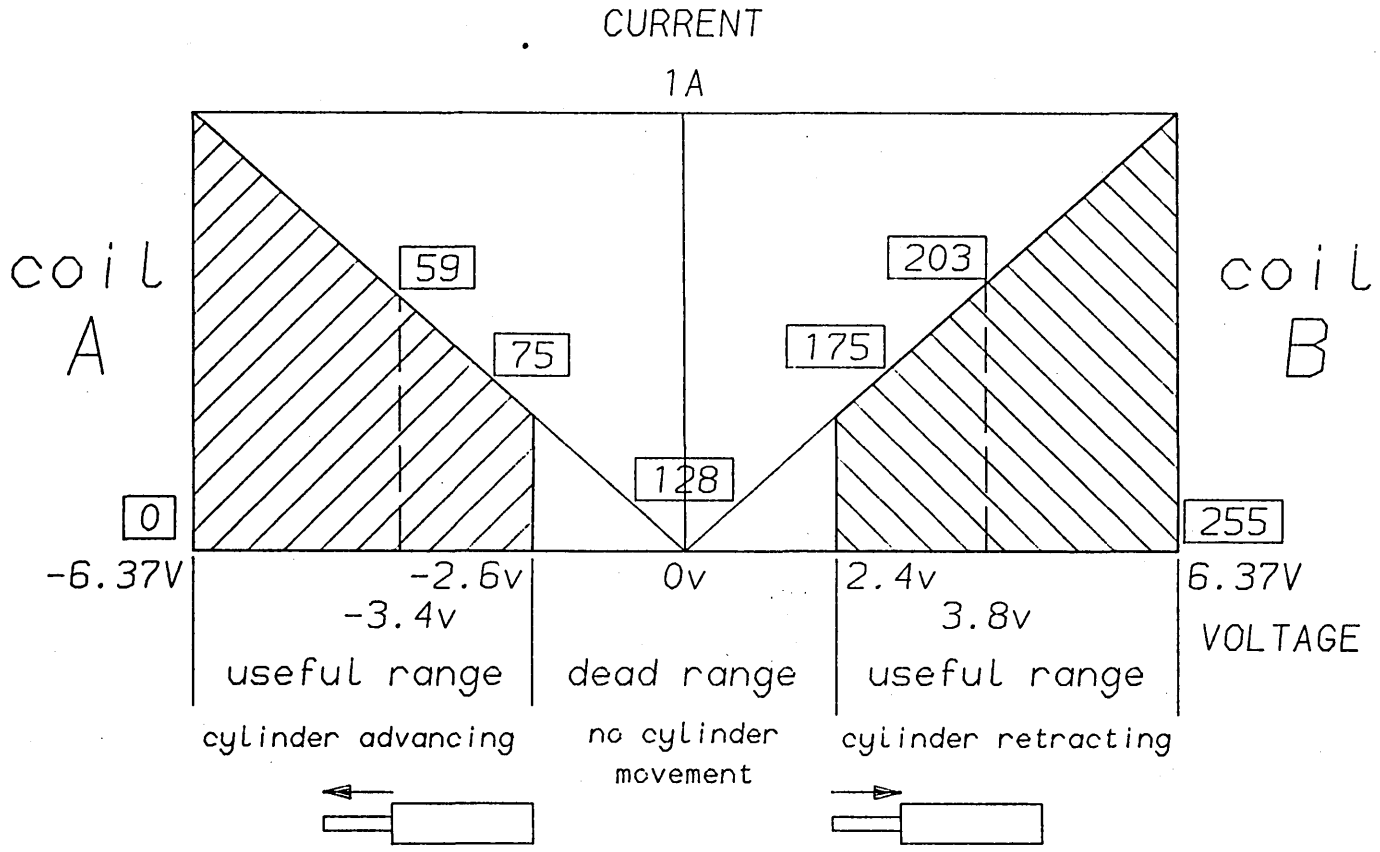
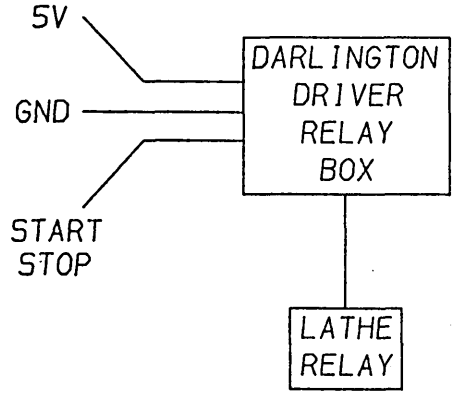


FIG 4.15: A schematic diagram of the DAC range



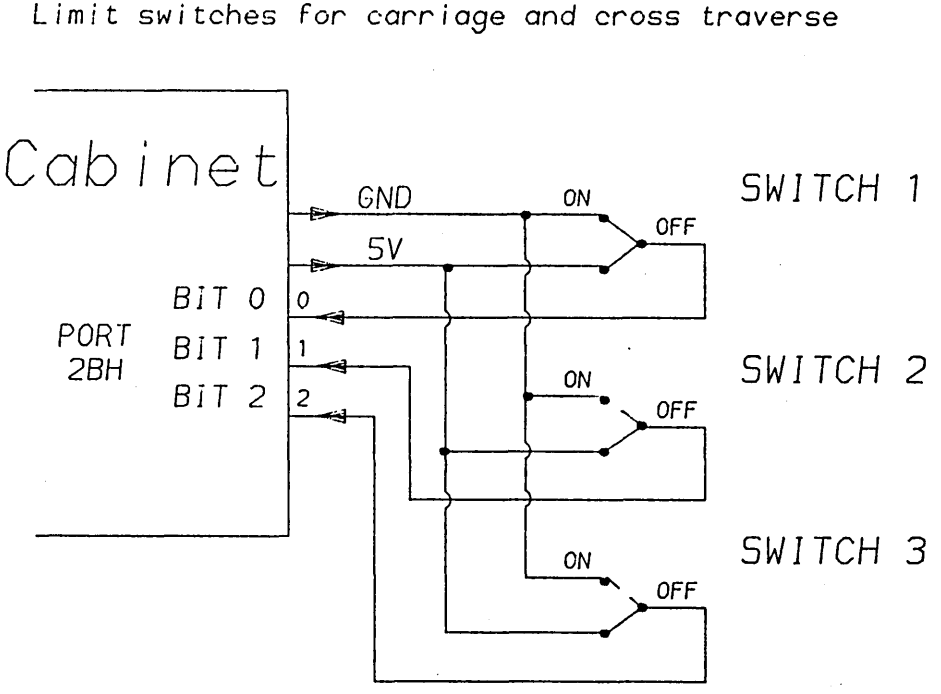


FIG 4.16: Lathe ON/OFF switch drive arrangement

Chapter 5: Software development and testing

5.1 - Introduction

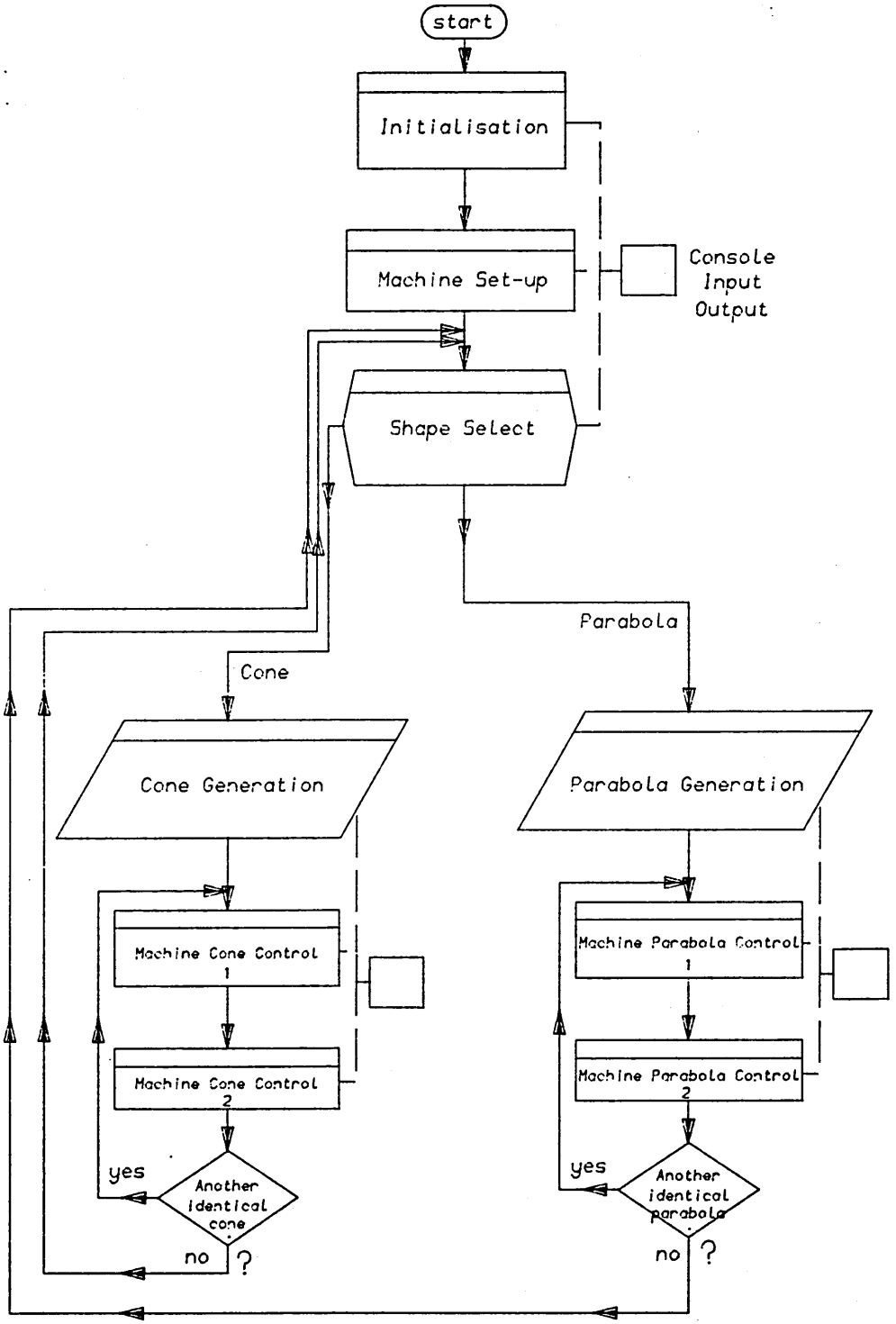
The flow-turning program structure is illustrated with a hierarchy chart. The function of all the modules is described while important Machine Cone Control module is explained with flowcharts.

5.2 - Outline description of software from operator's point of view

This section is concerned with the menu part of the software before, during and after the process as it appears on the VDU screen to the operator (see figs 5.1 to 5.3).

When the program starts, the 'Initialisation' module generates textual information about the flow-turning program on the VDU. The 'Initialisation' module is followed by the 'Machine Set-up' module which provides interactive messages to the operator to enable the rig to be set up for the flow-turning process. Program control then progresses to the 'Shape Select' module, whereby a contour is selected. Either the 'Cone Generation' or the 'Parabola Generation' will follow where the parameters have to be specified. After this, control of the flow-turning process can start in the 'Machine Cone Control' or 'Machine Parabola Control', where the workpiece will take the contour shape. After the workpiece has been formed, the program will ask whether another identical piece is required. If the answer is yes, then control remains in 'Machine Cone Control' or 'Machine Parabola Control'. Otherwise control passes to 'Shape Select' for specifying a new shape. The program listing (in PL/M 80 and Assembly languages) is shown in appendix 2.

5.3 - Software development cycle



Schematic layout of 5.1, 5.2 and 5.3

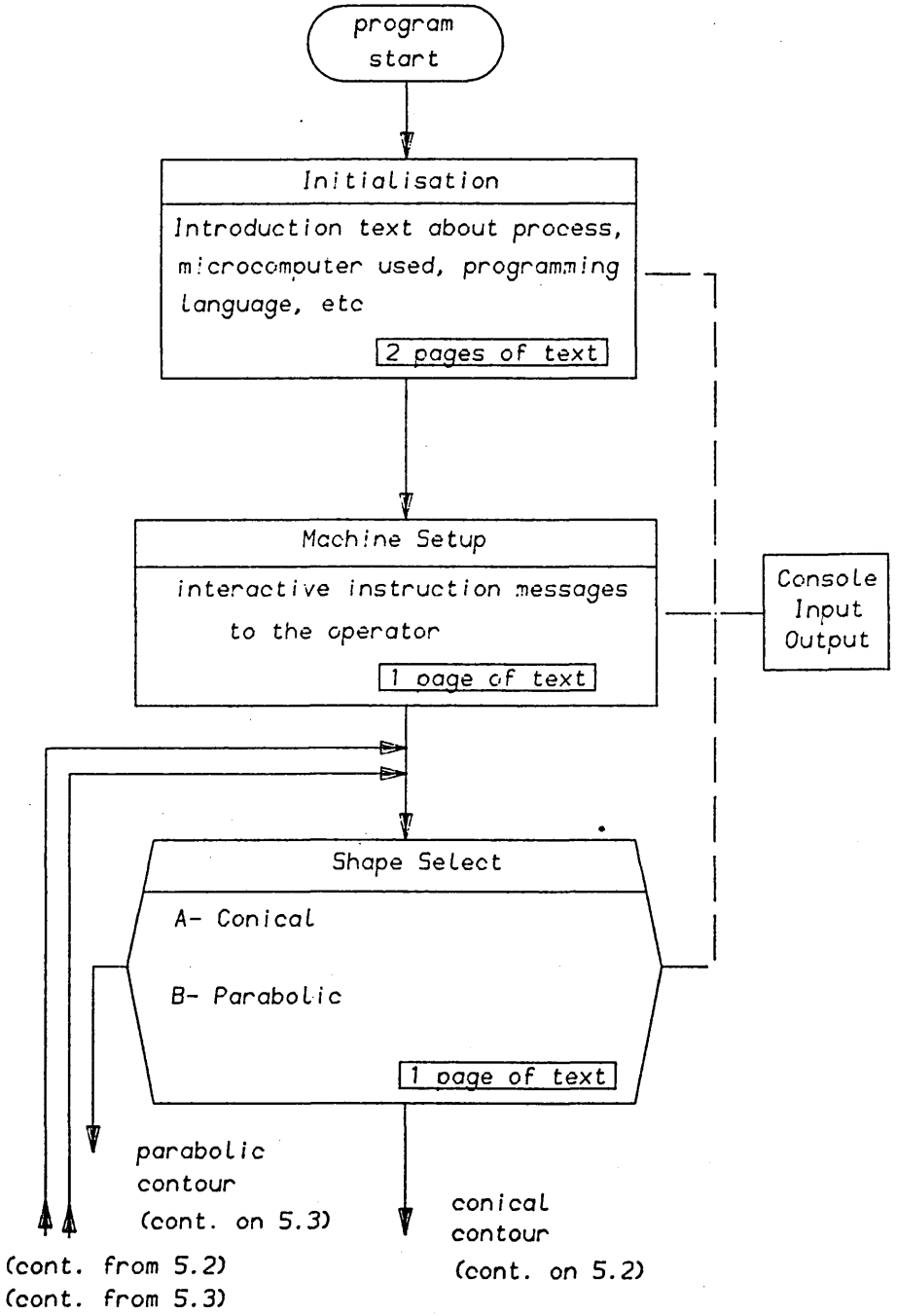


FIG 5.1: The program as seen from the operator's view (1)

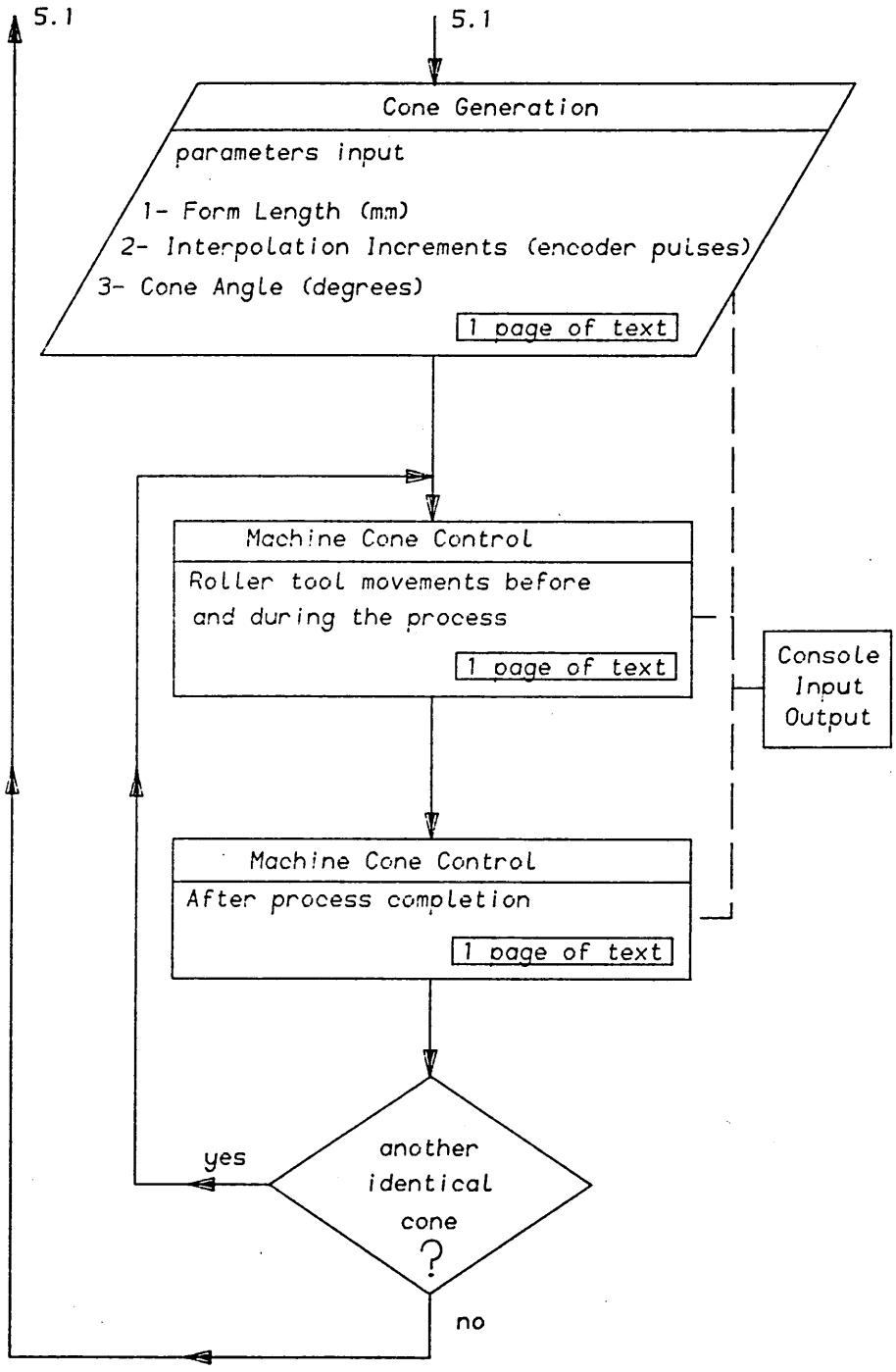


FIG 5.2: The program as seen from the operator's view (2)

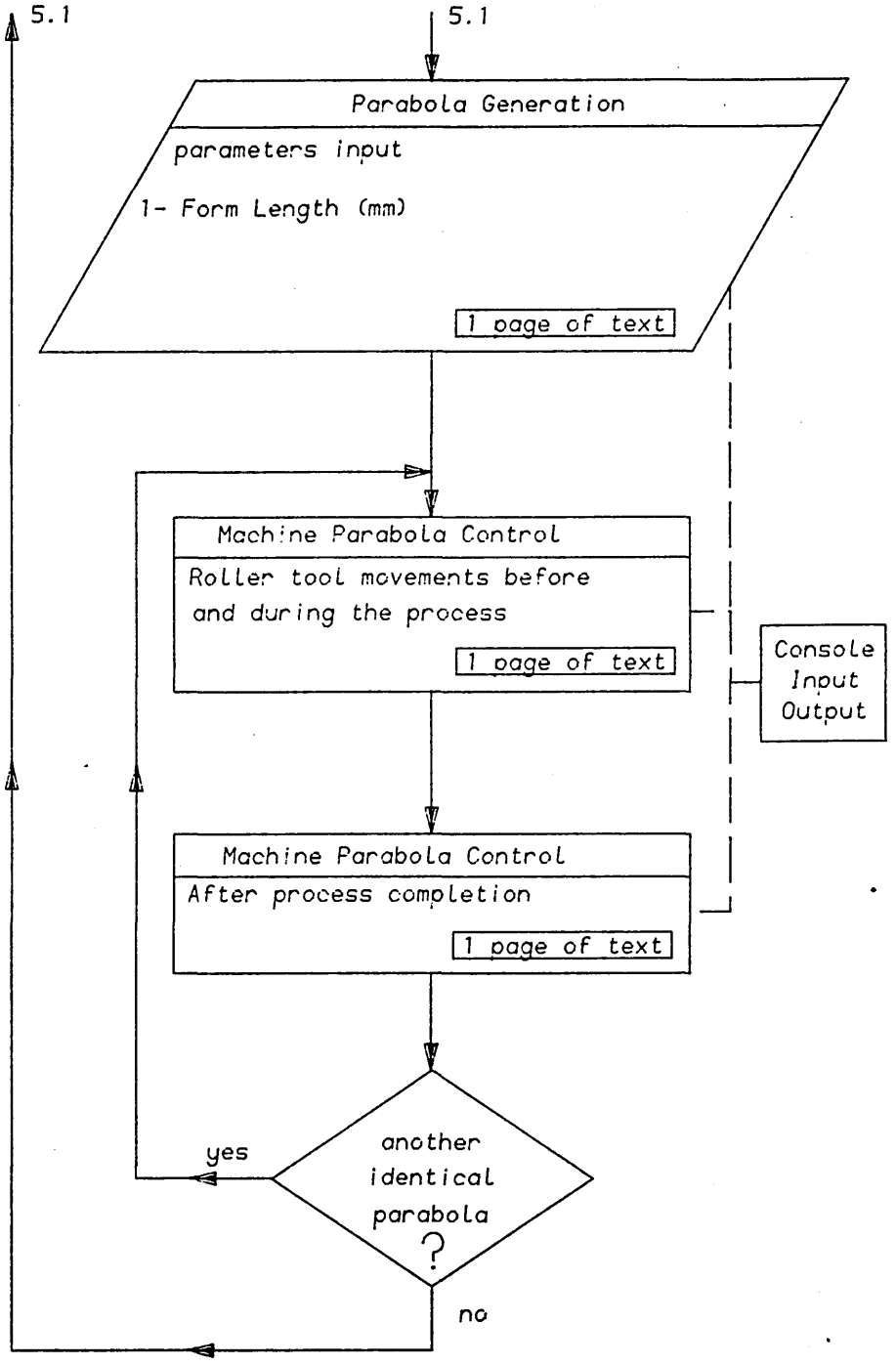


FIG 5.3: The program as seen from the operator's view (3)

After each module has been written, it is compiled, linked to other modules and then located into the relevant hardware address space. The In-Circuit Emulator is then invoked and the software tested without having to 'blow' an EPROM. If any faults are found, the user has to return to the high level language, make the necessary changes and then re-compile the module(s) and re-run under ICE until a satisfactory result is achieved, (see also section 2.4.3 and fig 2.1 chapter 2).

5.4 - 'Flow' module

This is the main module for the flow-turning process. It calls 'Initialisation', 'Machine Setup', 'Shape Select' and either 'Cone Generation' followed by 'Machine Cone Control' or 'Parabola Generation' followed by 'Machine Parabola Control'. The module is illustrated with the hierarchy chart in fig 5.4.

5.5 - 'Initialisation' module

This is the first module to be called by the main flow module. It performs all the necessary initial hardware resetting on the SDK-85. The SDK-85 resetting includes port configurations, resetting ACIA 1 and zeroing the SDK-85 display. To prevent unnecessary conversions by the ADC, the ADC convert signal is set high at all times until a conversion is required.

Likewise the bipolar DAC is initialised so as to keep the roller stationary. Zero volts are output to the valve controller which in turn sends a current proportional to this voltage to the proportional directional valve. The cylinder will remain in place until further commands are received from the software. The shaft encoder counter is also reset to zero.

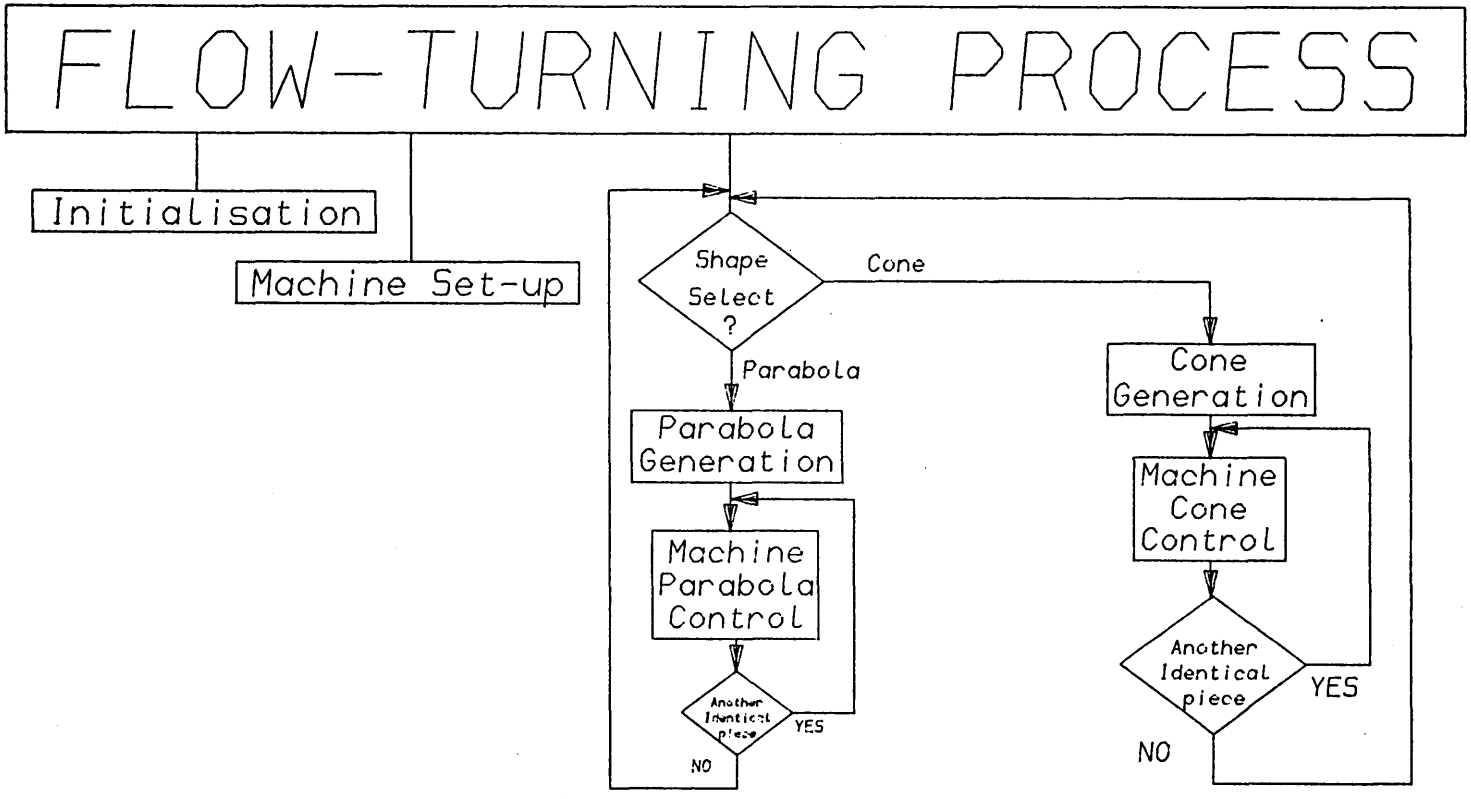


FIG 5.4: Flow-turning process hierarchy chart

5.6 - 'Console I/O' module

This is a utility module which contains all the console terminal input-output routines. Its procedures are called upon by the other modules.

'Console Out' is a procedure whereby a character is output to the VDU screen. The procedure is useful in displaying the introductory text and other text. All the characters were output through the serial channel ACIA 1.

'Message' procedure uses 'Console Out' to present messages on the VDU. To input one character from the VDU keyboard as for example during input of the operator's parameters in 'Cone Generation' module, 'Console In' procedure is invoked. In 'Decimal Value Input' procedure, a maximum of two digits can be input, tested for the correct character value and then stored. These input numbers represent the cone input parameters or parabola parameters.

5.7 - 'Machine Set-up' module

This procedure directs the operator to set up the flow-turning machine and the associated hardware. The instructions are displayed one by one on the VDU screen and the operator is expected to respond accordingly.

5.8 - 'Shape Select' module

This module displays a menu giving a choice of contour i.e the former profile or the specified mandrel. A choice of the two above mentioned contours is available.

5.9 - 'Cone Generation' module

This is a module whereby the cone contour variables are entered into the program memory. Cone variables are displayed on the VDU as their values are keyed in by the operator. If the value typed is not the desired one, it can then be rectified with the delete key on the VDU console. The routine allows either one or two digits to to be keyed in. After each parameter is properly entered, it must be followed by a RETURN on the VDU keyboard, see fig 5.5.

5.10 - 'Parabola Generation' module

This is a module whereby the parabola contour variables are entered into the program memory. It serves a similar purpose to the 'Cone Generation' module.

5.11 - 'Machine Cone Control' module

This is the module which actually controls the flow-turning process. All the control and monitoring of the flow-turning machine is carried out in this module and in particular the series of roller movements required to produce a conical shape is generated. The input data from the 'Cone Generation' module are passed to this module so that the movement is described according to the parameters specified. The movements include advancing, retracting the roller and following the required contour, as determined by the 'Conical Interpolation' routine.

The module holds the 'Shaft Encoder' procedure which reads the carriage position along the workpiece i.e the tool position (x-axis),

ADC INPUT

A routine to read the 12-bit analogue digital converter. Two ports are used namely

port 21H (1-8) the least significant bits.
port 23H (9-12) the most significant bits.

The conversion is initialised, then a given time delay has to elapse (for conversion to be completed).

The ports now can be read (to get digital values), which are joined together in a 16-bit variable namely (ADCINWORD).

The 1st and the 2nd bits are masked off to give 10 bits variable namely (ADCINWORD).

SHAFT ENCODER

A routine to read the carriage position at any time from start till leadscrew stops completely. The value is inputed through port 29H.

If the value read exceeds 255, then 1 is added to the high byte to give the total counts, that is the total number of pulses (i.e absolute position).

A 16-bit variable will be returned namely (CARRIAGE$POSITION) upon calling the typed procedure.

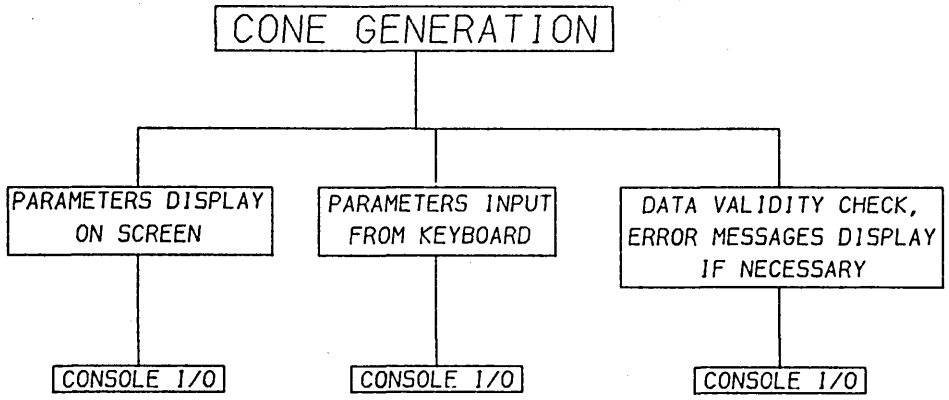


FIG 5.5: ADC Input, Shaft Encoder and Cone Generation procedures

while 'ADC Input' procedure reads the radial position (y-axis) via the rectilinear transducer, see fig 5.5.

'Tool Advance' is a procedure which is called upon by the 'Machine Cone Control' procedure to advance the tool to the position specified by the parameter, see fig 5.6.

'Tool Retract' is a procedure which is called upon by the 'Machine Cone Control' module to retract the tool to the position specified by the parameter, see fig 5.7.

The 'Cone Interpolation' routine is needed to calculate the retract distance in the y-axis corresponding to the distance moved in the x-axis. The distance moved by the carriage is the independent variable upon which the tool movement depends. The greater this distance is, the larger the retract distance will be, see figs 5.8 - 5.11.

5.12 - 'Machine Parabola Control' module

This module controls the flow-turning process for a parabolic shape and thus serves a similar purpose to the 'Machine Cone Control' module. The 'Shaft Encoder', 'ADC Input', 'Tool Advance', 'Tool Retract' routines are the same as those used in the 'Machine Cone Control' module. The interpolation procedure 'Par Interpolation' will be explained in section 5.14.

5.13 - 'Display' module

A program enhancement was incorporated to indicate the position of the carriage at any point in time by displaying the number of shaft encoder pulses received. This was done to provide a check on the shaft encoder and counter circuit by means of a visual comparison.

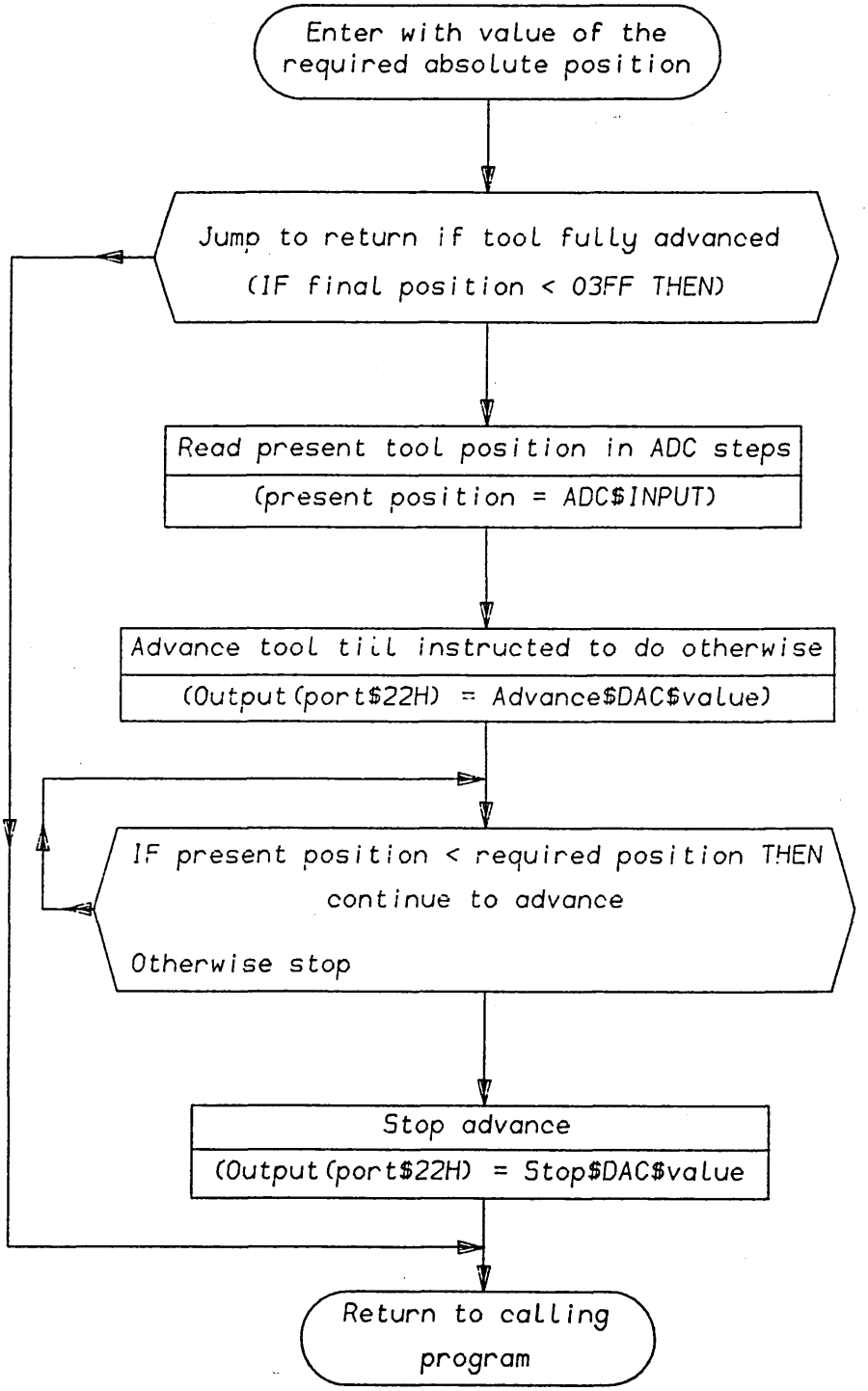


FIG 5.6: Tool Advance procedure flowchart

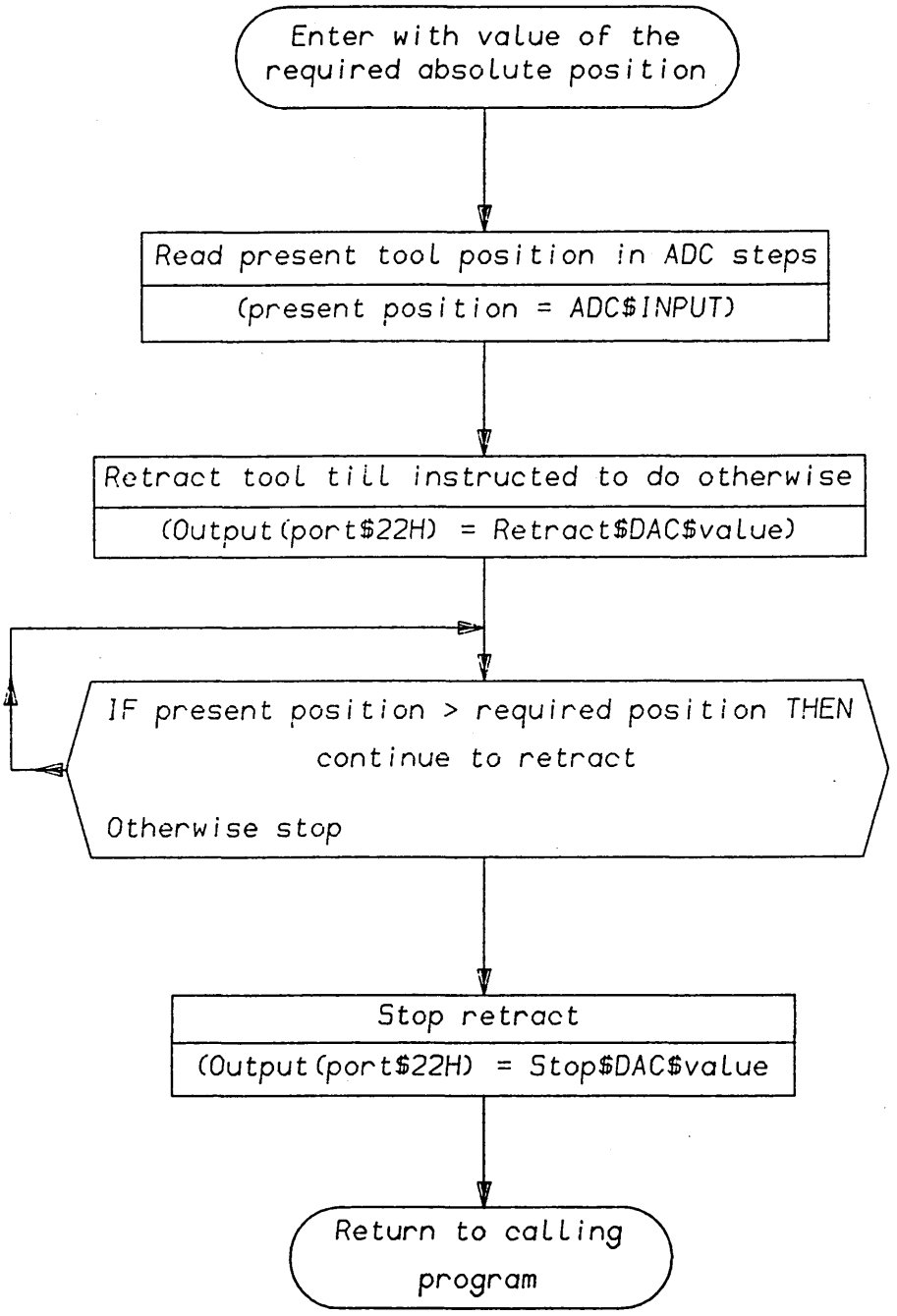


FIG 5.7: Tool Retract procedure flowchart

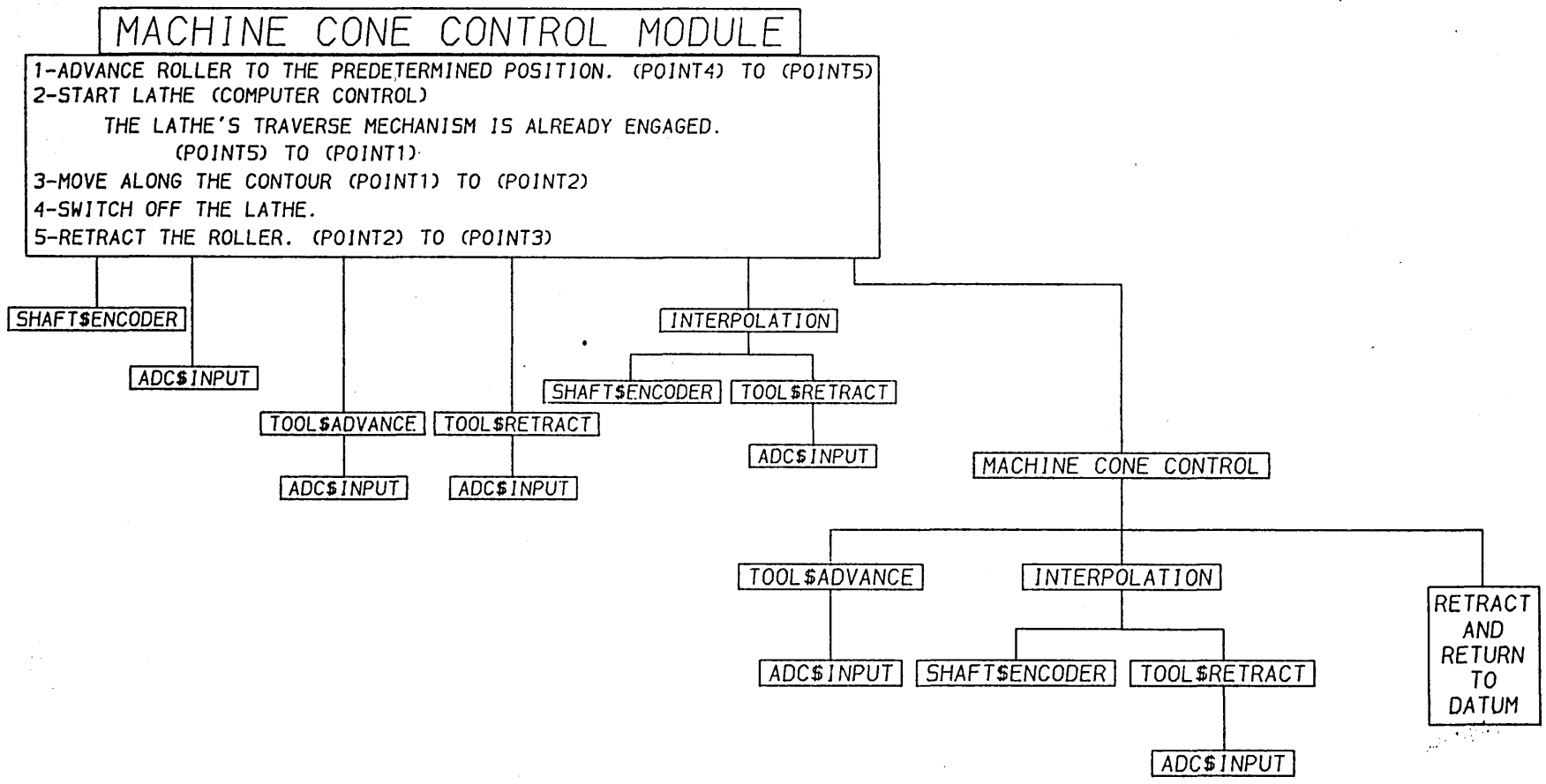


FIG 5.8: Machine Cone Control module

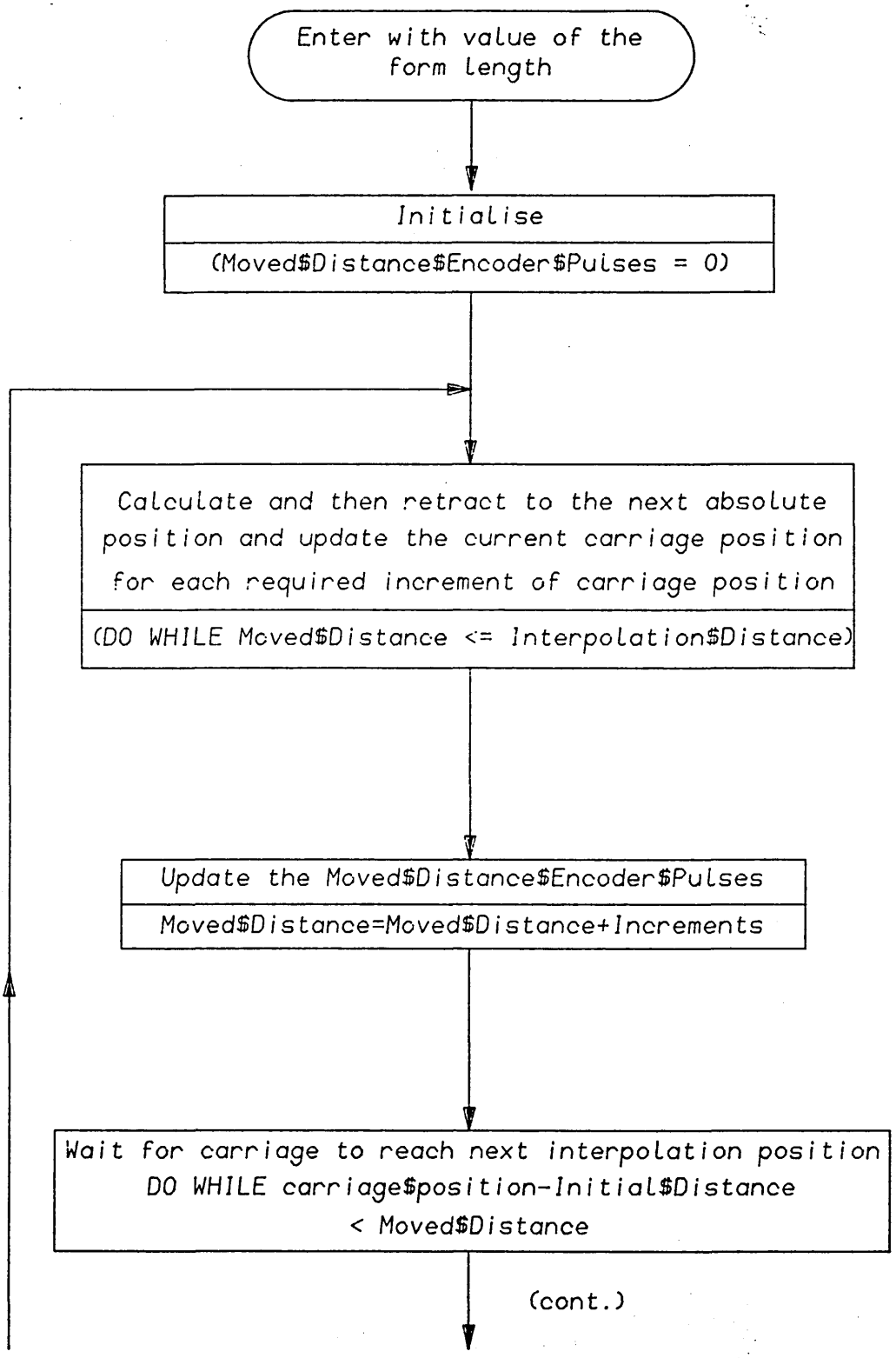


FIG 5.9: Interpolation procedure flowchart (1)

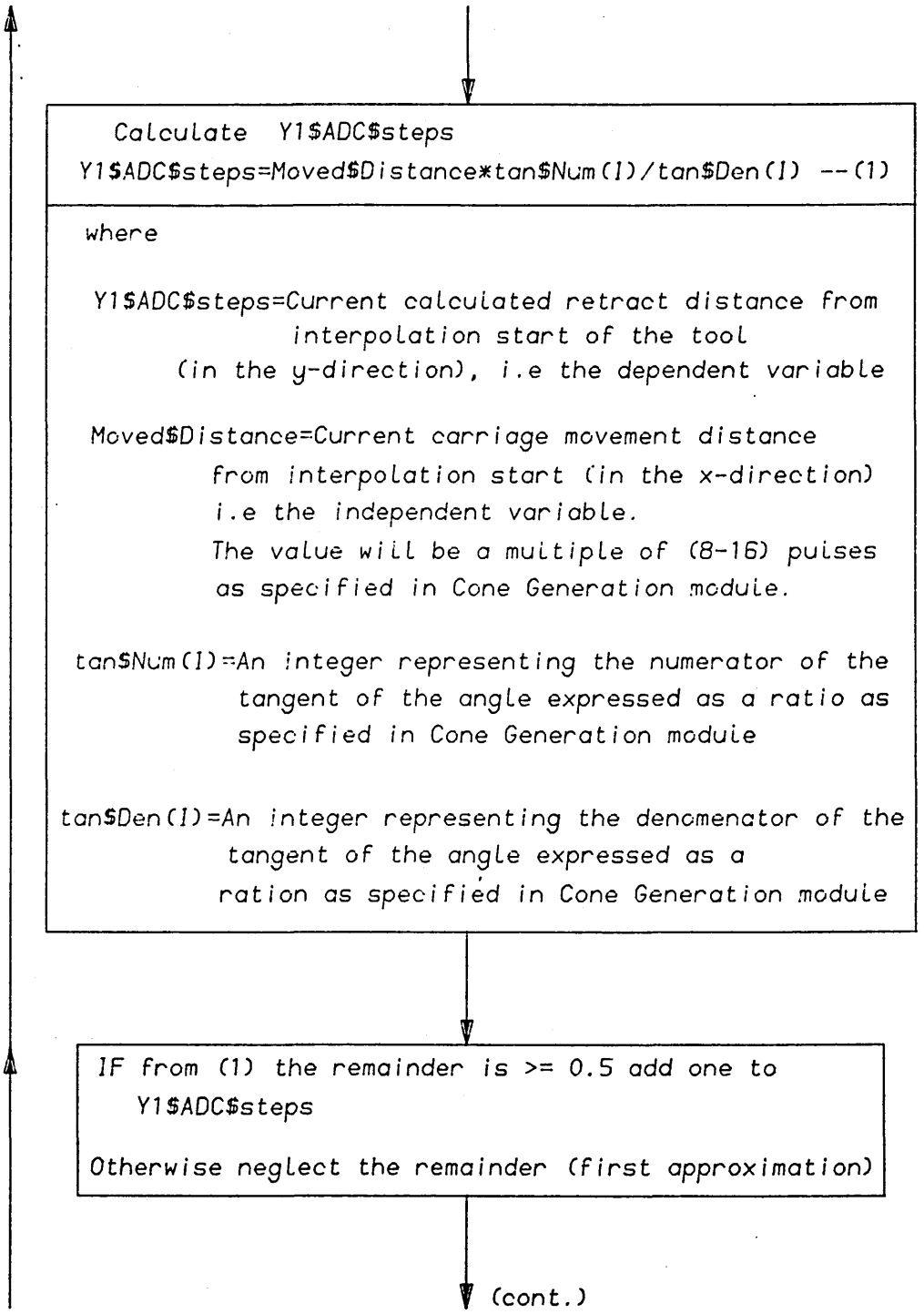


FIG 5.10: Interpolation procedure flowchart (2)

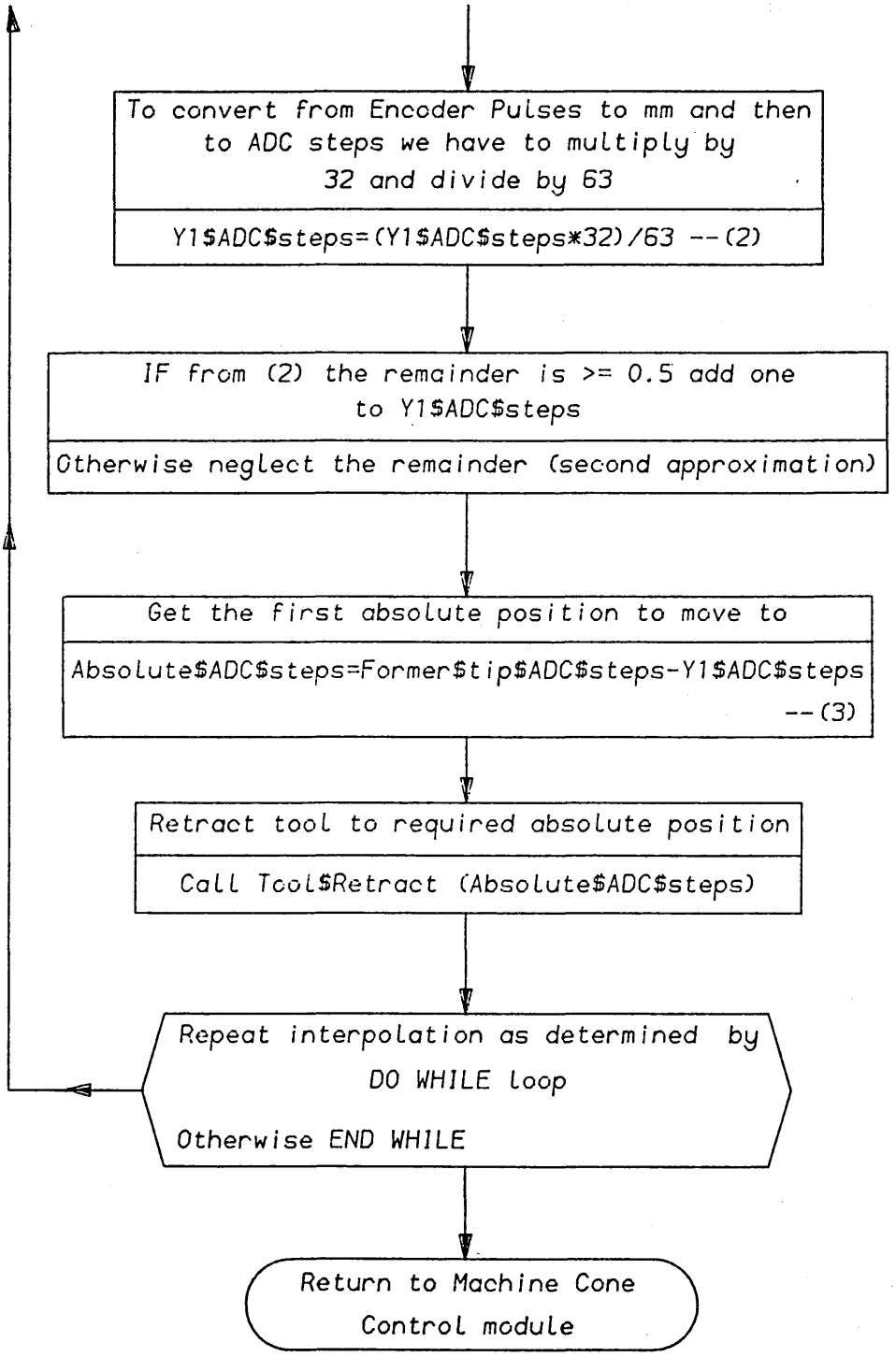


FIG 5.11: Interpolation procedure flowchart (3)

The number of pulses was displayed on the SDK-85, the 4 digit display indicating the total number of pulses and the 2 digit the number of pulses in the current byte. The two monitor routines 'UPDDT' and 'UPDAD' required for this purpose were included in the main program. These were the only modules written in assembly code. The 'Initialisation' module clears both fields upon executing the program.

5.14 - 'Par Interpolation' procedure

This routine is different from 'Con Interpolation', since the points of the parabolic contour were calculated beforehand and stored in a look-up table. The routine had to be done this way since the PL/M 80 language could not calculate the square root in the parabola equation adequately, owing to the lack of floating point arithmetic.

First, the parabolic equation constant was determined and then the interpolation points were calculated every 16 encoder pulses (equivalent of 1.016 mm). The interpolation points (real numbers) were represented as ratios of integers which could be held as PL/M 80 variables.

To find these integer ratios, a program was written (in Basic language) which estimated and printed out the ratios (see appendix 3 for the program).

5.15 - Other program notes

The tangents of the cone angles had to be introduced in a form which could be held as a PL/M 80 variable. Since this language could not deal with real numbers, the tangents had to be expressed as ratios consisting of two integers. These two integers had to be found by trial and error, since they had to be chosen to give the greatest

possible accuracy, whilst not permitting the ultimate limit of 65535 (for sixteen bit integers) to be exceeded during the evaluation of the expressions in which they are involved. All the calculations had to be worked out in advance to avoid overflow. Similarly, the form length had to be converted from millimeters to encoder pulses.

Both the tangent and the form length tables were incorporated in 'Machine Cone Control' module in array form where the required value could be obtained by referring to that array index. Fig 5.12 shows the cone and the parabola procedures.

5.16 - Software testing

The main modules, 'Initialisation', 'Console I/O', 'Shape Select', 'Cone Generation', 'Machine Cone Control' were designed first. 'Display', 'Parabola Generation' and 'Machine Parabola Control' were added later with other alterations in 'Machine Cone Control' module.

Software testing was aimed at checking the flow-turning controller performance, locating visible sources of error, data manipulation testing, a thorough examination of the magnitude of integer numbers resulting from mathmatical operations, such as multiplication, uncovering unexpected snags and diagnosing possible shortcomings.

Each movement made by the tool had to be checked for correspondance with the requested flow-turned profile. The roller path can be seen in fig 5.13. The y-axis distance (4-5) was found to be incorrect. The same applied to the x-axis distance (5-1). These errors were due to hardware. Some minor faults were rectified immediately while other more serious problems were solved at a later stage (see chapter 6). Tool movement along the conical contour (1-2) was not precisely checked at this point, although it appeared to be consistent with a conical profile.

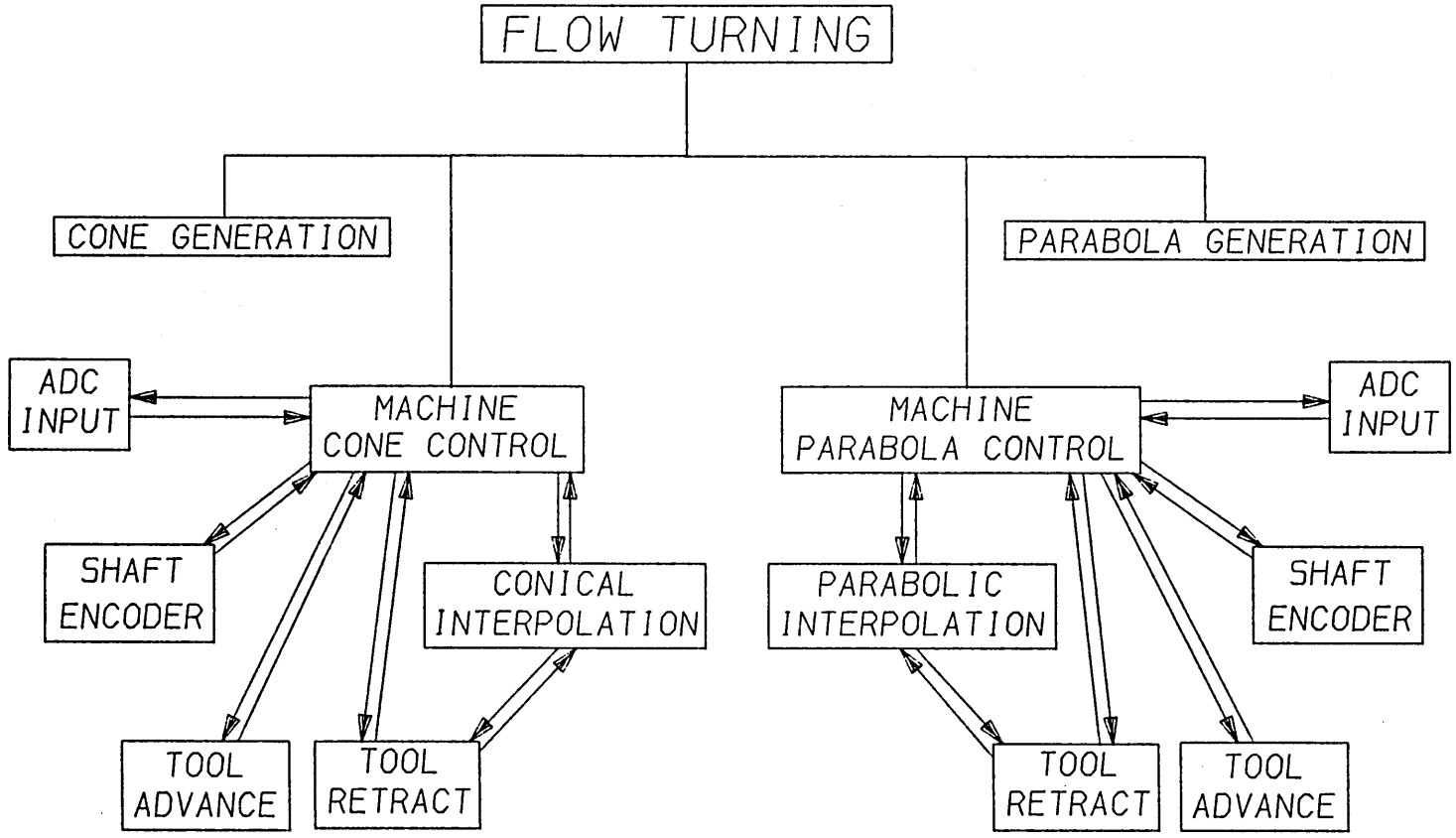


FIG 5.12: Cone and Parabola procedures

First movement

Point4 to Point5

Advance roller

Second movement

Point5 to Point1

Move left

Third movement

Point1 to Point2

Conical contour

Forth movement

Point2 to Point3

Retract roller

Fifth movement

Point3 to Point4

Manual movement

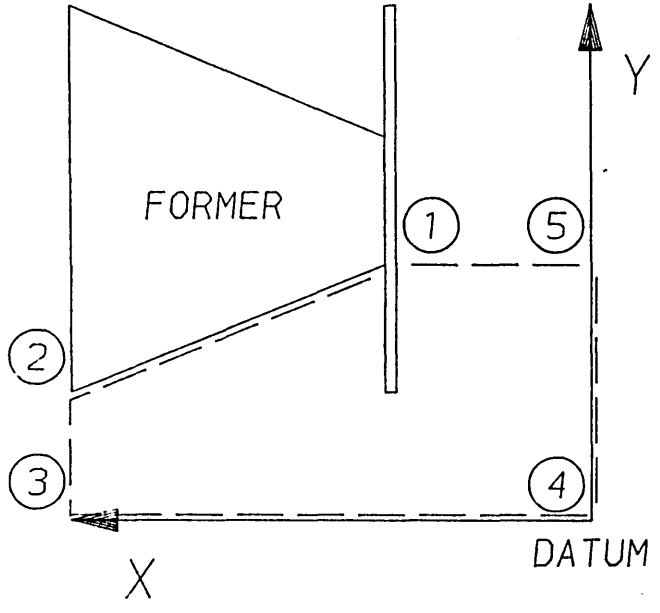


FIG 5.13: Roller prescribed path

Chapter 6: Rig commissioning

6.1 - Introduction

The initial stages of program development and debugging were accomplished using a development system with an in-circuit emulator. Subsequently the software was integrated with the hardware and tested on the rig in real time. The modifications found necessary during the development process are detailed below.

6.2 - Testing observations (version 1)

After transferring the program to the EPROM chips and running it in the real application, the following faults were observed:-

1- The advance distance was not correct (the observed distance moved was greater than the calculated distance).

2- The counter reading was inconsistent.

3- The contour had a greater conical angle than the input value of 30 degrees (former angle).

4- The tool did not retract to the datum position, but in fact moved further.

To identify these errors, hardware components were checked with special software routines. Some of the equipment used is shown in plate 6.1.

6.2.1 - Linear transducer recalibration

The transducer was found to have an unstable supply voltage. Instead of the correct calculated value of 11.719 volts it was 8.19

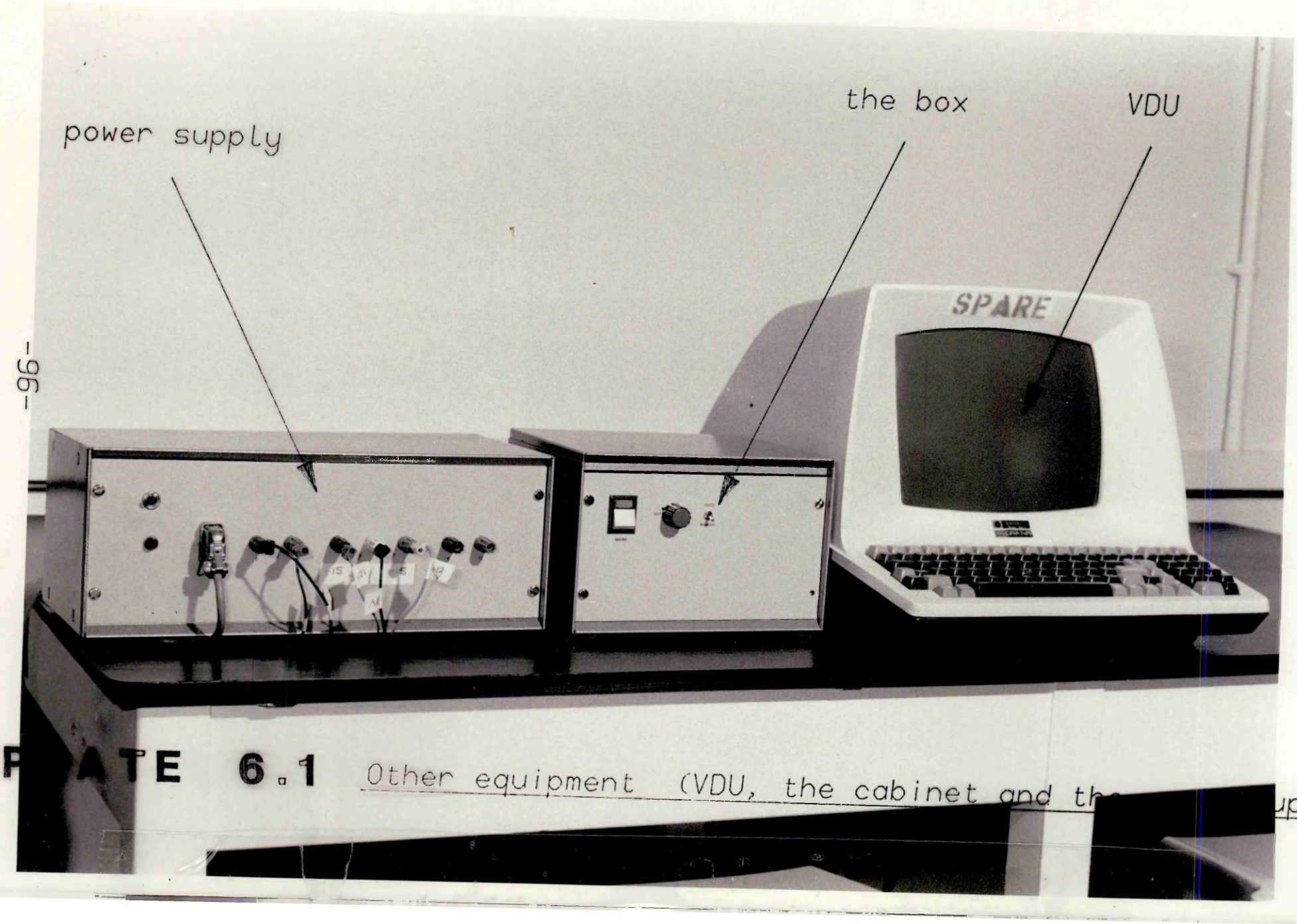


PLATE 6.1 Other equipment (VDU, the cabinet and the ... upp

volts. The voltage was brought back to 11.719 by adjusting the power supply potentiometer.

Rerunning the program to ensure that transducer resolution had the correct value along its entire length, a special routine to test the transducer performance (PROG2) was loaded manually into the SDK-85, (see appendix 1 for PROG2).

Using a micrometer, the accuracy of transducer measurement was checked and the resolution was found to be within the specified tolerance throughout the range.

It was also found that the resolution remained constant throughout the length. An illustration is provided in fig 6.1. Another problem was power supply drift, which had to be measured over a period of time to ascertain a reliable transducer performance. Mains to the power supply had to be checked as well.

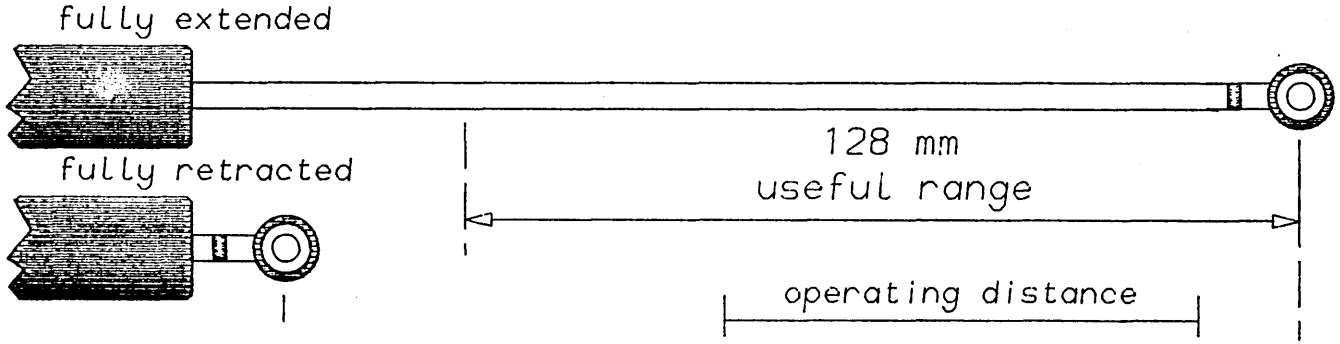
The power supply had a tendency to shift upwards over a period of time. An example of the fluctuation over a 30 minute period was from 11.719 to 11.85 volts. The problem was overcome by using a separate high stability d.c. power supply.

The transducer in conjunction with the ADC enabled movements of 1/8 mm to be detected corresponding to an ADC resolution of 10 mV per least significant bit.

6.2.2 - Counter test

A persistent and unidentifiable fault occurred on the counter board, so another counter board had to be made. When tested it was found to be satisfactory.

To verify counter operation, a mark was made with a pen on the carriage edge; then upon running the program to displace the carriage, the distance was measured by making another mark at the new carriage



	Transducer Length in (mm)	0	10	20	30	40	50	60	70	80	90	100	110	120	130	140	150
*	Reading 1 (hexadecimal)	3FF	3FF	3FF	3C5	374	320	2D0	283	228	1CE	178	120	106	B2	60	0
*	Reading 2 (hexadecimal)	3FF	3FF	3FF	3CD	37C	328	2D8	28B	230	1D6	180	128	10E	BA	68	0
	Difference (resolution)	0	0	0	8	8	8	8	8	8	8	8	8	8	8	8	0

* Two 1mm apart readings shown on the SDK-85 display using the micrometer head

FIG 6.1: Transducer test with PROG2 program

position. This was repeated several times and the distance was found to be within acceptable tolerance in each case.

The distance travelled corresponded to the number of pulses stored in the 'Machine Cone Control' module and a visual check on the number of rotations of the leadscrew agreed with this.

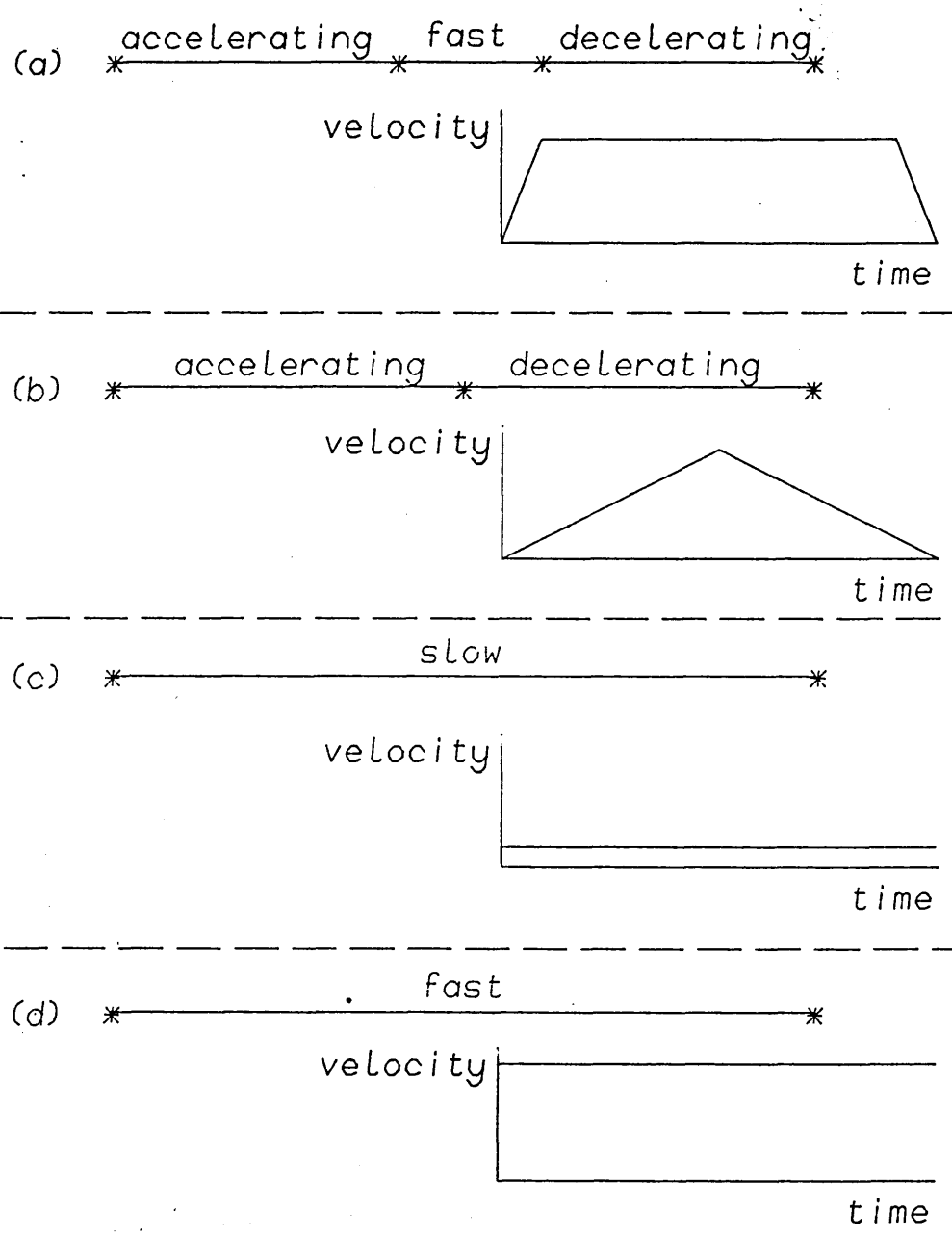
6.2.3 - DAC test

The roller forming tool can be moved (in the y-axis) either manually by placing the switch in the manual position and then twisting the potentiometer clockwise or anticlockwise to advance or retract the tool, or it may be automatically driven by the computer, by moving to the auto position.

When the power was switched on while in the manual switch position, and without the program running, the tool remained stationary until it received an advance or retract signal, but in the auto position, it fully advanced instantly to the far end, pushing the limit switch with the risk of damage. This effect was due to the logic high signal sent out by the DAC to the valve controller as a result of microprocessor output being high until initialised by the program. To overcome this, the sequence of operating the oil pump and then running the program had to be reversed i.e the program execution had to be done first, then the pump was switched on.

In version 1, advancing and retracting of the tool were done at the maximum speeds. However this may not be the most efficient method of performing for this application owing to persistent tool overshoot. Other possible ways of controlling tool movement, (see fig 6.2.) are by

(a) Acceleration from zero to maximum velocity, constant velocity



Advance distance either (c) or (d)

Retract distance (c) only

(b) shouldn't be used

FIG 6.2: Cylinder movement techniques

for a period, followed by deceleration to a halt at full distance.

(b) Acceleration then deceleration (no speed limit).

(c) Slow movement throughout traverse.

(d) Maximum velocity from start to finish.

All the above-mentioned methods of controlling tool movement were possible implementations, but options (a) and (b) were dismissed because of

1- The need to simplify the control algorithm in order to reduce the response time.

2- The lack of ability to control speed adequately by varying the voltage.

The relationship between the voltage applied to the valve controller (which is converted subsequently into a current fed to the proportional valve) and the corresponding equivalent decimal value (used in programming) to the cylinder speed for tool advance and retract is given in fig 4.13 chapter 4. The velocities quoted were found experimentally by measuring the travel time for a predetermined piston stroke of 148 mm.

These figures show firstly that large voltage thresholds exist for both tool advance and retract before tool movement occurs and secondly that tool speeds are too critically dependent on voltage in the area of these threshold levels for reliable measurement. This means in practice that velocities of less than 12.6 and 27 mm/s could not be obtained for tool advance and tool retract movements.

It was decided to choose (during the process) a constant tool speed which was well below the maximum speed and could be tuned to give a satisfactory contour.

6.2.4 - Contour shape

The conical angle of the contour was found to be larger than set in version 1 of the program (see section 6.2). To determine the reason for this, the 'Interpolation' routine values (retractions) were calculated and compared with the retractions observed. An example of the calculated interpolations can be seen in fig 6.3. The actual retractions were significantly greater than the set values. The assumptions initially made were therefore reviewed for possible errors.

On reviewing the assumptions it was realised that the software had been written in such a way as to allow relative distances to be specified in 'Tool Advance' and 'Tool Retract' (which are included in 'Machine Cone Control' procedure). This method proved to be error-prone, as it carried a cumulative inaccuracy to the next distance to be moved (without checking the required position).

Each call to the 'Tool Advance' and 'Tool Retract' procedures specified the amount of movement from the current position. Also since the tool was not decelerated and the tool stop condition was only initiated when the tool reached the final position, it was likely that tool overshoot would occur at each interpolation. This overshoot was not taken into account in the calculation of the next interpolation and so the error increased continuously to a maximum at the end (see fig 6.4 a).

Another method of specifying the positions was by the use of algorithms ('Tool Advance' and 'Tool Retract') based on absolute position, whereby the final location to be moved to was specified for each increment (retraction). In this way, if an error occured in determining the first point it would not be passed on to the next point.

To show a sample calculation of the Interpolation routine, the following data are used:-

1- Form Length = 50 mm
= 787 pulses see figure 5.33

2- Increments$Encoder$Pulses = 16 pulses

3- Cone Angle = 30 degrees
tan$Num(30) = 26
tan$Den(30) = 45 see figure 5.32

For the above 2 and 3 also see the program listing in Appendix 2 Cone Generation and Machine Cone Control modules. Figures 5.29 to 5.31 for Interpolation flowcharts.

Sample Calculation

Y1ADCsteps=Moved$Distance*tan$Num(30)/tan*Den(30) --(1)

$$= 16 * \frac{26}{45}$$

$$= 9.24$$

IF from (1) remainder is >= 0.5 THEN add one to Y1ADCsteps
Y1ADCsteps = 9

Y1ADCsteps=Y1$ADC$$steps*32/63 --(2)

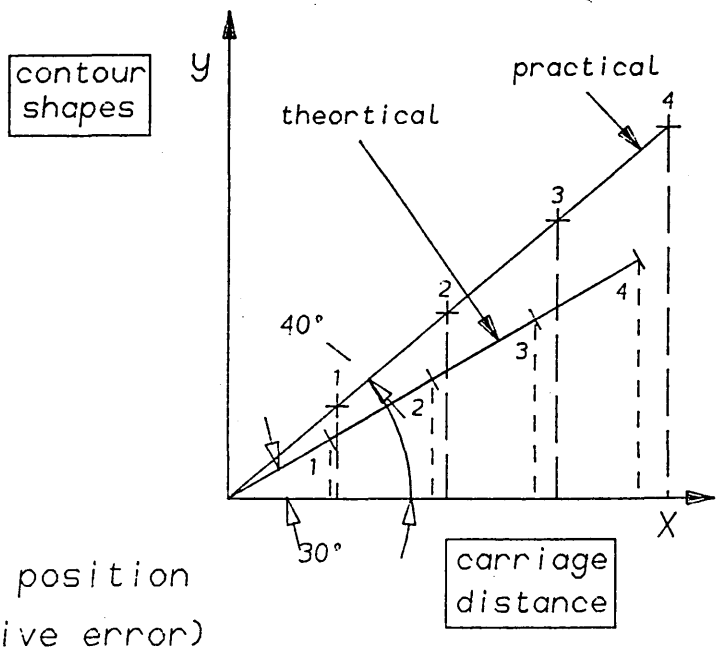
$$= 9 * 32/63$$

$$= 4.57$$

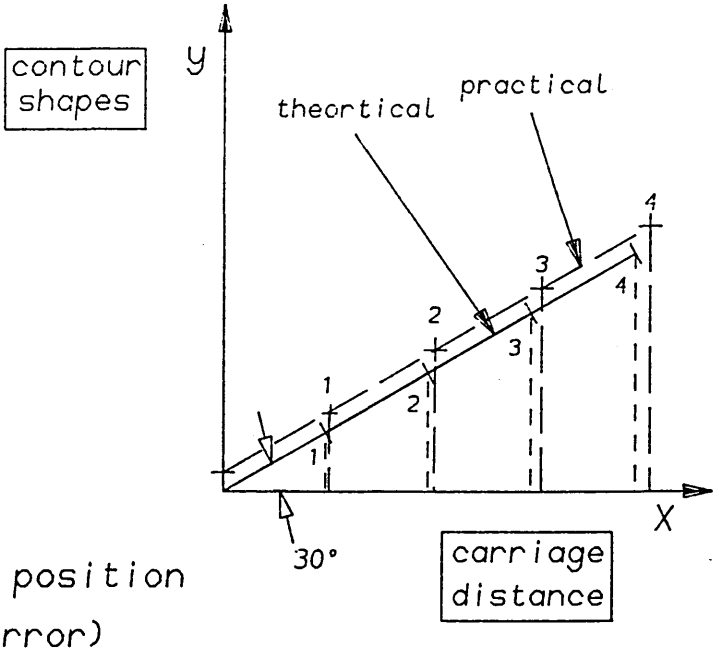
IF from (2) remainder is >= 0.5 THEN add one to Y1ADCsteps
Y1ADCsteps = 5

The calculated values are shown in figure 6.4.

FIG 6.3: An example of the calculated retract distances for the Interpolation routine



(a) Relative position
(cumulative error)



(b) Absolute position
(steady error)

FIG 6.4: Possible options for locating position

It should be noted that there would be a slight deviation from the conical contour based on the absolute position since the CPU could not continue counting the number of pulses from the encoder (carriage position) while it performed other arithmetic manipulations at the same moment (see fig 6.4 b).

6.3 - Further software modifications and enhancements

The major faults and shortcomings of version 1 of the program became more apparent when tests were carried out on the flow-turning rig. The alterations included the following:

1- Resetting and enabling the counter chip was done in 'Initialisation' module only. This has to be done in 'Machine Cone Control' module ('Machine Cone Control' procedure) for the successive turnings.

2- The data and the address fields on the SDK-85 were not used in version 1. An improvement was made by displaying the counter reading (the contents of port 29H), which is a byte variable and the cumulative count (carriage position), which is an address variable on the data and the address fields respectively. These displays were initialised to zero at the start of the flow-turning process and continuously updated until the end of 'Interpolation' procedure.

3- The advance and the retract speeds (see section 6.2.3) were decreased to reduce the tool overshoot.

4- A new approach based on calculation of absolute rather than relative position during 'Interpolation' was implemented. This necessitated alterations in 'Tool Advance', 'Tool Retract' and 'Interpolation' and other pertinent module modifications.

6.4 - Spindle speed and feedrate ranges

From section 1.7 chapter 1, the flow-turning surface speed range was specified to be between 1000 to 2000 ft/min, when converted to rpm this gives a minimum speed of 1591 rpm and a maximum of 3183 rpm. These speeds are not available on the lathe since the maximum is 750 rpm, also the software requirement necessitated using lower speeds than 750 rpm.

From section 1.7 we also have for best results (excellent surface finish), the feedrate range is between (0.0508 - 1.12) mm/rev. For acceptable results (surface finish not important), the feedrate range is (0.762 - 1.27) mm/rev.

It is clear from the above that the acceptable and the best feedrates range on the lathe will lie between (0.226 - 1.27) mm/rev.

6.5 - Selection of the appropriate roller speeds

In the first version of the program, maximum cylinder speeds were used. However to achieve the best contour, two parameters had to be considered:-

1- Lathe spindle speeds and feeds.

2- Roller tool retract speeds.

A combination of the appropriate lathe spindle speeds and the carriage feedrates was selected from the available values. It was necessary that the roller tool had time to finish retracting the calculated increment for each interpolation point before the carriage reached the next point for interpolation. This imposed a lower limit on the choice of roller speed, as set out below.

$$\frac{\text{Roller retract increment}}{\text{Roller speed}} < \frac{\text{Carriage increment}}{\text{Carriage speed}}$$

where:

carriage speed = carriage feed * spindle speed

and

$$\frac{\text{roller retract increment}}{\text{carriage increment}} = \tan(\text{cone angle})$$

Therefore

minimum roller speed =
carriage feed * spindle speed * tan (cone angle)

The time taken by roller to complete incremental movement on y-axis is less than the time taken by the carriage on x-axis. The roller speed was selected to comply with the above condition, and could be adjusted by varying the digital output signal to the DAC. The digital values used in version 2 of the program were 177 and 73 for advance and retract respectively, and these complied with the above condition and also reduced the overshoot to an acceptable level.

6.6 - Testing observations (version 2)

After the changes explained in section 6.2.4 had been completed, the second version of the program was observed to give a satisfactory contour. However, some further comments about this program version are relevant.

When testing this version of the software, the Interpolation routine appeared to malfunction persistently. The routine was checked again for possible errors without success. During running of the program, it was observed that the number of retractions actually moved by the roller tool did not correspond to the calculated number of retractions. It was concluded that this effect was due to the roller tool overshooting. This tendency to overshoot caused excessive incremental tool movement resulting in zero movement at some steps

because the actual tool position already exceeded the required position. This effect was relatively easy to identify as the carriage position address variable was displayed on the microcomputer. To rectify this overshooting, the cylinder speed was slowed down by adjusting the proportional valve amplifier. On rerunning the program the problem was found to have been minimized.

Another problem which occurred was in defining the datum position at which 'Interpolation' should start. This was due to the variation in angular position of the leadscrew when the traverse was engaged every time the 'Interpolation' started. To overcome this problem, a micro switch was fitted on the carriage path at a position corresponding to the former tip and the software was amended so that 'Interpolation' commenced at this position.

6.7 - Experimental testing

The modified rig described in chapter 3 was used in the experimentation tests which were aimed at verifying the rig effectiveness for producing a successful controller to control the flow-turning process.

Some checks on the rig had to be made before any tests could be carried out: in particular the former had to be checked for concentricity with a dial gauge.

The sample was held firm against the former by the tailstock. The roller tool had to be brought close to the former at a distance of about 10-20 mm. The tailstock was placed as far forward as possible on the lathe bed so as to maximize rigidity by reducing the extended part of the tailstock holding the disc. However a large amount of tailstock extension was still required which, as observed later, produced an undesirable effect on the samples.

The advance tool distance from the datum to the former tip, i.e the start point along the y-axis, was 72 mm of the transducer extension. The corresponding forming start point along the x-axis was initiated by the carriage actuating a limit switch clamped on the lathe bedway. This could also be manipulated manually to obtain the correct starting position.

After forming the disc, the roller retracted to the datum, the carriage movement ceased gradually after the lathe has been automatically switched off. The carriage traverse then has to be disengauged and the carriage returned manually to the starting position close to the tailstock ready for the next run. The tool movements are shown diagramatically in fig 5.13 chapter 5.

The forming process was initiated from the SDK-85 microcomputer keyboard and then the rest of the process was monitored from the VDU. For repeated samples the software provides a facility to avoid keying in the same parameters again and again. After fixing the workpiece and switching on the power, the program can be executed on the SDK-85 board as follows:

1- Enter the interrupt service routine starting address at 20C8H and 20CDH (three bytes each C3H, F0H and A7H).
2- Press 'GO 8000H' followed by 'EXEC' keys.
3- Enter the parameters as required.

If the roller deviates from its prescribed path, then VECT INTR key on the SDK-85 board or the emergency stop switch on the lathe front should be pressed. The vector interrupt routine is shown in appendix 2.

Throughout the test program, different grades of greases were used in the experimental work. For the 162 samples produced three different

types were used, namely G.56/T grease Silkolene, Simnia grease 0 Shell (as lubricant 1) and Garia C oil Shell (as lubricant 2).

Before performing the actual tests with the samples, the contour was plotted using an adapted dial gauge pedestal fitted with a ball point pen. This involved fixing the moving carriage with the ball point pen in contact with a sheet of paper mounted on a rigid plate on the lathe bed.

The conical contours were examined using the maximum and the minimum values of the software parameters governing the resolution of carriage position. Figs 6.5 and 6.6 show the plotted contours.

With each of these resolutions, the conical test angles of 30,35,40,45 were drawn. These curves were then checked against the above angle values and found to be correct.

The same procedure was repeated with the parabolic contour and the plot is shown in fig 6.7. However in this case a fixed resolution of 16 encoder increments was used throughout.

After verifying the plots, tests were conducted to produce actual flow-turned samples. After concluding the tests each of the samples was measured. Three measurement tests were performed, namely the cone angle, the reduction in thickness and the ensuing hardness. The test programme is shown in fig 6.8.

A Hilger and Watts universal optical projector with a magnification of 15 was used to measure the cone angle. The thickness was measured using a MITUTOYO digital dial gauge linked to a DP1 dataprocessor. Hardness tests were carried out using the Vickers Pyramid Hardness Tester.

cone resolution = 8 encoder pulses

spindle speed = 48 r.p.m.

cone length = 50 mm

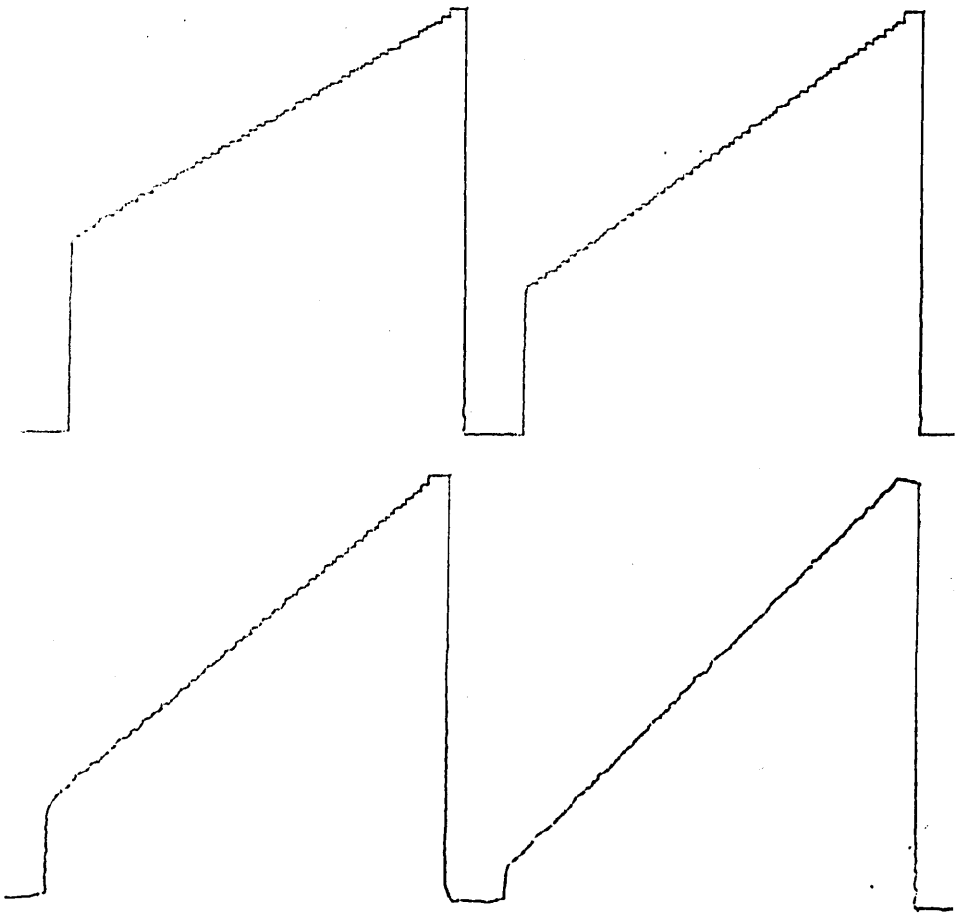
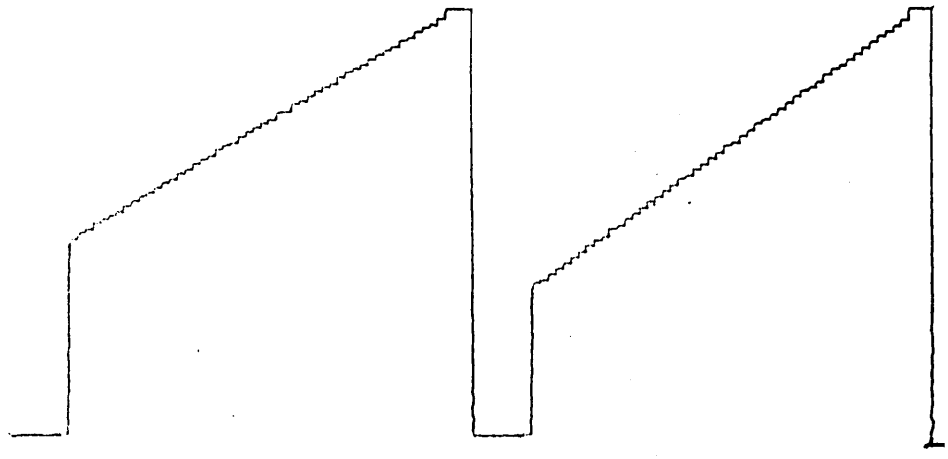


Fig 6.5: Conical contours with 8 encoder pulses

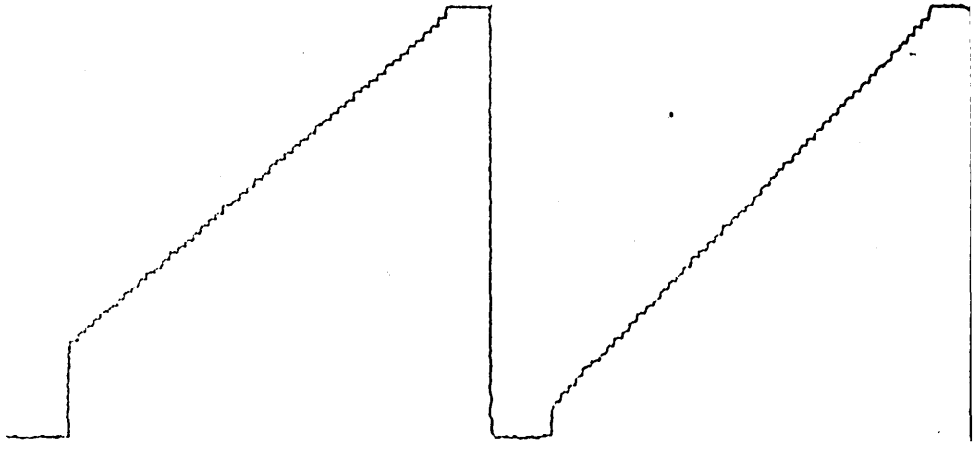
cone resolution = 16 encoder pulses

spindle speed = 48 r.p.m.

cone length = 50 mm



(a) 30° (b) 35°



(c) 40° (b) 45°

Fig 6.6: Conical contours with 16 encoder pulses

parabola resolution = 16 encoder pulses

spindle speed = 48 r.p.m.

different parabola lengths

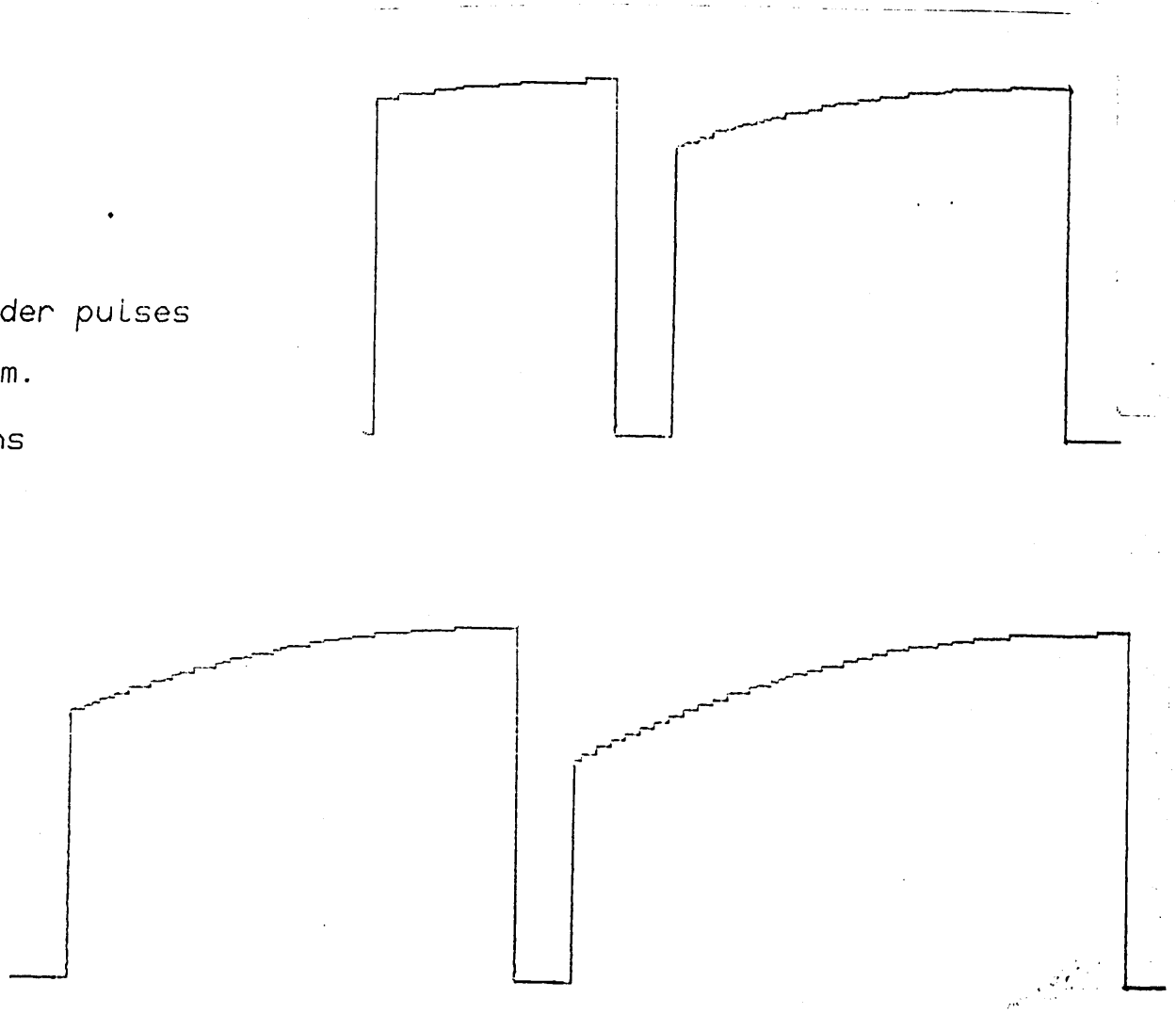


Fig 6.7: Parabola contour with 16 encoder pulses

SHEFFIELD CITY POLYTECHNIC

DEPARTMENT OF MECHANICAL AND PRODUCTION ENGINEERING

COMPUTER AIDED FLOW TURNING

by

A H Mohamad

Final testing on the rig

Testing with a conical former and with cone parameters as follows:-

1 Former angle = 30 degrees
2 Increments (distance between two interpolations) = 16 encoder pulses
3 Form length = 40 mm

First test

Choose a spindle speed = 95 rpm
Roller movement angles (contour)

30 degree	3 specimens
35 degrees	3 specimens
40 degrees	3 specimens
45 degrees	3 specimens

Second test

Choose roller movement angle (contour) to be 40 degrees

Spindle speed 1 = 95 rpm	3 specimens
Spindle speed 2 = 135 rpm	3 specimens
Spindle speed 3 = 186 rpm	3 specimens

Third test

Choose a spindle speed = 95 rpm
Roller movement angle = 30, 35, 40, 45 degrees

No lubricant (dry)	30, 35, 40, 45	3 specimens at each angle
Lubricant 1	30, 35, 40, 45	3 specimens at each angle
Lubricant 2	30, 35, 40. 45	3 specimens at each angle

Fourth test

Choose a spindle speed = 95 rpm
Roller movement angle = 30, 35, 40, 45 degrees

Lathe carriage feed 1 = 1.27 mm/rev	3 specimens
Lathe carriage feed 2 = 0.552 mm/rev	3 specimens
Lathe carriage feed 3 = 0.226 mm/rev	3 specimens

Total specimens = 12 + 9 + 36 + 12

= 69

Repeat the tests with
Increments (distance between two interpolations = 8 encoder pulses (smaller steps)

Total specimens = 138

FIG 6.8: Test programme for the flow-turning process

Chapter 7: Results and discussion

7.1 - Results

From the results shown in tables 1 to 8, graphs 1 to 5 were plotted as seen in appendix 4. The following observations can be made.

The resulting angles from the forming process tend to exhibit a slight difference to the input cone angle from test 1, as shown in graph 1. This trend is almost the same for the 8 and 16 encoder increment (which is the software interpolation step). In flow-turning with a 30 degree former, there is a maximum variation of 3.78 and 3.41 degrees for 8 and 16 encoder increments respectively. With 35, 40 and 45 degrees there is a maximum variation of 1.48 degrees. The reason for the above variation was the inefficient tailstock support set-up discussed in section 6.7 chapter 6. More precise cone angles could be achieved in future with a better set-up design.

Also from the first test, it can be seen from graph 2 that the reduction in wall thickness decreases as the cone angle increases in spinning (35, 40,45) which is about the same for the two interpolation steps. For 8 encoder increments and 30 degree cone angle, the maximum reduction in flow-turning is 25.27% while it is 36.04% with 16 encoder increments. The above trend is to be expected since with a larger increment, the less frequent movement of the tool away from the conical profile results in higher contact pressures between tool and workpiece.

From the second test graph 3, the original hardness of the disc was 43 VPN which remained almost the same with 8 encoder pulses, decreasing to slightly below its original value as spindle speed decreased. With larger increments, that is higher contact pressure between the tool and workpiece, the hardness decreased more steeply as spindle speed decreased.

In flow-turning and spinning processes the use of different lubricants influences the hardness. The effect of the lubricants on the hardness are shown in graph 4, test 1 and 3. Running the process with 8 encoder increments and with lubricant 1, 2 and dry condition, the hardness remains almost the same while with grease (thicker lubricant) the hardness has decreased considerably compared with the unworked material. With 16 encoder increments, the hardness has shown large decreases of similar magnitude both with and without lubricants. This is surprising and difficult to explain since the hardness would be expected to increase with work hardening, or at least remain the same.

In test 4 graph 5, showing the effect of using three different feeds, the hardness value almost retained the original value with 8 encoder pulses, while it decreased by similar amount for all feed rates with 16 as seen from the three curves.

The hardness machine was checked for possible errors, and the tests were repeated, but both checks confirmed that the original results were valid.

Owing to the uneveness of the surface and the curvature of the disc, the hardness test was not easy to set up, adjust and measure. A total of four readings was taken per sample and then these values were averaged. If it were possible to straighten the sample to obtain an easy method of testing this would introduce more stresses into the sample and thus the resulting hardness would not be representative of the after test hardness.

The surface smoothness obtained was not anticipated. From observation it was clear that in the real tests the tool missed an average of 2 in 5 interpolation points, which produced an uneven surface. In contrast, in the contour testing carried out prior to the real testing the plotted curves were acceptable and fairly accurate, see section 6.7 chapter 6. The uneveness of the surface was quite

evident in the flow-turning samples having 30 degrees and to a lesser extent with the spun samples having 35,40 and 45.

After writing the parabola software the contour was plotted as seen in fig 6.7. The parabolic profile starts horizontal(right hand side of the plot) and it appears that because tool movements are small there is no significant overshoot but as the profile becomes steeper, the larger tool movements seem to be accompained be overshoot with the result that the steps become more uneven than expected. The contour was subsequently tested with some samples. A full test programme was not conducted (as with the conical contour) as it was thought that the tests should await implementation of the major hardware and software changes mentioned below. Two samples of A were formed successfully, but when BS 1470 was used it was a bit soft and the samples were torn during the forming process.

7.2 - Implementation of the control system

The research was aimed at making a suitable flow-turning process controller, which would include both hardware and software.

A top-down design methodology was adopted in designing the controller with the intention of clarifying the requirements in terms of hardware and software at an early stage. A schematic layout was drafted for the electronic circuit control followed by the individual board details. Individual circuits were built while other necessary complementary hardware was designed and manufactured.

After estimating the hydraulic requirements, which included the force magnitude, calculations were performed for the required capacity of the directional proportional control valve, the cylinder, the pump, the motor, the relief valve and appropriate oil tank size. The circuit was then fabricated accordingly.

The modifications to the SDK-85 microcomputer board were carried out only after the estimated program memory size was determined in order to establish the number of PROMs to be installed on the board.

In parallel with this, the software design was thought of in terms of blocks doing specific tasks. These tasks were then divided into subtasks to reduce the complexity involved until it was not possible to divide any further.

The software modules were reduced to a minimum in order to give a simple efficient design. The first six modules were written for the conical contour and then another two were added for the parabolic contour.

There were some problems associated with this work, which involved both the software and the hardware. The hardware problems included the following:

The counter board was found to be faulty on testing and gave inconsistent readings. It was decided to have it replaced with a second counter board which proved to operate satisfactorily.

There was an uncertainty about the transducer performance when tested with the software debugger ICE-85. This difficulty was dealt with by inserting a d.c. voltage follower board in the circuit.

The hydraulic power pack was wrongly placed below the cylinder level causing air to be trapped inside. Continuous circulation of oil has minimized this effect to an acceptable level.

The parabolic former was not in the original scheme and had to be made for the final testing towards the end of the research. It took more than a month to make on the CNC lathe as a program had to be written to produce the parabolic profile.

Another difficulty faced at a later stage was a faulty EPROM chip. It was not possible to establish the sources of error and finally another SDK-85 board was used. As far as the software was concerned

the first problem encountered was the counter latching. To avoid inputting readings to the SDK-85 whilst the counter value was changing, a piece of software was incorporated in the 'Shaft Encoder' routine to sample the counter value 10 consecutive times and return a reading only when the ten samples were all the same.

Another problem observed during program development was involved in the tool movement technique. Implementation of a contour based on absolute rather than relative position movement has significantly improved the contour.

An additional problem was that the 8085 CPU integer arithmetic caused a restriction on the calculations. The PL/M 80 compiler could only handle 8 and 16 bit unsigned integer values, thus any real numbers involved as coefficients in the contour equations had to be represented as ratios of positive 16 bit integers with some loss of accuracy. Also, if overflow occurred during the addition or multiplication of 16 bit integers the result would be represented modulo 65535, i.e it would be in error by 65535. To avoid overflow, the contour equations were written such that, in general, a multiplication was followed by a division to prevent the intermediate result from approaching 65535 (see 'Cone Interpolation' procedure). This writing of the equations also involved changing the values of the rational numbers representing the real coefficients to reduce the magnitude of numerator and denominator. This eliminated the risk of overflow at the cost of an acceptable loss of accuracy. Finally, during division computations, the value of the remainder was lost. The equations were written in such a way that resulting position was rounded to the nearest integer. The above procedure was practicable for the linear equations involved in the cone contour, but could not be applied to the parabolic interpolations which involved the computation of square roots. The points on the parabolic

contour to which the tool is required to move were therefore calculated and stored in the program beforehand in the form of a look-up table.

7.3 - Future improvements, modifications and suggestions

A better control over the flow-turning process could be achieved if the following suggestions, modifications and improvements were implemented:

For the 8085 processor, it is possible to add more accuracy to the calculations involved by incorporating a floating point software package. However, this would make the response time slower which is undesirable in this control application, so this idea had to be abandoned. A more advanced processor such as the 8086 with 8087 co-processor will have a hardware floating point which is faster than the software floating point. At the commencement of the project, however, the 8085 was chosen for reasons given in chapter 2 section 2.4. The PL/M 80 programming language used has a simple algorithm, easy to use, quicker to learn and efficient which made it suitable for such an application.

In further research using floating point arithmetic, it would be possible to obtain different parabola resolutions (instead of the constant 16 encoder pulses) in addition to having different parabola contours (as the parabola constant could be varied) since the points could be directly calculated instead of being derived by use of a look-up table. This would allow any parabolic profile to be generated and, depending on the other controls used, an improved contour could be obtained.

As the roller tool moved along the former and the workpiece started to take the required contour, there was no direct way to verify whether the CPU was acting fast enough to perform the various manipulations.

This possibility may be reduced by using a faster processor than the 8085 processor family. Moving up to the 16 bit 8086 processor would be advantageous in future work, using PL/M 86 programming language which is similar to PL/M 80.

In a future implementation with floating point arithmetic, the 'Cone Interpolation' and 'Parabola Interpolation' procedures would be modified to calculate the tool position directly without the use of data stored in look-up tables.

In addition to the above mentioned, there was no fail-safe procedure. The absence of this precaution would cause problems if a power failure occurred during the flow-turning process. A standby power supply could be used.

An improvement added at a later stage of the work was the addition of an emergency stop button which would switch off the lathe after retracting the tool to relieve contact pressure if the tool deviated from the prescribed path.

Conical formers with different angles could be made for further tests since they are relatively easy to produce on an ordinary lathe, unlike the parabolic former which needed a CNC lathe. (Note that, with floating point arithmetic and direct calculation as mentioned above, the cone contouring algorithm could generate cones of any angle, with no requirement for data calculated in advance).

One possibility for further work would be the use of different materials for flow-turning. This was not considered in this research as the main concern was to get the controller working (which consumed most of the time). In addition, the effect of variation in workpiece size could be investigated by performing tests over a range of material thickness and diameters. Using larger workpiece would enable testing the component for ultimate tensile strength after the process, which was not possible with the present workpiece size.

Another suggestion would be to employ a variable speed lathe to establish the highest spindle speed compatible with the software limitations since operating at higher speed would be advantageous with respect to surface finish. Once the speed is established, it could be used subsequently in the flow-turning process for improved results.

The hydraulics utilized performed satisfactorily and gave enough force to form the commerically pure aluminium BS 1470/SIC. Yet, for harder metals such as steel, harder rollers and formers would have to be made as mentioned in chapter 3 section 3.3.

A more accurate control over the tool movement could be achieved with the use of a better directional proportional control valve than the present NG6 valve which determines the oil flow rate to the cylinder which in turn determines the speed. Greater control over velocity in the lower tool speed range would facilitate smoother incremental changes in tool position. Alternatively, if a means of introducing a variable damping to the tool could be incorporated in then probably the overshoot could be considerably reduced.

Drawing the contour prior to the tests could be done more accurately by using electronic or electrical means instead of the present method which cannot detect small movements such as 0.1 mm. The plotted contour could be checked against the one obtained from real testing and thus a close check on the effect of overshoot or tool contact force could be made, see section 6.7 chapter 6.

The encoder increment parameter, which determines the conical step could be improved by reducing the present minimum step of 8 equivalent to 0.5 mm to a minimum of 2 equivalent of 0.2 mm if the new controls were implemented as mentioned above.

Conclusion

A hardware/software design was produced for a microcomputer based flow-turning process controller which after development resulted in a fully functional flow-turning system.

The flow-turning roller tool was made to follow specific patterns of preprogrammed movements namely conical and parabolic contours. Experimentation and testing showed the contour to be successfully reproduced on the product.

The workpiece was also successfully formed to cone angles of 35, 40 and 45 degrees without a supporting former. In addition, flow-turning of a 30 degree cone angle was also achieved with a former.

Flow-turning of the 30 degree angle cones resulted in reductions of wall thickness of 25% - 31%.

The use of a higher spindle speed during forming would have been beneficial to give an improved surface finish. However, the present processor had limitations in terms of response time, therefore in order to achieve an improved quality of surface, a processor with faster response should be used.

REFERENCES

CHAPTER 1

1- SHEET METAL INDUSTRIES

Mar. 1975, Vol. 52, No.3, pg. 140-145

2- C.F.NOBLE & K.S.LEE

"A study in flow-turning"

Proc. of the int. conf. on Rotary Metal Working

Processes (1st) 1979

3- Design & Components in Engineering

Mar. 11 1965, pg. 6-13

4- SHEET METAL INDUSTRIES

May 1977, pg. 485-492

5- Modern Machine Shop

Mar. 11 1965, pg. 6-13

6- B.H.AMSTEAD & M.L.BEGEMAN

"Manufacturing Processes"

pg. 302-305 & pg. 383-401, Sixth Edition , 1969

7- R.A.C. SLATER & A.JOORABCHIAN

"An experimental study of the spin-forging of sheet metal cones

using a mandrel of constant cone angle"

U. of Birmingham, 1976, pg. 531-537

8- SHEET METAL INDUSTRIES

Apr. 1977, pg. 382-389

9- R.A.PAULTON & B.N.COLDING

"Two industrial processes for plastic deformation of metals"

Inst. of Mech, Engineers, Conf. on Eng'g

Manufacturing Technology, paper no.57, London.

Mar. 1958 pg. 54-62

10- SHEET METAL INDUSTRIES

Feb. 1970, Vol.47, No.2, pg. 131-136 & 144

11- SHEET METAL INDUSTRIES

July 1981, pg. 505-511

12- SHEET METAL INDUSTRIES

Feb. 1975, Vol. 52, No.2, pg. 72-75

13- SHEET METAL INDUSTRIES

Dec. 1975, pg. 749

CHAPTER 2

14- G.L. Simons

"The uses of microprocessors"

pg. preface, 22, 97

H. Charlesworth & Co. Ltd, Huddersfield, 1980

15- ICE-85 Instruction Manual

Intel corp. 1978

16- PL/M 80 programming manual

Intel corp. 1978

17- PL/M 80 compiler operators manual

Intel corp. 1977

18- Daniel D. McCRACKEN

"A guide to PL/M 80 programming for microcomputer applications"

Addison-Wesley, 1978

19- MCS-80/85TM family user's manual

Intel corp. 1979

CHAPTER 3

20- SHEET METAL INDUSTRIES

"Spin forming using Meehanite mandrels"

Nov. 1974, pg. 702-704

APPENDICES

Appendix I : Software test programs (8 sheets)

Appendix II : Main program modules (80 sheets)

Appendix III : A Basic program to find the integer ratios (2 sheets)

Appendix IV : Results (13 sheets)

```
ISIS-II PL/M-80 V3.1 COMPILATION OF MODULE TEST1
OBJECT MODULE PLACED IN :F2:prog1.OBJ
COMPILER INVOKED BY:  plm80 :F2:prog1.plm DEBUG


                $WORKFILES(:F2:,:F2:)
                $PAGEWIDTH(80)
                $PAGELENGTH(55)

  1             TEST$1:
                DO;
                   /* FILE$NAME  'PROG1'  */

                /************************************************************/
                /*                                                          */
                /******          TEST$1  DECLARATIONS              ******/
                /*                                                          */
                /************************************************************/

  2   1         DECLARE READ$PULSES ADDRESS;
  3   1         DECLARE DUMMY ADDRESS;
  4   1         DECLARE UPDAD ADDRESS DATA(0363H);
  5   1         DECLARE FOREVER LITERALLY 'WHILE 1';
  6   1         DECLARE DATA$DIRECTION$BYTE$2 LITERALLY '0000$0010B';
  7   1         DECLARE COMMAND$STATUS$REG$2  LITERALLY '28H';
  8   1         DECLARE PORT$2AH LITERALLY '2AH';

                /************************************************************/
                /*                                                          */
                /******         TEST$1  PROCEDURES               ******/
                /*                                                          */
                /************************************************************/

                /************************************************************/
                /* FUNCTION: SHAFT$ENCODER (ENCODER 8-BIT INPUT)            */
                /* PARAMETER INPUT: READ$PULSES                             */
                /* OUTPUTS: 16-BIT VALUE IN (.READ$PULSES)                  */
                /* PROCEDURE: PUBLIC                                        */
                /* CALLS: NOTHING                                           */
                /* DESCRIPTION: THE VALUE OF THE COUNT WILL BE STORED       */
                /*         IN TWO BYTES(16-BIT), THE HIGH AND THE LOW.      */
                /*     PORT$29H WILL BE READ, THE COUNT WILL BE STORED      */
                /*    IN THE LOW BYTE WHICH IS UPDATED CONTINUOUSLY.        */
                /*  WHEN THE VALUE EXCEEDS 255 THEN 1 WILL BE ADDED TO      */
                /* THE HIGH BYTE. PULSE.HIGH AND PULSE.LOW ARE STORED       */
                /*  IN MEMORY WHERE THEY CAN BE TESTED.                     */
                /************************************************************/
```

```
                $EJECT
 9   1          SHAFT$ENCODER:
                PROCEDURE ADDRESS PUBLIC;
10   2              DECLARE PULSE STRUCTURE(LOW BYTE,HIGH BYTE) AT (.READ$PULSES);
11   2              DECLARE COUNTER BYTE;
12   2              DECLARE PORT$29H LITERALLY '29H';

13   2              COUNTER=INPUT(PORT$29H); /* READ CARRIAGE POSITION */
14   2              IF COUNTER<PULSE.LOW  THEN
15   2                 PULSE.HIGH=PULSE.HIGH+1;
16   2              PULSE.LOW=COUNTER;
17   2              RETURN READ$PULSES;
18   2          END SHAFT$ENCODER;

                /************************************************************/
                /*                                                          */
                /******          PORTS   INITIALISATION             ******/
                /*                                                          */
                /************************************************************/

                /*  EXPANSION RAM */

19   1              OUTPUT(COMMAND$STATUS$REG$2)=DATA$DIRECTION$BYTE$2;

                /* PORT A (29H) INPUT (SHAFT ENCODER) */

                /************************************************************/
                /*                                                          */
                /*****          COUNTER   INITIALISATION            *****/
                /*                                                          */
                /************************************************************/

20   1              OUTPUT(PORT$2AH)=00; /* INITIALISE RESET LINE TO FALSE */
21   1              OUTPUT(PORT$2AH)=01; /*  (0V). GENERATE COUNTER RESET */
22   1              OUTPUT(PORT$2AH)=00;

                /************************************************************/
                /*                                                          */
                /******          MAIN   PROGRAM                     ******/
                /*                                                          */
                /************************************************************/
```

```
            $EJECT
  23   1    READ$PULSES=0; /* INITIALISE THE 16-BIT VARIABLE TO 0 */
  24   1    DO FOREVER;
  25   2        CALL UPDAD(DUMMY,SHAFT$ENCODER);
  26   2    END; /* END OF DO FOREVER */

  27   1    END TEST$1;



MODULE INFORMATION:

     CODE AREA SIZE     = 004BH     75D
     VARIABLE AREA SIZE = 0005H      5D
     MAXIMUM STACK SIZE = 0002H      2D
     90 LINES READ
     0 PROGRAM ERROR(S)

END OF PL/M-80 COMPILATION
```

```
ISIS-II PL/M-80 V3.1 COMPILATION OF MODULE TEST2
OBJECT MODULE PLACED IN :F2:prog2.OBJ
COMPILER INVOKED BY:  plm80 :F2:prog2.plm DEBUG


                $WORKFILES(:F2:,:F2:)
                $PAGEWIDTH(80)
                $PAGELENGTH(55)

      1         TEST$2:
                DO;
                   /* FILE$NAME   'PROG2'  */

                /************************************************************/
                /*                                                          */
                /******                TEST$2   DECLARATIONS           ******/
                /*                                                          */
                /************************************************************/

      2   1     DECLARE DATA$DIRECTION$BYTE$1 LITERALLY '0000$0010B';
      3   1     DECLARE DATA$DIRECTION$BYTE$2 LITERALLY '0000$0010B';
      4   1     DECLARE COMMAND$STATUS$REG$1  LITERALLY '20H';
      5   1     DECLARE COMMAND$STATUS$REG$2  LITERALLY '28H';
      6   1     DECLARE DUMMY ADDRESS;
      7   1     DECLARE UPDAD ADDRESS DATA(0363H);
      8   1     DECLARE FOREVER LITERALLY 'WHILE 1';
      9   1     DECLARE PULSE$HIGH LITERALLY '0000$0010B';
     10   1     DECLARE PULSE$LOW  LITERALLY '0000$0000B';
     11   1     DECLARE PORT$2AH   LITERALLY '2AH';
     12   1     DECLARE COUNT BYTE;

                /************************************************************/
                /*                                                          */
                /******                TEST$2   PROCEDURES             ******/
                /*                                                          */
                /************************************************************/

                /************************************************************/
                /* FUNCTION: ADC$INPUT (12-BIT ADC)                         */
                /* PARAMETERS INPUT: ANALOGE SIGNAL                         */
                /* OUTPUTS: 12-BIT DIGITAL VALUE                            */
                /* PROCEDURE: PUBLIC                                        */
                /* CALLS: NOTHING                                           */
                /* DESCRIPTION: TO INITIALISE CONVERSION A HIGH PULSE IS    */
                /*            GIVEN FOLLOWED BY A LOW PULSE, THIS IS DONE BY */
                /*         BIT$1 PORT$2AH.A CERTAIN TIME HAS TO ELAPSE      */
                /*       BEFORE READING THE PORTS                           */
                /*                                21H (8-BIT) LSB           */
                /*                                23H (9-12BIT) MSB         */
                /************************************************************/
```

```
                $EJECT
 13   1         ADC$INPUT:
                PROCEDURE ADDRESS PUBLIC;
 14   2             DECLARE PORT$21H   LITERALLY '21H';
 15   2             DECLARE PORT$23H   LITERALLY '23H';
 16   2             DECLARE ADC$IN$WORD ADDRESS;

 17   2             OUTPUT(PORT$2AH)=PULSE$LOW;
 18   2             OUTPUT(PORT$2AH)=PULSE$HIGH;
 19   2             CALL TIME(1); /* A DELAY OF 100 MICROSECONDS UNTIL */
                                  /*     CONVERSION IS COMPLETED      */

                    /* READ PORT 23H, MASK OFF 4 MSB'S, DOUBLE IT(I.E */
                    /* ADD 8 ZEROS TO THE LEFT) AND THEN ROTATE LEFT  */
                    /*  8 DIGITS.                                     */

 20   2             ADC$IN$WORD=SHL(DOUBLE(INPUT(PORT$23H) AND 0000$1111B),8);

                    /* ADD THIS VALUE TO THE INPUT OF PORT$21H AND THEN */
                    /*  ROTATE RIGHT 2 DIGITS (TO RID OFF 2 LSB'S).     */

 21   2             ADC$IN$WORD=SHR((ADC$IN$WORD+INPUT(PORT$21H)),2);

 22   2             RETURN ADC$IN$WORD; /* RETURN 10-BIT VALUE */
 23   2         END ADC$INPUT;

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

                /*****************************************************************/
                /*                                                               */
                /******              PORTS   INITIALISATION             ******/
                /*                                                               */
                /*****************************************************************/

                /*   BASIC   RAM   */

 24   1         OUTPUT(COMMAND$STATUS$REG$1)=DATA$DIRECTION$BYTE$1;

                /*  PORT 21H  INPUT  ADC (8 LSB)  */
                /*  PORT 23H  INPUT  ADC (4 MSB)  */

                /*   EXPANSION   RAM    */

 25   1         OUTPUT(COMMAND$STATUS$REG$2)=DATA$DIRECTION$BYTE$2;

                /*  PORT 2AH  OUTPUT (START ADC COMMAND) */
```

```
            $EJECT
            /*************************************************************/
            /*                                                           */
            /******             ADC  INITIALISATION             ******/
            /*                                                           */
            /*************************************************************/

 26   1     OUTPUT(PORT$2AH)=PULSE$HIGH; /* START COMMAND TO FALSE */

            /*************************************************************/
            /*                                                           */
            /******             MAIN   PROGRAM                  ******/
            /*                                                           */
            /*************************************************************/

 27   1     DO FOREVER;

 28   2        CALL UPDAD(DUMMY,ADC$INPUT);

               /* A DELAY OF 1 SECOND */

 29   2        COUNT=1;
 30   2        DO WHILE COUNT<=25;
 31   3           CALL TIME(200);
 32   3           COUNT=COUNT+1;
 33   3        END; /* DO WHILE */

 34   2     END;  /* END OF DO FOREVER */

 35   1     END TEST$2;



MODULE INFORMATION:

    CODE AREA SIZE     = 006AH    106D
    VARIABLE AREA SIZE = 0005H      5D
    MAXIMUM STACK SIZE = 0004H      4D
    122 LINES READ
    0 PROGRAM ERROR(S)

END OF PL/M-80 COMPILATION
```

```
ISIS-II PL/M-80 V3.1 COMPILATION OF MODULE TEST3
OBJECT MODULE PLACED IN :F2:prog3.OBJ
COMPILER INVOKED BY:  plm80 :F2:prog3.plm DEBUG


                $WORKFILES(:F2:,:F2:)
                $PAGEWIDTH(80)
                $PAGELENGTH(55)

  1             TEST$3:
                DO;
                   /* FILE$NAME    'PROG3'  */

                /************************************************************/
                /*                                                          */
                /*******              TEST$3    DECLARATIONS          ********/
                /*                                                          */
                /************************************************************/

  2   1         DECLARE DATA$DIRECTION$BYTE$1 LITERALLY '0000$0010B';
  3   1         DECLARE COMMAND$STATUS$REG$1  LITERALLY '20H';
  4   1         DECLARE DUMMY ADDRESS;
  5   1         DECLARE FOREVER  LITERALLY 'WHILE 1';

                /************************************************************/
                /*                                                          */
                /*******              TEST$3    PROCEDURES            *******/
                /*                                                          */
                /************************************************************/

  6   1         VALUE:
                PROCEDURE PUBLIC;
  7   2            DECLARE I   BYTE;
  8   2            DECLARE PORT$22H LITERALLY '22H';

  9   2            DO I=0 TO 255;
 10   3               OUTPUT(PORT$22H)=I;
 11   3            END;

 12   2         END VALUE;

                /************************************************************/
                /*                                                          */
                /*******           PORTS   INITIALISATION           *******/
                /*                                                          */
                /************************************************************/
```

```
            $EJECT
            /* BASIC  RAM  */

 13   1     OUTPUT(COMMAND$STATUS$REG$1)=DATA$DIRECTION$BYTE$1;

            /* PORT 22H  OUTPUT (8 BIT) DAC   */

            /************************************************************/
            /*                                                          */
            /******           MAIN   PROGRAM                    ******/
            /*                                                          */
            /************************************************************/

 14   1     DO FOREVER;
 15   2        CALL VALUE;
 16   2     END;  /* END FOREVER */


 17   1     END TEST$3;



MODULE INFORMATION:

     CODE AREA SIZE     = 002AH     42D
     VARIABLE AREA SIZE = 0003H      3D
     MAXIMUM STACK SIZE = 0002H      2D
     60 LINES READ
     0 PROGRAM ERROR(S)

END OF PL/M-80 COMPILATION
```

```
ISIS-II PL/M-80 V3.1 COMPILATION OF MODULE FLOWTURNINGMAIN
OBJECT MODULE PLACED IN :F2:flow.OBJ
COMPILER INVOKED BY:  plm80 :F2:flow.plm DEBUG


                $WORKFILES(:F2:,:F2:)
                $PAGEWIDTH(80)
                $PAGELENGTH(55)

  1             FLOW$TURNING$MAIN:
                DO;
                    /* MAIN MODULE */

                    /* FILE$NAME    'FLOW' */

                /**********************************************************/
                /*                                                        */
                /******    EXTERNAL  VARIABLES  DECLARATIONS    ******/
                /*                                                        */
                /**********************************************************/

                    /* NONE */

                /**********************************************************/
                /*                                                        */
                /******    PUBLIC  VARIABLES  DECLARATIONS      ******/
                /*                                                        */
                /**********************************************************/

                    /* NONE */

                /**********************************************************/
                /*                                                        */
                /******    LOCAL  VARIABLES  DECLARATIONS       ******/
                /*                                                        */
                /**********************************************************/

  2   1             DECLARE FOREVER LITERALLY 'WHILE 1';
  3   1             DECLARE SELECT$BYTE BYTE;

                /**********************************************************/
                /*                                                        */
                /******    EXTERNAL  PROCEDURES  DECLARATIONS   ******/
                /*                                                        */
                /**********************************************************/

  4   1             INITIALISATION:PROCEDURE EXTERNAL;
  5   2             END INITIALISATION;

  6   1             MACHINE$SETUP:PROCEDURE EXTERNAL;
  7   2             END MACHINE$SETUP;
```

```
 8  1          SHAPE$SELECT:PROCEDURE BYTE EXTERNAL;
 9  2          END SHAPE$SELECT;

10  1          CONE$GENERATION:PROCEDURE EXTERNAL;
11  2          END CONE$GENERATION;

12  1          MACHINE$CONE$CONTROL:PROCEDURE EXTERNAL;
13  2          END MACHINE$CONE$CONTROL;

14  1          PARABOLA$GENERATION:PROCEDURE EXTERNAL;
15  2          END PARABOLA$GENERATION;

16  1          MACHINE$PARABOLA$CONTROL:PROCEDURE EXTERNAL;
17  2          END MACHINE$PARABOLA$CONTROL;

                /*************************************************/
                /*                                               */
                /******    LOCAL  PROCEDURES  DECLARATIONS    ******/
                /*                                               */
                /*************************************************/

                   /* NONE */

                /*************************************************/
                /*                                               */
                /******    PUBLIC  PROCEDURES  DECLARATIONS   ******/
                /*                                               */
                /*************************************************/

                   /* NONE */

                /*************************************************/
                /*                                               */
                /******             MAIN  PROGRAM             ******/
                /*                                               */
                /*************************************************/

18  1          CALL    INITIALISATION;
19  1          CALL    MACHINE$SETUP;

20  1          DO FOREVER;

21  2          SELECT$BYTE=SHAPE$SELECT;  /* SELECT A CONTOUR */
22  2          IF SELECT$BYTE='A'  THEN   /* IF CONE IS SELECTED THEN */
23  2          DO;
24  3             CALL   CONE$GENERATION;  /* ENTER CONE PARAMETERS */
25  3             CALL   MACHINE$CONE$CONTROL; /* FLOW-TURNING PROCESS */
26  3          END;

               ELSE     /* OTHERWISE SELECT A PARABOLA */
27  2          DO;
```

```
 28  3            CALL   PARABOLA$GENERATION; /* ENTER PARABOLA LENGTH */
 29  3            CALL   MACHINE$PARABOLA$CONTROL; /* FLOW-TURNING PROCESS *
       - /
 30  3         END;

 31  2         END; /* DO FOREVER */

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

 32  1      END FLOW$TURNING$MAIN;
```

MODULE INFORMATION:

```
    CODE AREA SIZE     = 002BH     43D
    VARIABLE AREA SIZE = 0001H      1D
    MAXIMUM STACK SIZE = 0002H      2D
    107 LINES READ
    0 PROGRAM ERROR(S)
```

END OF PL/M-80 COMPILATION

```
ISIS-II PL/M-80 V3.1 COMPILATION OF MODULE INITIALISATIONMODULE
OBJECT MODULE PLACED IN :F2:initia.OBJ
COMPILER INVOKED BY:  plm80 :F2:initia.plm DEBUG


          $WORKFILES(:F2:,:F2:)
          $PAGEWIDTH(80)
          $PAGELENGTH(55)

  1       INITIALISATION$MODULE:
          DO;
             /* FILE$NAME   'INITIA' */

          /*************************************************************/
          /*                                                           */
          /******      EXTERNAL  VARIABLES  DECLARATIONS       ******/
          /*                                                           */
          /*************************************************************/

             /* NONE */

          /*************************************************************/
          /*                                                           */
          /******      PUBLIC  VARIABLES  DECLARATIONS         ******/
          /*                                                           */
          /*************************************************************/

  2   1      DECLARE PORT$2AH$OUTPUT BYTE PUBLIC;

          /*************************************************************/
          /*                                                           */
          /******      LOCAL  VARIABLES  DECLARATIONS          ******/
          /*                                                           */
          /*************************************************************/

             /* ASCII CODE CHARACTERS */

  3   1      DECLARE ESC LITERALLY '1BH';
  4   1      DECLARE CLEAR$SCREEN LITERALLY '1AH';
  5   1      DECLARE QUOTE LITERALLY '27H';

             /* ACIA$1 RESET, COMMAND AND CONFIG VALUES */

  6   1      DECLARE ACIA$1$COMMAND$REG BYTE AT (0E000H);
  7   1      DECLARE ACIA$1$RESET LITERALLY '03H';
  8   1      DECLARE ACIA$CONFIG$1 LITERALLY '15H';

             /* RAM'S AND PORTS DESIGNATION */
             /* BASIC RAM */

  9   1      DECLARE DATA$DIRECTION$BYTE$1 LITERALLY '0000$0010B';
```

```
10  1       DECLARE COMMAND$STATUS$REG$1 LITERALLY '20H';

            /* EXPANSION RAM */

11  1       DECLARE DATA$DIRECTION$BYTE$2 LITERALLY '0000$0010B';
12  1       DECLARE COMMAND$STATUS$REG$2 LITERALLY '28H';

            /* PORTS DESIGNATION */

13  1       DECLARE PORT$22H LITERALLY '22H';
14  1       DECLARE PORT$2AH LITERALLY '2AH';
15  1       DECLARE PORT$2AH$OFF$LATHE LITERALLY '1111$1011B';
16  1       DECLARE PORT$2BH$EXTENDED  LITERALLY '0000$0001B';
17  1       DECLARE ADC$IDLE           LITERALLY '0000$0010B';

18  1       DECLARE KBD$DPLY$CONTROL BYTE AT (1900H);
19  1       DECLARE KMODE LITERALLY '0';
20  1       DECLARE KBNIT LITERALLY '0CCH';

21  1       DECLARE CLEAR$DATA$FIELD LITERALLY '0';
22  1       DECLARE CLEAR$ADDRESS$FIELD LITERALLY '0';

        /*************************************************************/
        /*                                                           */
        /******     EXTERNAL  PROCEDURES  DECLARATIONS        ******/
        /*                                                           */
        /*************************************************************/

23  1       CONSOLE$OUT:
            PROCEDURE(CHAR) EXTERNAL;
24  2           DECLARE CHAR BYTE;
25  2       END CONSOLE$OUT;

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

26  1       MESSAGE:
            PROCEDURE(POINTER,LAST$ELEMENT) EXTERNAL;
27  2           DECLARE POINTER ADDRESS;
28  2           DECLARE LAST$ELEMENT ADDRESS;
29  2       END MESSAGE;

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

30  1       GET:
            PROCEDURE(TARGET$CHAR) EXTERNAL;
31  2           DECLARE TARGET$CHAR BYTE;
32  2       END GET;

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

33  1       UPDDT:
            PROCEDURE(PAR$1) EXTERNAL;
```

```
 34  2          DECLARE PAR$1 BYTE;
 35  2        END UPDDT;

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

 36  1        UPDAD:
              PROCEDURE(PAR$2) EXTERNAL;
 37  2          DECLARE PAR$2 ADDRESS;
 38  2        END UPDAD;

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

 39  1        INIT:
              PROCEDURE EXTERNAL;
 40  2        END INIT;

          /*************************************************************/
          /*                                                           */
          /******      LOCAL  PROCEDURES  DECLARATIONS        ******/
          /*                                                           */
          /*************************************************************/

             /* NONE */

          /*************************************************************/
          /*                                                           */
          /******      PUBLIC  PROCEDURES  DECLARATIONS       ******/
          /*                                                           */
          /*************************************************************/

          /*************************************************************/
          /* FUNCTION: INITIALISATION  (INITIALISE COMPONENTS)         */
          /* PARAMETERS INPUT: NONE                                    */
          /* OUTPUTS: TWO PAGES OF TEXT ON THE VDU                     */
          /* PROCEDURE: PUBLIC                                         */
          /* CALLS: CONSOLE$OUT, MESSAGE, GET                          */
          /* DESCRIPTION: A PROCEDURE TO OUTPUT TO THE SCREEN TWO      */
          /*            PAGES OF TEXT ONE AFTER ANOTHER. THE           */
          /*         OPERATOR IS TO PRESS THE SPACE BAR TO PROMOTE     */
          /*     FURTHER PAGE. THE PAGES SHOW AN IDEA ABOUT THE        */
          /*    PROJECT.                                               */
          /*************************************************************/

 41  1    INITIALISATION:
          PROCEDURE PUBLIC;

          /* INITIALISATION FIRST PAGE OF TEXT */

          /*** ESC,'=',  TWO BYTES TO ADDRESS CURSOR ON SCREEN
              '!','.',  TWO BYTES TO INDICATE NUMBER OF ROW AND COLUMN
          RESPECTIVELY, IN THIS CASE  ROW 2 COLUMN 15 SEE TABLE ***/
```

```
          /* 1st page of text */

42   2    DECLARE PAGE$1(*) BYTE DATA
          (ESC,'=','!','.',     /* R2 C15 */
          'MICROPROCESSOR AIDED FLOW-TURNING'
          ,ESC,'=','"','.',     /* R3 C15 */
          '**********************************'
          ,ESC,'=','£',' ',     /* R4 C1  */
          'OBJECT: To control the FLOW-TURNING PROCESS by an SDK-85 microcom
      -    puter.'
          ,ESC,'=','$',' ',     /* R5 C1  */
          '------'
          ,ESC,'=','&',' ',     /* R7 C1  */
          'DESCRIPTION: A process to make an axisymmetric components out of
      -    circular'
          ,ESC,'=',QUOTE,' ',   /* R8 C1  */
          '-----------    discs. A roller is to move and to press the rotati
      -    ng disc'
          ,ESC,'=','(','-',     /* R9 C14 */
          'against the former. The final shape is dependent upon the'
          ,ESC,'=',')','-',     /* R10 C14 */
          'former shape which could be one of the following:'
          ,ESC,'=','+','(',     /* R12 C9 */
          '1-CONICAL contour'
          ,ESC,'=',',',',(',    /* R13 C9 */
          '2-PARABOLIC contour'
          ,ESC,'=','.','(',     /* R15 C9 */
          ' Wall thickness is to be reduced after the process.'
          ,ESC,'=','4',' ',     /* R21 C1 */
          'More!! depress SP to proceed');

          /* 2nd page of text */

43   2    DECLARE PAGE$2(*) BYTE DATA
          (ESC,'=',' ','%',     /* R1 C6 */
          '************************************************************'
          ,ESC,'=','!','+',     /* R2 C12 */
          'M.Phil Project              BY  ALADDIN  H.  MOHAMAD'
          ,ESC,'=','"','%',     /* R3 C6 */
          '************************************************************'
          ,ESC,'=','£','+',     /* R4 C12 */
          'Mechanical and Production Engineering Department'
          ,ESC,'=','%','&',     /* R6 C7 */
          'Supervisors :-'
          ,ESC,'=','&','5',     /* R7 C22 */
          '1-Dr M.SARWAR (Mech. & Production Eng.)'
          ,ESC,'=',QUOTE,'5',   /* R8 C22 */
          '2-Dr M.S.J. HASHMI (Mech. & Production Eng.)'
          ,ESC,'=','(','5',     /* R9 C22 */
          '3-Dr J.R. TRAVIS (Elec. & Electronics Eng.)'
          ,ESC,'=',')','(',     /* R10 C9 */
          'TITLE -  Microprocessor Aided Flow-Turning.'
```

```
          ,ESC,'=','*','$',     /* R11 C5 */
          'DESCRIPTION -'
          ,ESC,'=','+','3',     /* R12 C20 */
          'The program will control roller movement according to a'
          ,ESC,'=',',','6',     /* R13 C23 */
          'prescribed path.'
          ,ESC,'=','-',QUOTE,   /* R14 C8 */
          'Calls -  Initialisation, Machine$Setup, Console$Input$Output,'
          ,ESC,'=','.','0',     /* R15 C17 */
          'Shape$Select, Cone$Generation, Machine$Cone$Control,'
          ,ESC,'=','/','0',     /* R16 C17 */
          'Parabola$Generation, Machine$Parabola$Control.'
          ,ESC,'=','0',QUOTE,   /* R17 C8 */
          'Program Requirements       11K bytes of memory'
          ,ESC,'=','1',QUOTE,   /* R18 C8 */
          'I/O    Requirements       2 (8-BIT) Input Ports'
          ,ESC,'=','2','B',     /* R19 C35 */
          '2 (8-BIT) Output Ports'
          ,ESC,'=','3','B',     /* R20 C35 */
          '2 (6-BIT) Input Ports'
          ,ESC,'=','4',QUOTE,   /* R21 C8 */
          'Programming  Language          PL/M 80'
          ,ESC,'=','5',QUOTE,   /* R22 C8 */
          'Microcomputer  Used            INTEL-85 with 8085 CPU',
          ESC,'=','7','+',      /* R24 C12 */
          'Press SP to proceed please');

          /********************* PORTS  INITIALISATION ********************/

44   2        OUTPUT(COMMAND$STATUS$REG$1) =DATA$DIRECTION$BYTE$1;
              /* BASIC RAM CONFIGURATION (BITS FROM RIGHT TO LEFT)
                 PORT A (21H)  INPUT  (ADC)    1-8 BIT
                 PORT B (22H)  OUTPUT (DAC)    1-8 BIT
                 PORT C (23H)  INPUT  (ADC)   9-12 BIT
              */

45   2        OUTPUT(COMMAND$STATUS$REG$2) =DATA$DIRECTION$BYTE$2;
              /* EXPANSION RAM CONFIGURATION (BITS FROM RIGHT TO LEFT)
                 PORT A (29H)  INPUT  (COUNTER)
                 PORT B (2AH)  OUTPUT (SIGNALS)
                 PORT C (2BH)  INPUT  (SWITCHES)
              */
          /*
          BASIC RAM (8155)
          ----------------
          INPUT PORT 21H:INDICATOR BITS (ADC 8 LEAST SIGNIFICANT BITS)
           BIT$0:
           BIT$1:
           BIT$2:
           BIT$3:            ALL USED
           BIT$4:
           BIT$5:
```

```
 BIT$6:
 BIT$7:

OUTPUT PORT 22H:INDICATOR BITS (DAC)
 BIT$0:
 BIT$1:
 BIT$2:
 BIT$3:            ALL USED
 BIT$4:
 BIT$5:
 BIT$6:
 BIT$7:

INPUT PORT 23H:INDICATOR BITS (ADC 4 MOST SIGNIFICANT BITS)
 BIT$0:  ADC  BIT$8
 BIT$1:  ADC  BIT$9
 BIT$2:  ADC  BIT$10
 BIT$3:  ADC  BIT$11
 BIT$4:  NOT USED
 BIT$5:  NOT USED

EXPANSION RAM (8155)
--------------------
INPUT PORT 29H:INDICATOR BITS (COUNTER)
  BIT$0:
  BIT$1:
  BIT$2:
  BIT$3:            ALL USED
  BIT$4:
  BIT$5:
  BIT$6:
  BIT$7:

OUTPUT PORT 2AH:INDICATOR BITS (SIGNALS)
  BIT$0:   (COUNTER RESET BIT)
  BIT$1:   (START CONVERSION ADC BIT)
  BIT$2:   (ON/OFF LATHE BIT)
  BIT$3:   (ON/OFF OIL PUMP BIT)
  BIT$4:   NOT USED
  BIT$5:   NOT USED
  BIT$6:   NOT USED
  BIT$7:   NOT USED

INPUT PORT 2BH:INDICATOR BITS
  BIT$0: CYLINDER$ADVANCE$SWITCH
  BIT$1: NOT USED
  BIT$2: CARRIAGE$LIMIT$SWITCH
  BIT$3: NOT USED
  BIT$4: NOT USED
  BIT$5: NOT USED
*/
/****************** ACIA  INITIALISATION ********************/
```

```
            /* ACIA$1 */

 46   2       ACIA$1$COMMAND$REG = ACIA$1$RESET;  /* RESETS ACIA$1 */
 47   2       ACIA$1$COMMAND$REG = ACIA$CONFIG$1; /* ACIA$1 CONFIGURATION $16
          -   BAUD RATE */

            /************  COMPONENTS INITIALISATION  ****************/

           /* INTERRUPT ENABLE  RST 6.5 AND RST 7.5 */

 48   2         CALL  INIT;

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

           /* INITIALISE THE OUTPUT BYTE OF PORT 2AH TO 1 */
           /*  I.E COUNTER INHIBITED, LATHE OFF AND       */
           /*       ADC CONVERT LOW                       */

 49   2       PORT$2AH$OUTPUT=1;
 50   2       OUTPUT(PORT$2AH)=PORT$2AH$OUTPUT;

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

           /** ADC (CONVERTOR COMMAND) **/
           /* INITIALISE THE START COMMAND TO TRUE  (5V) */

 51   2         PORT$2AH$OUTPUT=PORT$2AH$OUTPUT OR ADC$IDLE;
 52   2         OUTPUT(PORT$2AH)=PORT$2AH$OUTPUT;

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

           /** DAC INITIALISE **/
           /* INITIALISE DAC TO 0V i.e 128 */

 53   2         OUTPUT(PORT$22H)=128;

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

           /**  INITIALISE SDK-85 KEYBOARD DISPLAY  **/

 54   2         KBD$DPLY$CONTROL=KMODE;
 55   2         KBD$DPLY$CONTROL=KBNIT;

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

              /**  CLEAR SDK-85 DISPLAY  **/

 56   2          CALL UPDDT(CLEAR$DATA$FIELD);
 57   2          CALL UPDAD(CLEAR$ADDRESS$FIELD);

           /*************************************************************/
```

```
 58  2          CALL   CONSOLE$OUT(CLEAR$SCREEN);

                /* FIRST PAGE OF TEXT */

 59  2          CALL   MESSAGE(.PAGE$1,LAST(PAGE$1)); /* PAGE 1 */
 60  2          CALL   GET(' ');                      /* GET SP */
 61  2          CALL   CONSOLE$OUT(CLEAR$SCREEN); /* CLEAR SCREEN */

                /* SECOND PAGE OF TEXT */

 62  2          CALL   MESSAGE(.PAGE$2,LAST(PAGE$2)); /* PAGE 2 */
 63  2          CALL   GET(' ');                      /* GET SP */
 64  2          CALL   CONSOLE$OUT(CLEAR$SCREEN); /* CLEAR SCREEN */

 65  2          END INITIALISATION;

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

 66  1      END INITIALISATION$MODULE;



MODULE INFORMATION:

    CODE AREA SIZE     = 0668H   1640D
    VARIABLE AREA SIZE = 0001H      1D
    MAXIMUM STACK SIZE = 0002H      2D
    374 LINES READ
    0 PROGRAM ERROR(S)

END OF PL/M-80 COMPILATION
```

```
ISIS-II PL/M-80 V3.1 COMPILATION OF MODULE CONSOLEINPUTOUTPUTMODULE
OBJECT MODULE PLACED IN :F2:consol.OBJ
COMPILER INVOKED BY:  plm80 :F2:consol.plm DEBUG


                $WORKFILES(:F2:,:F2:)
                $PAGEWIDTH(80)
                $PAGELENGTH(55)

  1             CONSOLE$INPUT$OUTPUT$MODULE:
                DO;
                   /* FILE$NAME  'CONSOL' */

                /**************************************************/
                /*                                                */
                /******     EXTERNAL  VARIABLES  DECLARATIONS    ******/
                /*                                                */
                /**************************************************/

                   /* NONE */

                /**************************************************/
                /*                                                */
                /******     PUBLIC  VARIABLES  DECLARATIONS      ******/
                /*                                                */
                /**************************************************/

                   /* NONE */

                /**************************************************/
                /*                                                */
                /******     LOCAL  VARIABLES  DECLARATIONS       ******/
                /*                                                */
                /**************************************************/

                   /* ASCII CODE CHARACTERS */

  2   1            DECLARE ESC LITERALLY '1BH';
  3   1            DECLARE BELL LITERALLY '07H';
  4   1            DECLARE SPACE LITERALLY '20H';
  5   1            DECLARE BACK$SPACE LITERALLY '08H';
  6   1            DECLARE CARRIAGE$RETURN LITERALLY '0DH';
  7   1            DECLARE DELETE LITERALLY '7FH';

                   /* ACIA$1 FLAGS, COMMANDS AND DATA VALUES */

  8   1            DECLARE RXRDY$1 LITERALLY '01H';
  9   1            DECLARE TXRDY$1 LITERALLY '02H';
 10   1            DECLARE ACIA$1$COMMAND$REG BYTE AT (0E000H);
 11   1            DECLARE ACIA$1$DATA$REG BYTE AT (0E001H);
```

```
                    /* MODULE VARIABLES */

 12   1             DECLARE I ADDRESS;

 13   1             DECLARE MESS$1(*)BYTE DATA
                    (ESC,'j');                  /* START INVERSE */
 14   1             DECLARE MESS$2(*)BYTE DATA
                    (ESC,'k');                  /* END INVERSE */

                  /*************************************************************/
                  /*                                                           */
                  /*******     EXTERNAL  PROCEDURES  DECLARATIONS      *******/
                  /*                                                           */
                  /*************************************************************/

                    /* NONE */

                  /*************************************************************/
                  /*                                                           */
                  /*******     LOCAL  PROCEDURES  DECLARATIONS         *******/
                  /*                                                           */
                  /*************************************************************/

                    /* NONE */

                  /*************************************************************/
                  /*                                                           */
                  /*******     PUBLIC  PROCEDURES  DECLARATIONS        *******/
                  /*                                                           */
                  /*************************************************************/

                  /*************************************************************/
                  /* FUNCTION: CONSOLE$OUT (CONSOLE OUTPUT TO SCREEN)          */
                  /* PARAMETERS INPUT: CHAR(BYTE)                              */
                  /* OUTPUTS: CHAR                                             */
                  /* PROCEDURE: PUBLIC                                         */
                  /* CALLS: NOTHING                                            */
                  /* DESCRIPTION: SENDS A CHARACTER TO SCREEN                  */
                  /*************************************************************/
                                                                        /* */
 15   1           CONSOLE$OUT:                                          /* */
                  PROCEDURE(CHAR) PUBLIC;                               /* */
 16   2              DECLARE CHAR BYTE;                                 /* */
 17   2                DO WHILE NOT ((ACIA$1$COMMAND$REG AND TXRDY$1)=2);/* */
 18   3                END;                                             /* */
 19   2                ACIA$1$DATA$REG=CHAR;                            /* */
 20   2           END CONSOLE$OUT;                                      /* */
                                                                        /* */
                  /*************************************************************/


                  /*************************************************************/
```

```
                /* FUNCTION: MESSAGE (MESSAGE DISPLAYED ON SCREEN)              */
                /* PARAMETERS INPUT: POINTER(ADDRESS), LAST$ELEMENT(ADDRESS)    */
                /* OUTPUTS: MESSAGE CHARACTERS CHAR                             */
                /* PROCEDURE: PUBLIC                                            */
                /* CALLS: CONSOLE$OUT                                           */
                /* DESCRIPTION: OUTPUTS A TEXT MESSAGE ON SCREEN VIA ACIA$1     */
                /*****************************************************************/
                                                                        /*      */
 21   1        MESSAGE:                                                /*      */
                PROCEDURE(POINTER,LAST$ELEMENT)PUBLIC;                  /*      */
 22   2            DECLARE POINTER ADDRESS;                            /*      */
 23   2            DECLARE TEXT$MESSAGE BASED POINTER(2000) BYTE;      /*      */
 24   2            DECLARE LAST$ELEMENT ADDRESS;                       /*      */
 25   2              DO I=0 TO LAST$ELEMENT;                           /*      */
 26   3                    CALL CONSOLE$OUT(TEXT$MESSAGE(I));          /*      */
 27   3              END;                                              /*      */
 28   2        END MESSAGE;                                            /*      */
                                                                        /*      */
                /*****************************************************************/


                /*****************************************************************/
                /* FUNCTION: CONSOLE$IN (OBTAIN A CHAR)                         */
                /* PARAMETERS INPUT: NONE                                       */
                /* OUTPUTS: NONE                                                */
                /* PROCEDURE: TYPED(BYTE), PUBLIC                               */
                /* CALLS: NOTHING                                               */
                /* DESCRIPTION: WAITS UNTIL A CHARACTER IS ENTERED FROM         */
                /*         KEYBOARD THEN ASSIGNS THE VALUE TO A VARAIABLE CHAR  */
                /*****************************************************************/
                                                                        /*      */
 29   1        CONSOLE$IN:                                             /*      */
                PROCEDURE BYTE PUBLIC;                                  /*      */
 30   2            DECLARE CHAR BYTE;                                  /*      */
 31   2              DO WHILE NOT ((ACIA$1$COMMAND$REG AND RXRDY$1)=1);  /*      */
 32   3              END;                                              /*      */
 33   2              CHAR=ACIA$1$DATA$REG AND 7FH ;/* STRIP OFF PARITY BIT    */
 34   2              RETURN CHAR;                                      /*      */
 35   2        END CONSOLE$IN;                                         /*      */
                                                                        /*      */
                /*****************************************************************/


                /*****************************************************************/
                /* FUNCTION: GET (GET THE APPROPRIAT CHARACTER FROM KEYBOARD)   */
                /* PARAMETERS INPUT: THE REQUIRED PARAMETER (' ')               */
                /* OUTPUTS: NONE                                                */
                /* PROCEDURE: PUBLIC                                            */
                /* CALLS: CONSOLE$OUT, CONSOLE$IN                               */
                /* DESCRIPTION: READS A CHAR, IF IT IS THE TARGET ONE THEN      */
                /*            IT RETURNS TO THE CALLING PROGRAM, OTHERWISE      */
                /*          WILL RING A BELL AND THE OPERATOR SHOULD NOW        */
```

```
                /*        PRESS THE TARGET CHAR                              */
                /*************************************************************/
                                                                     /* */
 36   1         GET:                                                /* */
                PROCEDURE(TARGET$CHAR) PUBLIC;                       /* */
 37   2             DECLARE TARGET$CHAR BYTE;                       /* */
 38   2             DECLARE CHAR BYTE;                              /* */
 39   2               CHAR=CONSOLE$IN; /* READ A CHARACTER FROM KEYBOARD   */
 40   2               DO WHILE CHAR<>TARGET$CHAR;                   /* */
 41   3                  CALL CONSOLE$OUT(BELL); /* RING A BELL          */
 42   3                  CHAR=CONSOLE$IN; /* READ A CHARACTER FROM KEYBOARD */
 43   3               END;                                          /* */
 44   2         END GET;                                            /* */
                                                                     /* */
                /*************************************************************/


                /*************************************************************/
                /* FUNCTION: DECIMAL$VALUE$INPUT (ONE OR TWO DECIMAL DIGIT)  */
                /* PARAMETERS INPUT: NONE                                    */
                /* OUTPUTS: NONE                                             */
                /* PROCEDURE: TYPED(BYTE), PUBLIC                            */
                /* CALLS: CONSOLE$OUT, CONSOLE$IN                            */
                /* DESCRIPTION: TO INPUT A NUMBER CONSISTING OF ONE OR TWO   */
                /*            DIGITS FROM THE KEYBOARD FOLLOWED BY CARRIAGE  */
                /*          RETURN. IF DIGIT PRESSED IS (0 TO 9) THEN IT     */
                /*        WOULD BE ACCEPTED, OTHERWISE A BELL WILL RING, THE */
                /*     NUMBER WILL BE DELETED FROM THE SCREEN AND THE CURSOR  */
                /*   WILL RETURN TO THE INITIAL POSITION AND THE OPERATOR     */
                /*  CAN NOW ENTER THE RIGHT DIGITS.                          */
                /*************************************************************/
                                                                     /* */
 45   1         DECIMAL$VALUE$INPUT:                                /* */
                PROCEDURE BYTE PUBLIC;                               /* */
 46   2             DECLARE VALUE BYTE;                             /* */
 47   2             DECLARE NO$DIGITS BYTE;                         /* */
 48   2             DECLARE CHAR BYTE;                              /* */
 49   2             DECLARE DECIMAL$DIGIT BYTE;                     /* */
                                                                     /* */
 50   2             VALUE=0;        /* INITIALISE                       */
 51   2             NO$DIGITS=0;    /* INITIALISE                       */
                                                                     /* */
 52   2             CHAR=CONSOLE$IN; /* READ A CHARACTER FROM KEYBOARD      */
                                                                     /* */
 53   2             DO WHILE CHAR <> CARRIAGE$RETURN;               /* */
                              /*** IF CHAR=DECIMAL DIGIT               ***/
 54   3              IF CHAR>='0' AND CHAR<='9' THEN /* ANY DIGIT 0 TO 9  */
 55   3                DO;                                          /* */
 56   4                   NO$DIGITS=NO$DIGITS+1;                    /* */
 57   4                   IF NO$DIGITS<=2 THEN                      /* */
 58   4                      DO;                                    /* */
 59   5                         CALL  CONSOLE$OUT(CHAR); /* CHAR ON SCREEN */
```

```
 60   5                  DECIMAL$DIGIT=CHAR-'0';                    /* */
 61   5                  VALUE=VALUE$10;                            /* */
 62   5                  VALUE=VALUE+DECIMAL$DIGIT;                 /* */
 63   5               END; /* END IF                                   */
                    ELSE                                            /* */
 64   4               DO;                                            /* */
 65   5                  NO$DIGITS=2;                                /* */
 66   5                  CALL CONSOLE$OUT(BELL); /* RING A BELL      */
 67   5               END; /* END ELSE                                 */
 68   4             END; /*** END IF CHAR=DECIMAL$DIGIT              ***/
                                                                    /* */
                      /*** IF CHAR=DELETE                           ***/
 69   3          ELSE IF CHAR=DELETE THEN                          /* */
 70   3              IF NO$DIGITS>0 THEN                           /* */
 71   3                DO;                                         /* */
 72   4                  VALUE=VALUE-DECIMAL$DIGIT;                /* */
 73   4                  VALUE=VALUE/10;                           /* */
 74   4                  DECIMAL$DIGIT=VALUE; /* OK FOR MAX 2-DIGITS ONLY */
 75   4                  NO$DIGITS=NO$DIGITS-1;                    /* */
 76   4                  CALL CONSOLE$OUT(BACK$SPACE);/* GO ONE SPACE BACK */
 77   4                  CALL CONSOLE$OUT(SPACE); /* DELETE DIGIT    */
 78   4                  CALL CONSOLE$OUT(BACK$SPACE); /* GO BACK AGAIN */
 79   4                END; /* END IF                                 */
                      ELSE       /* NO$DIGITS=0                      */
 80   3                  CALL CONSOLE$OUT(BELL); /* RING A BELL     */
                            /*** END CHAR=DELETE                   ***/
                                                                    /* */
                            /*** CHAR=ANY OTHER CHAR               ***/
                    ELSE                                            /* */

 81   3                CALL CONSOLE$OUT(BELL); /* RING A BELL        */
                            /*** END CHAR=ANY OTHER CHAR           ***/
 82   3                CHAR=CONSOLE$IN; /* READ A CHAR FROM KEYBOARD */
 83   3            END; /* WHILE                                     */
 84   2             RETURN VALUE;                                  /* */
                                                                    /* */
 85   2           END DECIMAL$VALUE$INPUT;                         /* */
                                                                    /* */
              /*****************************************************************/

 86   1       END CONSOLE$INPUT$OUTPUT$MODULE;


MODULE INFORMATION:

     CODE AREA SIZE     = 014AH    330D
     VARIABLE AREA SIZE = 000EH     14D
     MAXIMUM STACK SIZE = 0004H      4D
     243 LINES READ
     0 PROGRAM ERROR(S)
```

```
ISIS-II PL/M-80 V3.1 COMPILATION OF MODULE MACHINESETUPMODULE
OBJECT MODULE PLACED IN :F2:setup.OBJ
COMPILER INVOKED BY:  plm80 :F2:setup.plm DEBUG


            $WORKFILES(:F2:,:F2:)
            $PAGEWIDTH(80)
            $PAGELENGTH(55)

  1         MACHINE$SETUP$MODULE:
            DO;
               /* FILE$NAME  'SETUP' */

            /*****************************************************************/
            /*                                                               */
            /******      EXTERNAL  VARIABLES  DECLARATIONS           ******/
            /*                                                               */
            /*****************************************************************/

  2   1        DECLARE PORT$2AH$OUTPUT BYTE EXTERNAL;

            /*****************************************************************/
            /*                                                               */
            /******      PUBLIC  VARIABLES  DECLARATIONS             ******/
            /*                                                               */
            /*****************************************************************/

               /* NONE */

            /*****************************************************************/
            /*                                                               */
            /******      LOCAL  VARIABLES  DECLARATIONS              ******/
            /*                                                               */
            /*****************************************************************/

               /* ASCII CODE CHARACTERS */

  3   1        DECLARE BELL LITERALLY  '07H';
  4   1        DECLARE BACK$SPACE LITERALLY  '08H';
  5   1        DECLARE LINE$FEED  LITERALLY  '0AH';
  6   1        DECLARE CARRIAGE$RETURN  LITERALLY  '0DH';
  7   1        DECLARE CLEAR$SCREEN  LITERALLY  '1AH';
  8   1        DECLARE ESC  LITERALLY '1BH';
  9   1        DECLARE HOME LITERALLY  '1EH';
 10   1        DECLARE SP  LITERALLY  '20H';
 11   1        DECLARE DELETE  LITERALLY  '7FH';
 12   1        DECLARE QUOTE LITERALLY '27H';     /* SINGLE QUOTE */
 13   1        DECLARE DQUOTE LITERALLY '22H';    /* DOUBLE QUOTE */

 14   1        DECLARE PORT$2BH LITERALLY '2BH';
 15   1        DECLARE PORT$2BH$INPUT BYTE;
```

```
 16  1          DECLARE PORT$2BH$ADVANCED LITERALLY '0000$0001B';

 17  1          DECLARE PORT$2AH$START$PUMP LITERALLY '0000$1000B';
 18  1          DECLARE PORT$2AH LITERALLY '2AH';

                /* MODULE VARIABLES */

 19  1          DECLARE LETTER BYTE;

             /**************************************************************/
             /*                                                            */
             /*******     EXTERNAL  PROCEDURES  DECLARATIONS           *******/
             /*                                                            */
             /**************************************************************/
                                                                       /* */
 20  1          MESSAGE:                                             /* */
                PROCEDURE(POINTER,LAST$ELEMENT) EXTERNAL;            /* */
 21  2             DECLARE POINTER ADDRESS;                          /* */
 22  2             DECLARE LAST$ELEMENT ADDRESS;                     /* */
 23  2          END MESSAGE;                                         /* */
                                                                       /* */
             /**************************************************************/
                                                                       /* */
 24  1          GET:                                                 /* */
                PROCEDURE(TARGET$CHAR) EXTERNAL;                     /* */
 25  2             DECLARE TARGET$CHAR BYTE;                         /* */
 26  2          END GET;                                             /* */
                                                                       /* */
             /**************************************************************/

             /**************************************************************/
             /*                                                            */
             /*******      LOCAL  PROCEDURES  DECLARATIONS           *******/
             /*                                                            */
             /**************************************************************/

                /* NONE */

             /**************************************************************/
             /*                                                            */
             /*******     PUBLIC  PROCEDURES  DECLARATIONS           *******/
             /*                                                            */
             /**************************************************************/

             /**************************************************************/
             /* FUNCTION: SETUP                                            */
             /* PARAMETERS INPUT: NONE                                     */
             /* OUTPUTS: NONE                                              */
             /* PROCEDURE: PUBLIC                                          */
             /* CALLS: MESSAGE, GET                                        */
             /* DESCRIPTION: A PROCEDURE PRODUCING COMMANDS TO OPERATOR TO  */
             /*           SET UP THE EQUIPMENTS AND MAKE THEM READY FOR    */
```

```
                /*       OPERATION. THE COMMAND WILL BE DISPLAYED ONE BY ONE */
                /*     WHILE THE OPERATOR IS EXPECTED TO DO AS TOLD AND THEN  */
                /*   PRESS SPACE KEY AFTER CARRING OUT EACH ONE. ALSO THE TEXT */
                /* WILL BE DISPLAYED FLASHING AND THEN WILL TURN TO NORMAL     */
                /*  VIDEO AFTER PRESSING SPACE.                               */
                /*************************************************************/

 27   1         MACHINE$SETUP:
                PROCEDURE PUBLIC;

                /* MESSAGES TO BE OUTPUTED TO THE VDU SCREEN.    */
                /*  THESE INCLUDE MACHINE SETUP INSTRUCTIONS AND */
                /*   OTHER MESSAGES.                             */

                /* TEXT IN REVERSE VIDEO */

 28   2         DECLARE PAGE$3(*) BYTE DATA
                (ESC,'=',SP,'1',                    /* R1 C18 */
                ESC,'j',        /* START REVERSE */
                'MICROPROCESSOR AIDED FLOW-TURNING',
                ESC,'k',        /* END REVERSE */
                ESC,'=','!','1',                    /* R2 C18 */
                ESC,'j',        /* START REVERSE */
                '*********************************',
                ESC,'k',        /* END REVERSE */
                ESC,'=',DQUOTE,DQUOTE,              /* R3 C3 */
                ESC,'j',        /* START REVERSE */
                'M.Phil project',
                ESC,'k',        /* END REVERSE */
                ESC,'=','£','X',                    /* R4 C57 */
                ESC,'j',        /* START REVERSE */
                'ALADDIN H. MOHAMAD',
                ESC,'k',        /* END REVERSE */
                ESC,'=','$','8',                    /* R5 C25 */
                'MACHINE SET-UP',
                ESC,'=','%','8',                    /* R6 C25 */
                '-------------');
 29   2         DECLARE BLINK(*) BYTE DATA
                (ESC,'^');                      /* BLINKING CHARACTERS */
 30   2         DECLARE NORMAL(*) BYTE DATA
                (ESC,'q');                      /* STOP BLINKING */
 31   2         DECLARE ERASE(*) BYTE DATA
                (ESC,'y');                      /* ERASE TO END OF SCREEN WITH NUL
            -    LS */
 32   2         DECLARE ST$ERA(*) BYTE DATA
                (ESC,'=','$','8');                  /* R5 C25 */
 33   2         DECLARE MESS$1R8C5(*) BYTE DATA
                (ESC,'=',QUOTE,'$');                /* R8 C5 */
 34   2         DECLARE MESS$1(*) BYTE DATA
                ('1-Place workpiece in position and depress  SP.');
 35   2         DECLARE MESS$2R9C5(*) BYTE DATA
                (ESC,'=','(','$');                  /* R9 C5 */
```

```
 36  2     DECLARE MESS$2(*) BYTE DATA
           ('2-Start oil pump and depress  SP.');
 37  2     DECLARE MESS$3R10C5(*) BYTE DATA
           (ESC,'=',')','$');                    /* R10 C5 */
 38  2     DECLARE MESS$3(*) BYTE DATA
           ('3-Energise valve controller V/I then SP.');
 39  2     DECLARE MESS$4R11C5(*) BYTE DATA
           (ESC,'=','*','$');                    /* R11 C5 */
 40  2     DECLARE MESS$4(*) BYTE DATA
           ('4-Switch on ADC, DAC, encoder and SDK-85, then SP.');
 41  2     DECLARE MESS$5R12C5(*) BYTE DATA
           (ESC,'=','+','$');                    /* R12 C5 */
 42  2     DECLARE MESS$5(*) BYTE DATA
           ('5-Check tool is at datum position, if not move to datum then SP.
         - ');
 43  2     DECLARE MESS$6(*) BYTE DATA
           (ESC,'=','4','$',                     /* R21 C5 */
           'Press SP to proceed.');

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

              /*     MAIN  PROGRAM     */

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

           /* A SERIES OF CALLS WHICH DISPLAY THE INSTRUCTIONS ON */
           /*  THE VDU SCREEN.                                    */

 44  2     CALL    MESSAGE(.PAGE$3,LAST(PAGE$3));
 45  2     CALL    MESSAGE(.MESS$1RBC5,LAST(MESS$1RBC5));
 46  2     CALL    MESSAGE(.BLINK,LAST(BLINK));
 47  2     CALL    MESSAGE(.MESS$1,LAST(MESS$1));
 48  2     CALL    GET(' ');
 49  2     CALL    MESSAGE(.MESS$1R8C5,LAST(MESS$1R8C5));
 50  2     CALL    MESSAGE(.NORMAL,LAST(NORMAL));

 51  2     CALL    MESSAGE(.MESS$2R9C5,LAST(MESS$2R9C5));
 52  2     CALL    MESSAGE(.BLINK,LAST(BLINK));
 53  2     CALL    MESSAGE(.MESS$2,LAST(MESS$2));

           /* SWITCH ON THE OIL PUMP */

 54  2     PORT$2AH$OUTPUT=PORT$2AH$OUTPUT OR PORT$2AH$START$PUMP;
 55  2     OUTPUT(PORT$2AH)=PORT$2AH$OUTPUT;

 56  2     CALL    GET(' ');
 57  2     CALL    MESSAGE(.MESS$2R9C5,LAST(MESS$2R9C5));
 58  2     CALL    MESSAGE(.NORMAL,LAST(NORMAL));

 59  2     CALL    MESSAGE(.MESS$3R10C5,LAST(MESS$3R10C5));
 60  2     CALL    MESSAGE(.BLINK,LAST(BLINK));
 61  2     CALL    MESSAGE(.MESS$3,LAST(MESS$3));
```

```
 62  2      CALL    GET(' ');
 63  2      CALL    MESSAGE(.MESS$3R10C5,LAST(MESS$3R10C5));
 64  2      CALL    MESSAGE(.NORMAL,LAST(NORMAL));

 65  2      CALL    MESSAGE(.MESS$4R11C5,LAST(MESS$4R11C5));
 66  2      CALL    MESSAGE(.BLINK,LAST(BLINK));
 67  2      CALL    MESSAGE(.MESS$4,LAST(MESS$4));
 68  2      CALL    GET(' ');
 69  2      CALL    MESSAGE(.MESS$4R11C5,LAST(MESS$4R11C5));
 70  2      CALL    MESSAGE(.NORMAL,LAST(NORMAL));

 71  2      CALL    MESSAGE(.MESS$5R12C5,LAST(MESS$5R12C5));
 72  2      CALL    MESSAGE(.BLINK,LAST(BLINK));
 73  2      CALL    MESSAGE(.MESS$5,LAST(MESS$5));
 74  2      CALL    GET(' ');
 75  2      CALL    MESSAGE(.MESS$5R12C5,LAST(MESS$5R12C5));
 76  2      CALL    MESSAGE(.NORMAL,LAST(NORMAL));

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

              /* Check that PISTON$ADVANCED$SWITCH is closed,       */
              /*  if not then advance the piston till it is closed  */
              /*      i.e  tool is at datum.                        */

 77  2          PORT$2BH$INPUT=INPUT(PORT$2BH); /* READ PORT */
 78  2          IF (PORT$2BH$INPUT AND PORT$2BH$ADVANCED)=0 THEN
 79  2          DO;
 80  3             OUTPUT(22H)=59; /* START ADVANCE */
 81  3             PORT$2BH$INPUT=INPUT(PORT$2BH);  /* READ AGAIN */

                   /* CONTINUE TO ADVANCE TILL SWITCH IS CLOSED */
 82  3             DO WHILE (PORT$2BH$INPUT AND PORT$2BH$ADVANCED)=0;
 83  4                PORT$2BH$INPUT=INPUT(PORT$2BH);
 84  4             END;  /* DO WHILE */
 85  3          END; /* DO */
 86  2          OUTPUT(22H)=128; /* STOP PISTON ADVANCE */

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

 87  2      CALL    MESSAGE(.MESS$6,LAST(MESS$6));
 88  2      CALL    GET(' ');
 89  2      CALL    MESSAGE(.ST$ERA,LAST(ST$ERA));
 90  2      CALL    MESSAGE(.ERASE,LAST(ERASE));

 91  2      END MACHINE$SETUP;

              /*************************************************************
            - */

 92  1      END MACHINE$SETUP$MODULE;
```

```
MODULE INFORMATION:

     CODE AREA SIZE     = 031EH    798D
     VARIABLE AREA SIZE = 0002H      2D
     MAXIMUM STACK SIZE = 0002H      2D
     249 LINES READ
     0 PROGRAM ERROR(S)

END OF PL/M-80 COMPILATION
```

```
ISIS-II PL/M-80 V3.1 COMPILATION OF MODULE SHAPESELECTMODULE
OBJECT MODULE PLACED IN :F2:shape.OBJ
COMPILER INVOKED BY:  plm80 :F2:shape.plm DEBUG


           $WORKFILES(:F2:,:F2:)
           $PAGEWIDTH(80)
           $PAGELENGTH(55)

  1        SHAPE$SELECT$MODULE:
           DO;
              /* FILE$NAME  'SHAPE' */

           /************************************************************/
           /*                                                          */
           /******     EXTERNAL  VARIABLES  DECLARATIONS          ******/
           /*                                                          */
           /************************************************************/

              /* NONE */

           /************************************************************/
           /*                                                          */
           /******     PUBLIC  VARIABLES  DECLARATIONS            ******/
           /*                                                          */
           /************************************************************/

              /* NONE */

           /************************************************************/
           /*                                                          */
           /******     LOCAL  VARIABLES  DECLARATIONS             ******/
           /*                                                          */
           /************************************************************/

              /* ASCII CODE CHARACTERS */

  2   1       DECLARE BELL LITERALLY  '07H';
  3   1       DECLARE BACK$SPACE LITERALLY  '08H';
  4   1       DECLARE LINE$FEED  LITERALLY  '0AH';
  5   1       DECLARE CARRIAGE$RETURN   LITERALLY  '0DH';
  6   1       DECLARE CLEAR$SCREEN  LITERALLY  '1AH';
  7   1       DECLARE ESC  LITERALLY '1BH';
  8   1       DECLARE HOME LITERALLY  '1EH';
  9   1       DECLARE SP  LITERALLY  '20H';
 10   1       DECLARE DELETE  LITERALLY  '7FH';
 11   1       DECLARE QUOTE LITERALLY '27H';     /* SINGLE QUOTE */
 12   1       DECLARE DQUOTE LITERALLY '22H';    /* DOUBLE QUOTE */

              /* MODULE VARIABLES */
```

```
 13  1          DECLARE LETTER BYTE;

                /*************************************************************/
                /*                                                           */
                /******     EXTERNAL  PROCEDURES  DECLARATIONS           ******/
                /*                                                           */
                /*************************************************************/
                                                                          /* */
 14  1          CONSOLE$OUT:                                              /* */
                PROCEDURE(CHAR) EXTERNAL;                                 /* */
 15  2              DECLARE CHAR BYTE;                                    /* */
 16  2          END CONSOLE$OUT;                                          /* */
                                                                          /* */
                /*************************************************************/
                                                                          /* */
 17  1          MESSAGE:                                                  /* */
                PROCEDURE(POINTER,LAST$ELEMENT) EXTERNAL;                 /* */
 18  2              DECLARE POINTER ADDRESS;                              /* */
 19  2              DECLARE LAST$ELEMENT ADDRESS;                         /* */
 20  2          END MESSAGE;                                              /* */
                                                                          /* */
                /*************************************************************/
                                                                          /* */
 21  1          CONSOLE$IN:                                               /* */
                PROCEDURE BYTE EXTERNAL;                                  /* */
 22  2          END CONSOLE$IN;                                           /* */
                                                                          /* */
                /*************************************************************/
                                                                          /* */
 23  1          GET:                                                      /* */
                PROCEDURE(TARGET$CHAR) EXTERNAL;                          /* */
 24  2              DECLARE TARGET$CHAR BYTE;                             /* */
 25  2          END GET;                                                  /* */

                /*************************************************************/
                /*                                                           */
                /******      LOCAL  PROCEDURES  DECLARATIONS            ******/
                /*                                                           */
                /*************************************************************/

                   /* NONE */

                /*************************************************************/
                /*                                                           */
                /******      PUBLIC  PROCEDURES  DECLARATIONS           ******/
                /*                                                           */
                /*************************************************************/

 26  1      SHAPE$SELECT:
                PROCEDURE BYTE PUBLIC;

 27  2      DECLARE PAGE$4(*) BYTE DATA
```

```
                (ESC,'=','$','8',           /* R5 C25 */
                'SHAPE SELECT',
                ESC,'=','%','8',            /* R6 C25 */
                '------------',

                ESC,'=','*','(',            /* R11 C9 */
                'A-CONICAL',
                ESC,'=',',',',(',           /* R13 C9 */
                'B-PARABOLIC',

                ESC,'=','2','$',            /* R19 C5 */
                'type  A  or  B');

28   2          DECLARE NOT$1(*)BYTE DATA(
                ESC,'=','7','$',            /* R24 C5 */
                'Press SP to proceed please');

29   2          DECLARE POS$1(*)BYTE DATA
                (ESC,'=','4','*');          /* R21 C11 */

30   2          DECLARE ERR$1(*)BYTE DATA
                (ESC,'=','6','$',           /* R23 C5  */
                ESC,'^',  /* BLINKING CHARACTER */
                'This is a bad character, reenter one from the menu please.',
                ESC,'q'); /* END BLINKING */
31   2          DECLARE DEL$ERR$1(*)BYTE DATA
                (ESC,'=','6','&',           /* R23 C5 */
                ESC,'T');    /* ERASE TO END OF LINE */

                /* SHAPE SELECTION TEXT DISPLAY PAGE */

32   2          CALL MESSAGE(.PAGE$4,LAST(PAGE$4));

                /* KEY IN A LETTER, CHECK THAT IT IS ONE OF THE TWO */
                /*  LETTERS 'A' OR 'B' . IF NOT RING A  BELL AND    */
                /*    ISSUE A BLINKING ERROR MESSAGE.               */

33   2          LETTER=CONSOLE$IN; /* GET LETTER */
34   2          DO WHILE LETTER<>'A' AND LETTER<>'B'; /* TEST IF FALSE */
35   3             CALL    CONSOLE$OUT(BELL); /* RING A BELL */
36   3             CALL    MESSAGE(.ERR$1,LAST(ERR$1));
37   3             LETTER=CONSOLE$IN; /* GET NEW LETTER */
38   3          END; /* WHILE */

39   2          CALL    MESSAGE(.DEL$ERR$1,LAST(DEL$ERR$1));
40   2          CALL    MESSAGE(.POS$1,LAST(POS$1));

41   2          IF   LETTER='A'  THEN  CALL CONSOLE$OUT('A');
43   2          ELSE                   CALL CONSOLE$OUT('B');

44   2          CALL MESSAGE(.NOT$1,LAST(NOT$1));
45   2          CALL GET(' ');
```

```
 46  2      RETURN LETTER;

 47  2      CALL    CONSOLE$OUT(CLEAR$SCREEN);

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

 48  2      END SHAPE$SELECT;

 49  1      END SHAPE$SELECT$MODULE;


MODULE INFORMATION:

     CODE AREA SIZE     = 0135H    309D
     VARIABLE AREA SIZE = 0001H      1D
     MAXIMUM STACK SIZE = 0002H      2D
     160 LINES READ
     0 PROGRAM ERROR(S)

END OF PL/M-80 COMPILATION
```

```
ISIS-II PL/M-80 V3.1 COMPILATION OF MODULE CONEGENERATIONMODULE
OBJECT MODULE PLACED IN :F2:conege.OBJ
COMPILER INVOKED BY:  plm80 :F2:conege.plm DEBUG


                $WORKFILES(:F2:,:F2:)
                $PAGEWIDTH(80)
                $PAGELENGTH(55)

      1         CONE$GENERATION$MODULE:
                DO;
                    /* FILE$NAME  'CONEGE' */

                /************************************************************/
                /*                                                          */
                /*******      EXTERNAL  VARIABLES  DECLARATIONS         ******/
                /*                                                          */
                /************************************************************/

                   /* NONE */

                /************************************************************/
                /*                                                          */
                /*******      PUBLIC  VARIABLES  DECLARATIONS          *******/
                /*                                                          */
                /************************************************************/

      2   1        DECLARE FORM$LENGTH$MM BYTE PUBLIC;
      3   1        DECLARE INCREMENTS$ENCODER$PULSES BYTE PUBLIC;
      4   1        DECLARE CONE$ANGLE$DEGREES BYTE PUBLIC;
      5   1        DECLARE ROLLER$POSITION$ADC$STEPS ADDRESS PUBLIC;

                /************************************************************/
                /*                                                          */
                /*******     LOCAL  VARIABLES  DECLARATIONS             *****/
                /*                                                          */
                /************************************************************/

                   /* ASCII CODE CHARACTERS */

      6   1        DECLARE  ESC    LITERALLY '1BH';
      7   1        DECLARE  QUOTE  LITERALLY '27H';  /* SINGLE QUOTE */
      8   1        DECLARE  DQUOTE LITERALLY '22H';  /* DOUBLE QUOTE */

      9   1        DECLARE  SPACE  LITERALLY '20H';
     10   1        DECLARE  BACK$SPACE LITERALLY '08H';
     11   1        DECLARE  BELL LITERALLY '07H';
     12   1        DECLARE  CLEAR$SCREEN LITERALLY '1AH';

     13   1        DECLARE  ROLLER$ADJUST BYTE;
     14   1        DECLARE  INCREMENTS BYTE;
```

```
 15  1        DECLARE  DIRECTION BYTE;

              /* 592 ADC STEPS IS EQUIVALENT TO 74 MM OF */
              /*          TRANSDUCER LENGTH              */

 16  1        DECLARE  FORMER$TIP$ADC$STEPS LITERALLY '592';

 17  1        DECLARE  TEST$POSITION BYTE;

            /*************************************************************/
            /*                                                           */
            /*****      EXTERNAL  PROCEDURES  DECLARATIONS           *****/
            /*                                                           */
            /*************************************************************/

            /*************************************************************/
                                                                    /* */
 18  1          CONSOLE$OUT:                                      /* */
              PROCEDURE(CHAR) EXTERNAL;                           /* */
 19  2                DECLARE CHAR BYTE;                         /* */
 20  2          END CONSOLE$OUT;                                  /* */
                                                                    /* */
            /*************************************************************/
                                                                    /* */
 21  1          MESSAGE:                                          /* */
              PROCEDURE(POINTER,LAST$ELEMENT) EXTERNAL;           /* */
 22  2                DECLARE POINTER ADDRESS;                   /* */
 23  2                DECLARE LAST$ELEMENT ADDRESS;              /* */
 24  2          END MESSAGE;                                      /* */
                                                                    /* */
            /*************************************************************/
                                                                    /* */
 25  1          CONSOLE$IN:PROCEDURE BYTE EXTERNAL;               /* */
 26  2          END CONSOLE$IN;                                   /* */
                                                                    /* */
            /*************************************************************/
                                                                    /* */
 27  1          GET:                                              /* */
              PROCEDURE(TARGET$CHAR) EXTERNAL;                    /* */
 28  2             DECLARE TARGET$CHAR BYTE;                      /* */
 29  2          END GET;                                          /* */
                                                                    /* */
            /*************************************************************/
                                                                    /* */
 30  1          DECIMAL$VALUE$INPUT:                              /* */
              PROCEDURE BYTE EXTERNAL;                            /* */
 31  2          END DECIMAL$VALUE$INPUT;                          /* */
                                                                    /* */
            /*************************************************************/

            /*************************************************************/
            /*                                                           */
```

```
                /*****      LOCAL   PROCEDURES   DECLARATIONS            *****/
                /*                                                          */
                /************************************************************/

                /************************************************************/
                /* FUNCTION: BACK$DELETE$POSITION                           */
                /* PARAMETERS INPUT: NONE                                   */
                /* OUTPUTS: NONE                                            */
                /* PROCEDURE:                                               */
                /* CALLS: CONSOLE$OUT                                       */
                /* DESCRIPTION: A PROCEDURE TO BACKSPACE TWO POSITIONS, DELETE  */
                /*           TWO CHARACTERS FROM SCREEN, THEN BACKSPACE TWO  */
                /*         POSITIONS, WHERE IT WAS INITIALLY. THIS PROCEDURE */
                /*         THEN REQUESTS OPERATOR TO ENTER THE CORRECT NUMBER */
                /*    AFTER A NUMBER NOT WITH IN THE SHOWN RANGE HAS BEEN TYPED */
                /*  IN.                                                     */
                /************************************************************/
                                                                      /* */
 32   1         BACK$DELETE$POSITION:                                 /* */
                PROCEDURE;                                          /* */
 33   2           CALL  CONSOLE$OUT(BACK$SPACE);                     /* */
 34   2           CALL  CONSOLE$OUT(BACK$SPACE);                     /* */
 35   2           CALL  CONSOLE$OUT(SPACE);                          /* */
 36   2           CALL  CONSOLE$OUT(SPACE);                          /* */
 37   2           CALL  CONSOLE$OUT(BACK$SPACE);                     /* */
 38   2           CALL  CONSOLE$OUT(BACK$SPACE);                     /* */
 39   2         END BACK$DELETE$POSITION;                            /* */
                                                                      /* */
                /************************************************************/

                /************************************************************/
                /*                                                          */
                /*****      PUBLIC   PROCEDURES   DECLARATIONS           *****/
                /*                                                          */
                /************************************************************/

                /************************************************************/
                /* FUNCTION: CONE$GENERATION                                */
                /* PARAMETERS INPUT: CONICAL PARAMETERS FROM KEYBOARD       */
                /* OUTPUTS: NONE                                            */
                /* PROCEDURE: PUBLIC                                        */
                /* CALLS: MESSAGE, DECIMAL$VALUE$INPUT, BACK$DELETE$POSITION,   */
                /*       CONSOLE$OUT                                        */
                /* DESCRIPTION: A PROCEDURE TO DISPLAY CONICAL PARAMETES ON     */
                /*           SCREEN, THEN OPERATOR IS REQUIRED TO FILL IN THE */
                /*        VARIABLES WITH APROPRIATE VALUES SELECTED FROM EACH */
                /*     VARIABLE LIMITS SHOWN ON THE SCREEN. FAILING TO DO SO  */
                /*   WILL RING A BELL AND OPERATOR IS REQUESTED TO ENTER THE   */
                /* RIGHT VALUE AGAIN.                                       */
                /************************************************************/

 40   1     CONE$GENERATION:
```

```
                PROCEDURE PUBLIC;

                    /* CONICAL CONTOUR PARAMETERS DISPLAY */

 41   2         DECLARE PAGE$5(*) BYTE DATA
                (ESC,'=','!','7',                   /* R2 C24 */
                'CONICAL  CONTOUR  PARAMETERS',
                ESC,'=',DQUOTE,'7',                 /* R3 C24 */
                '--------------------------',

                ESC,'=','%',')',                    /* R6 C10 */
                'Form Length (mm)                   Increments (Encoder Pulses)',
                ESC,'=','&',')',                    /* R7 C10 */
                '(30 to 74)                         (8 to 16)',

                ESC,'=',QUOTE,',',          /*       R8 C13          */
                ESC,'j',                    /* START REVERSE VIDEO   */
                ESC,'=',QUOTE,'/',          /*       R8 C16          */
                ESC,'k',                    /* END REVERSE VIDEO     */
                ESC,'=',QUOTE,'N',          /*       R8 C47          */
                ESC,'j',                    /* START REVERSE VIDEO   */
                ESC,'=',QUOTE,'Q',          /*       R8 C50          */
                ESC,'k',                    /* END REVERSE VIDEO     */

                ESC,'=','+','&',            /* R12 C7 */
                'Cone Angle (Degrees)',
                ESC,'=',',','&',            /* R13 C7 */
                '(30 to 45)',

                ESC,'=','-',')',            /* R14 C10 */
                ESC,'j',                    /* START REVERSE VIDEO  */
                ESC,'=','-',',',            /*      R14 C13         */
                ESC,'k',                    /* END REVERSE VIDEO    */

                ESC,'=','6','&',          /* R23 C7 */
                'After each parameter press RETURN please.',
                ESC,'=','7','!',          /* R24 C2 */
                'If incorrect value is entered, use DEL key to erase before RETURN
              - .');

 42   2         DECLARE NOT$1(*)BYTE DATA(
                ESC,'=','7','&',                /* R24 C7 */
                'Press SP to proceed please');

 43   2         DECLARE POS$1(*) BYTE DATA
                (ESC,'=',QUOTE,'-');            /* R8 C14 */

 44   2         DECLARE POS$2(*) BYTE DATA
                (ESC,'=',QUOTE,'0');            /* R8 C48 */

 45   2         DECLARE POS$3(*) BYTE DATA
                (ESC,'=','-','*');              /* R14 C11 */
```

```
 46  2     DECLARE ERR$1(*) BYTE DATA
           (ESC,'=','5','&',                /* R22 C7 */
           ESC,'^',   /* BLINKING CHARACTER */
           'Parameter not within limits, reenter please.',
           ESC,'q'); /* END BLINKING */

 47  2     DECLARE ERR$2(*) BYTE DATA
           (ESC,'=','5','&',                /* R22 C7 */
           ESC,'^',   /* BLINKING CHARACTER */
           'Enter either ''Y'' or ''N'' please.',
           ESC,'T'); /* ERASE TO END OF LINE */

 48  2     DECLARE ERR$3(*) BYTE DATA
           (ESC,'=','5','&',                /* R22 C7 */
           ESC,'^',   /* BLINKING CHARACTER */
           'Enter either ''F'' or ''R'' please.',
           ESC,'T'); /* ERASE TO END OF LINE */

 49  2     DECLARE DEL$ERR(*) BYTE DATA
           (ESC,'=','5','&',                /* R22 C7 */
           ESC,'T');

 50  2     DECLARE ADJ$1(*) BYTE DATA(
           ESC,'=','/',QUOTE,               /* R16 C8 */
           'Do you require roller adjustment ? (Y/N)');

 51  2     DECLARE ADJ$2(*) BYTE DATA(
           ESC,'=','1',QUOTE,                /* R18 C8 */
           'How many increments ? (1-80) (1 increment=1/8 mm)');

 52  2     DECLARE ADJ$3(*) BYTE DATA(
           ESC,'=','3',QUOTE,                /* R20 C8 */
           'Forward or reverse ? (F/R)');

 53  2     DECLARE POS$4(*) BYTE DATA(
           ESC,'=','0',')');                /* R17 C10 */

 54  2     DECLARE POS$5(*) BYTE DATA(
           ESC,'=','2',')');                /* R19 C10 */

 55  2     DECLARE POS$6(*) BYTE DATA(
           ESC,'=','4',')');                /* R21 C10 */

 56  2     DECLARE POS$7(*) BYTE DATA(
           ESC,'=','1','&');                /* R18 C7  */

 57  2     DECLARE TEST$1(*) BYTE DATA(
           ESC,'=','0','&',                 /* R17 C7  */
           'Do you require the roller test position (Y/N) ?');

 58  2     DECLARE DEL$T(*) BYTE DATA(
```

```
                ESC,'=','0','&',                /* R17 C7 */
                ESC,'T',
                ESC,'=','1','&',                /* R18 C7 */
                ESC,'T',
                ESC,'=','7',' ',                /* R24 C1 */
                ESC,'T');

 59   2     DECLARE DEL$F(*) BYTE DATA(
                ESC,'=','6',' ',                /* R23 C1 */
                ESC,'T',
                ESC,'=','7',' ',                /* R24 C1 */
                ESC,'T');

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

 60   2        CALL  CONSOLE$OUT(CLEAR$SCREEN);

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

 61   2        CALL    MESSAGE(.PAGE$5,LAST(PAGE$5));

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

                /*        CONE PARAMETERS INPUT                              */

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

                   /* INPUT FORM$LENGTH$MM */

 62   2        CALL    MESSAGE(.POS$1,LAST(POS$1)); /* POSITION */
 63   2        FORM$LENGTH$MM=DECIMAL$VALUE$INPUT; /* GET VALUE */
 64   2        DO WHILE  FORM$LENGTH$MM<30 OR FORM$LENGTH$MM>74; /* CHECK */
 65   3           CALL  BACK$DELETE$POSITION;
 66   3           CALL  MESSAGE(.ERR$1,LAST(ERR$1)); /* ERROR MESSAGE */
 67   3           CALL  CONSOLE$OUT(BELL); /* RING A BELL */
 68   3           CALL  MESSAGE(.POS$1,LAST(POS$1)); /* REPOSITION */
 69   3           FORM$LENGTH$MM=DECIMAL$VALUE$INPUT; /* GET NEW VALUE */
 70   3        END; /* WHILE */
 71   2        CALL    MESSAGE(.DEL$ERR,LAST(DEL$ERR));

                   /* INPUT INCREMENTS$ENCODER$PULSES */

 72   2        CALL    MESSAGE(.POS$2,LAST(POS$2)); /* POSITION */
 73   2           INCREMENTS$ENCODER$PULSES=DECIMAL$VALUE$INPUT; /* GET VALU
     -  E */
 74   2        DO WHILE  INCREMENTS$ENCODER$PULSES>16 OR INCREMENTS$ENCODER$P
     -  ULSES<8;
 75   3           CALL  BACK$DELETE$POSITION;
 76   3           CALL  MESSAGE(.ERR$1,LAST(ERR$1)); /* ERROR MESSAGE */
 77   3           CALL  CONSOLE$OUT(BELL); /* RING A BELL */
 78   3           CALL  MESSAGE(.POS$2,LAST(POS$2)); /* REPOSITION */
 79   3           INCREMENTS$ENCODER$PULSES=DECIMAL$VALUE$INPUT; /* GET NEW
```

```
        -   VALUE */
 80   3        END; /* WHILE */
 81   2        CALL    MESSAGE(.DEL$ERR,LAST(DEL$ERR));


                /* INPUT CONE$ANGLE$DEGREES */

 82   2        CALL    MESSAGE(.POS$3,LAST(POS$3)); /* POSITION */
 83   2        CONE$ANGLE$DEGREES=DECIMAL$VALUE$INPUT; /* GET VALUE */
 84   2        DO WHILE  CONE$ANGLE$DEGREES<30 OR CONE$ANGLE$DEGREES>45;
 85   3            CALL  BACK$DELETE$POSITION;
 86   3            CALL  MESSAGE(.ERR$1,LAST(ERR$1)); /* ERROR MESSAGE */
 87   3            CALL  CONSOLE$OUT(BELL); /* RING A BELL */
 88   3            CALL  MESSAGE(.POS$3,LAST(POS$3)); /* REPOSITION */
 89   3            CONE$ANGLE$DEGREES=DECIMAL$VALUE$INPUT; /* GET NEW VALUE *
        -   /
 90   3        END; /* WHILE */
 91   2        CALL    MESSAGE(.DEL$ERR,LAST(DEL$ERR));

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

               /*  TO COMPENSATE FOR THE ROLLER RADIUS, THIS DISTANCE HAS TO   */
               /*    BE ADDED TO THE INITIAL POSITION                          */
               /*   R(1-COS(30)) = 9(1-0.866)                                  */
               /*                = 1.2 MM  WHICH IS EQUIVALENT OF 10 ADC$STEPS */

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

 92   2       ROLLER$POSITION$ADC$STEPS=FORMER$TIP$ADC$STEPS+10;

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

 93   2       CALL MESSAGE(.DEL$F,LAST(DEL$F));  /* DELETE FOOTNOTE */

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

                /* ROLLER TEST POSITION REQUIRED ? */

 94   2        CALL MESSAGE(.TEST$1,LAST(TEST$1));
 95   2        TEST$POSITION=CONSOLE$IN;
 96   2        DO WHILE TEST$POSITION<>'Y' AND TEST$POSITION<>'N';
 97   3           CALL CONSOLE$OUT(BELL);
 98   3           CALL MESSAGE(.ERR$2,LAST(ERR$2));
 99   3           TEST$POSITION=CONSOLE$IN;
100   3        END; /* DO WHILE */
101   2        CALL MESSAGE(.DEL$ERR,LAST(DEL$ERR));
102   2        IF TEST$POSITION='N' THEN
103   2            DO;
104   3               CALL MESSAGE(.POS$7,LAST(POS$7));
105   3               CALL CONSOLE$OUT(TEST$POSITION);
106   3               CALL MESSAGE(.NOT$1,LAST(NOT$1));
107   3               CALL GET(' ');
```

```
108  3              CALL MESSAGE(.DEL$T,LAST(DEL$T));

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

                   /* ROLLER ADJUSTMENT REQUIRED ? */

109  3             CALL MESSAGE(.ADJ$1,LAST(ADJ$1));
110  3             ROLLER$ADJUST=CONSOLE$IN;
111  3             DO WHILE ROLLER$ADJUST<>'Y' AND ROLLER$ADJUST<>'N';
112  4                 CALL  CONSOLE$OUT(BELL);
113  4                 CALL  MESSAGE(.ERR$2,LAST(ERR$2));
114  4                 ROLLER$ADJUST=CONSOLE$IN;
115  4             END;
116  3             CALL    MESSAGE(.DEL$ERR,LAST(DEL$ERR));
117  3             CALL    MESSAGE(.POS$4,LAST(POS$4));
118  3             CALL    CONSOLE$OUT(ROLLER$ADJUST);

119  3             IF ROLLER$ADJUST='Y' THEN
120  3             DO;

                   /* INPUT ROLLER ADJUSTMENT */

121  4             CALL MESSAGE(.ADJ$2,LAST(ADJ$2));
122  4             CALL MESSAGE(.POS$5,LAST(POS$5));
123  4             INCREMENTS=DECIMAL$VALUE$INPUT;
124  4             DO WHILE INCREMENTS<1 OR INCREMENTS>80;
125  5                 CALL  BACK$DELETE$POSITION;
126  5                 CALL  MESSAGE(.ERR$1,LAST(ERR$1));
127  5                 CALL  CONSOLE$OUT(BELL);
128  5                 CALL  MESSAGE(.POS$5,LAST(POS$5));
129  5                 INCREMENTS=DECIMAL$VALUE$INPUT;
130  5             END;
131  4             CALL MESSAGE(.DEL$ERR,LAST(DEL$ERR));

                   /* INPUT ADJUSTMENT DIRECTION */

132  4             CALL MESSAGE(.ADJ$3,LAST(ADJ$3));
133  4             DIRECTION=CONSOLE$IN;
134  4             DO WHILE DIRECTION<>'F' AND DIRECTION<>'R';
135  5                 CALL  CONSOLE$OUT(BELL);
136  5                 CALL  MESSAGE(.ERR$3,LAST(ERR$3));
137  5                 DIRECTION=CONSOLE$IN;
138  5             END;
139  4             CALL  MESSAGE(.DEL$ERR,LAST(DEL$ERR));
140  4             CALL  MESSAGE(.POS$6,LAST(POS$6));
141  4             CALL  CONSOLE$OUT(DIRECTION);

142  4             IF DIRECTION='F' THEN
143  4             ROLLER$POSITION$ADC$STEPS=ROLLER$POSITION$ADC$STEPS-INCREMENTS
          -  ;

                   ELSE
```

```
144   4          ROLLER$POSITION$ADC$STEPS=ROLLER$POSITION$ADC$STEPS+INCREMENTS
          -    ;
145   4          END;  /* IF ROLLER ADJUST = 'Y' */

146   3          CALL MESSAGE(.NOT$1,LAST(NOT$1));
147   3          CALL GET(' ');
148   3          CALL CONSOLE$OUT(CLEAR$SCREEN);

149   3          END; /* IF TEST$POSITION='N' */
                 ELSE /* IF TEST$POSITION='Y' THEN */
150   2           DO;
151   3             CALL MESSAGE(.POS$7,LAST(POS$7));
152   3             CALL CONSOLE$OUT(TEST$POSITION);
153   3             CALL MESSAGE(.NOT$1,LAST(NOT$1));
154   3             CALL GET(' ');

                   /* 552 ADC STEPS IS EQUIVALENT TO 69 MM OF */
                   /*          TRANSDUCER LENGTH              */

155   3                 ROLLER$POSITION$ADC$STEPS=552;
156   3           END;
               /*****************************************************************/

157   2        END CONE$GENERATION;

158   1        END CONE$GENERATION$MODULE;
```

MODULE INFORMATION:

```
    CODE AREA SIZE     = 0612H   1554D
    VARIABLE AREA SIZE = 0009H      9D
    MAXIMUM STACK SIZE = 0004H      4D
    429 LINES READ
    0 PROGRAM ERROR(S)
```

END OF PL/M-80 COMPILATION

```
ISIS-II PL/M-80 V3.1 COMPILATION OF MODULE MACHINECONECONTROLMODULE
OBJECT MODULE PLACED IN :F2:conemc.OBJ
COMPILER INVOKED BY:  plm80 :F2:conemc.plm DEBUG


                $WORKFILES(:F2:,:F2:)
                $PAGEWIDTH(80)
                $PAGELENGTH(55)

  1             MACHINE$CONE$CONTROL$MODULE:
                DO;
                    /* FILE$NAME  'CONEMC' */

                /*****************************************************************/
                /*                                                               */
                /******      EXTERNAL  VARIABLES  DECLARATIONS          ******/
                /*                                                               */
                /*****************************************************************/

  2   1             DECLARE FORM$LENGTH$MM BYTE EXTERNAL;
  3   1             DECLARE INCREMENTS$ENCODER$PULSES BYTE EXTERNAL;
  4   1             DECLARE CONE$ANGLE$DEGREES BYTE EXTERNAL;
  5   1             DECLARE ROLLER$POSITION$ADC$STEPS ADDRESS EXTERNAL;

  6   1             DECLARE PORT$2AH$OUTPUT BYTE EXTERNAL;

                /*****************************************************************/
                /*                                                               */
                /******      PUBLIC  VARIABLES  DECLARATIONS            ******/
                /*                                                               */
                /*****************************************************************/

                    /* NONE */

                /*****************************************************************/
                /*                                                               */
                /******      LOCAL  VARIABLES  DECLARATIONS             ******/
                /*                                                               */
                /*****************************************************************/

  7   1         DECLARE I BYTE; /* GET THE CONE$ANGLE$DEGREES INDEX */

  8   1         DECLARE CARRIAGE$POSITION$ENCODER$PULSES ADDRESS;

  9   1         DECLARE TAN$NUM(16)BYTE DATA(
                    26,3,78,37,56,7,53,49,25,17,47,73,9,
                    69,28,1);

 10   1         DECLARE TAN$DEN(16)BYTE DATA(
                    45,5,125,57,83,10,73,65,32,
                    21,56,84,10,74,29,1);
```

```
                /* ASCII CODE CHARACTERS */

 11   1      DECLARE ESC LITERALLY '1BH';
 12   1      DECLARE QUOTE LITERALLY '27H';
 13   1      DECLARE CLEAR$SCREEN LITERALLY '1AH';
 14   1      DECLARE BELL LITERALLY '07H';
 15   1      DECLARE CARRIAGE$RETURN LITERALLY '0DH';

 16   1      DECLARE TURNING$REQUIRED BYTE;
 17   1      DECLARE TRUE LITERALLY '0FFH';
 18   1      DECLARE FALSE LITERALLY '0';
 19   1      DECLARE ANSWER BYTE;

 20   1      DECLARE COUNTER$RESET LITERALLY '0000$0001B';
 21   1      DECLARE COUNTER$ENABLE LITERALLY '1111$1110B';

 22   1      DECLARE CLEAR$DATA$FIELD LITERALLY '0';
 23   1      DECLARE CLEAR$ADDRESS$FIELD LITERALLY '0';

 24   1      DECLARE KBD$DPLY$CONTROL BYTE AT (1900H);
 25   1      DECLARE KMODE LITERALLY '0';
 26   1      DECLARE KBNIT LITERALLY '0CCH';

 27   1      DECLARE PAGE$7(*) BYTE DATA
               (ESC,'=',' ','8',                    /* R1  C25 */
               'Flow-turning roller movements',
                ESC,'=','!','8',                    /* R2  C25 */
               '----------------------------',
                ESC,'=','£','$',                    /* R4  C5  */
               '1- Roller advancing (4-5)',
                ESC,'=','$','$',                    /* R5  C5  */
               '2- Lathe switched on, carriage moving left (5-1)',
                ESC,'=','%','$',                    /* R6  C5  */
               '3- Conical contour path (1-2)',
                ESC,'=','&','$',                    /* R7  C5  */
               '4- Lathe switched off, roller retracted (2-3)',
                ESC,'=',QUOTE,'$',                  /* R8  C5  */
               '5- Manual movement to datum (3-4)',
                ESC,'=',')','=',                    /* R10 C30 */
               '1                              5',
                ESC,'=','+','=',                    /* R12 C30 */
               '++++++++++++++++++++++++++++++',
                ESC,'=',',',';',                    /* R13 C28 */
               '+                             +',
                ESC,'=','-','9',                    /* R14 C26 */
               '+                              +',
                ESC,'=','.','7',                    /* R15 C24 */
               '+                               +',
                ESC,'=','/','3',                    /* R16 C20 */
               '2 +                               +',
                ESC,'=','0','3',                    /* R17 C20 */
```

```
                '+                                        +',
                 ESC,'=','1','3',                  /* R18 C20 */
                '+                                        +',
                 ESC,'=','2','3',                  /* R19 C20 */
                '++++++++++++++++++++++++++++++++++++++++',
                 ESC,'=','4','3',                  /* R21 C20 */
                '3                                        4  DATUM ');

 28  1      DECLARE PAGE$B(*) BYTE DATA(
                ESC,'=','%',':',                   /* R6  C27 */
                'Flow-turning is completed',
                ESC,'=','&',':',                   /* R7  C27 */
                '------------------------',
                ESC,'=',')','$',                   /* R10 C5  */
                '1- Please remove the finished component.',
                ESC,'=','+','$',                   /* R12 C5  */
                '2- Disengage the carriage and return to datum by
                 the handwheel.',
                ESC,'=','3','$',                   /* R20 C5  */
                'Another identical cone ?  (Y/N) then RETURN');
 29  1      DECLARE MESS$1(*) BYTE DATA(
                ESC,'=','5','$');                  /* R22 C5  */
 30  1      DECLARE ERROR$1(*) BYTE DATA(
                ESC,'^',   /* BLINKING STARTS  */
                ESC,'=','7','$',                   /* R24 C5  */
                'Enter either Y OR N please');
 31  1      DECLARE DELETE$ERROR$1(*) BYTE DATA(
                ESC,'=','7','$',                   /* R24 C5  */
                ESC,'T');  /* ERASE TO END OF LINE */

                /************************************************************/
                /*                                                          */
                /******      EXTERNAL  PROCEDURES  DECLARATIONS         ******/
                /*                                                          */
                /************************************************************/

 32  1          UPDDT:
                PROCEDURE(PAR$1) EXTERNAL;
 33  2            DECLARE PAR$1 BYTE;
 34  2          END UPDDT;

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

 35  1          UPDAD:
                PROCEDURE(PAR$2) EXTERNAL;
 36  2            DECLARE PAR$2 ADDRESS;
 37  2          END UPDAD;

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

 38  1          CONSOLE$OUT:
                PROCEDURE(CHAR) EXTERNAL;
```

```
 39  2          DECLARE CHAR BYTE;
 40  2        END CONSOLE$OUT;

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

 41  1        MESSAGE:
              PROCEDURE(POINTER,LAST$ELEMENT) EXTERNAL;
 42  2            DECLARE POINTER ADDRESS;
 43  2            DECLARE LAST$ELEMENT ADDRESS;
 44  2        END MESSAGE;

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

 45  1        CONSOLE$IN:PROCEDURE BYTE EXTERNAL;
 46  2        END CONSOLE$IN;

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

 47  1        GET:
              PROCEDURE(TARGET$CHAR) EXTERNAL;
 48  2            DECLARE TARGET$CHAR BYTE;
 49  2        END GET;

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

           /*************************************************************/
           /*                                                           */
           /******      LOCAL  PROCEDURES  DECLARATIONS             ******/
           /*                                                           */
           /*************************************************************/

           /*************************************************************/
           /* FUNCTION: SHAFT$ENCODER (ENCODER 8-BIT INPUT)             */
           /* PARAMETERS INPUT: NONE                                    */
           /* OUTPUTS: 16-BIT VALUE IN (.CARRIAGE$POSITION$ENCODER$PULSES) */
           /* PROCEDURE: LOCAL                                          */
           /* CALLS: NOTHING                                            */
           /* DESCRIPTION: THE ACCUMULATIVE VALUE OF THE COUNTER WILL BE  */
           /*            STORED IN TWO BYTES(16-BIT), THE HIGH AND THE  */
           /*          LOW. PORT$29H WILL BE READ, THE LOW BYTE WILL ALL */
           /*        THE TIME BE READ AND UPDATED, IF THE VALUE EXCEEDS  */
           /*     256 THEN 1 WILL BE ADDED TO THE HIGH BYTE. PULSE.HIGH  */
           /*  AND PULSE.LOW ARE STORED IN MEMORY WHERE THEY ARE REFERED */
           /* TO BY THE GLOBAL VARIABLE CARRIAGE$POSITION$ENCODER$PULSES. */
           /*************************************************************/
                                                                  /* */
 50  1     SHAFT$ENCODER:                                         /* */
           PROCEDURE ADDRESS;                                     /* */
 51  2        DECLARE  PULSE STRUCTURE(LOW BYTE,HIGH BYTE) AT     /* */
                        (.CARRIAGE$POSITION$ENCODER$PULSES);      /* */
 52  2        DECLARE READ$COUNT BYTE;                           /* */
 53  2        DECLARE NO$OF$READS LITERALLY '10';                /* */
```

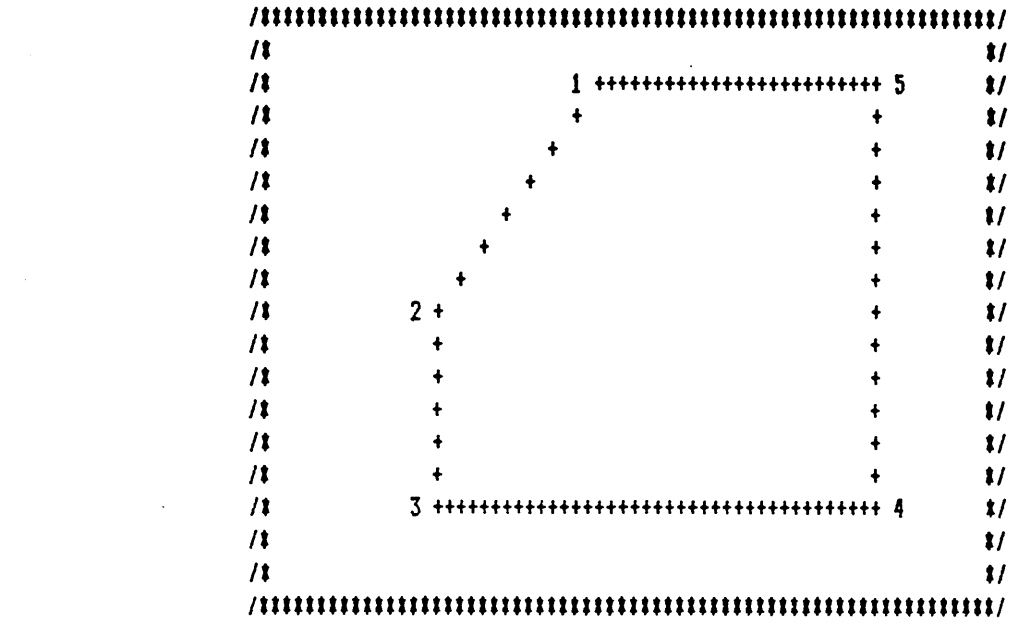
```
 54   2          DECLARE  COUNTER BYTE;                                  /* */
 55   2          DECLARE  COUNTER$1 BYTE;                                /* */
 56   2          DECLARE  PORT$29H LITERALLY '29H';                      /* */
                                                                         /* */
              /* THE CODE DEBOUNCES COUNTER BY SOFTWARE */

 57   2      RETRY:
                 READ$COUNT=NO$OF$READS;
 58   2          COUNTER=INPUT(PORT$29H);
 59   2          DO WHILE READ$COUNT>0;
 60   3            COUNTER$1=INPUT(PORT$29H);
 61   3            IF COUNTER<>COUNTER$1 THEN GOTO RETRY;
 63   3            READ$COUNT=READ$COUNT-1;
 64   3          END; /* WHILE */

 65   2          IF COUNTER<PULSE.LOW  THEN                              /* */
 66   2             PULSE.HIGH=PULSE.HIGH+1;                             /* */
 67   2          PULSE.LOW=COUNTER;                                      /* */

                 /* DISPLAY COUNTER'S VALUE ON SDK-85 LED'S */
 68   2          CALL UPDDT(COUNTER);
 69   2          CALL UPDAD(CARRIAGE$POSITION$ENCODER$PULSES);

 70   2          RETURN CARRIAGE$POSITION$ENCODER$PULSES;                /* */
 71   2      END SHAFT$ENCODER;                                          /* */
                                                                         /* */
              /*************************************************************/


              /*************************************************************/
              /* FUNCTION: ADC$INPUT (12-BIT ADC)                          */
              /* PARAMETERS INPUT: ANALOGUE VOLTAGE (0-10) VOLTS           */
              /* OUTPUTS: DIGITAL VALUES (0 - 1023)                        */
              /* PROCEDURE: LOCAL                                          */
              /* CALLS: TIME                                               */
              /* DESCRIPTION: OUTPUT PULSE LOW (00) AND THEN OUTPUT        */
              /*            PULSE HIGH (02) VIA PORT$2AH BIT$1 TO START    */
              /*          CONVERSION, WAITS TILL CONVERSION IS COMPLETED,   */
              /*        THEN INPUTS THE VALUES FROM PORTS                   */
              /*            21H (8-BIT)       8 LSB'S                     */
              /*            23H (9-12 BIT)    4 MSB'S                     */
              /*************************************************************/
                                                                        /* */
 72   1      ADC$INPUT:                                                 /* */
              PROCEDURE ADDRESS;                                        /* */
 73   2        DECLARE ADC$CONVERT LITERALLY '1111$1101B';              /* */
 74   2        DECLARE ADC$IDLE    LITERALLY '0000$0010B';              /* */
 75   2        DECLARE PORT$21H LITERALLY '21H'; /* ADC 1-8 BIT          */
 76   2        DECLARE PORT$23H LITERALLY '23H'; /* ADC 9-12 BIT         */
 77   2        DECLARE PORT$2AH LITERALLY '2AH'; /* START CONVERSION     */
 78   2        DECLARE ADC$IN$WORD ADDRESS;                             /* */
```

```
 79  2        OUTPUT(PORT$2AH)=PORT$2AH$OUTPUT AND ADC$CONVERT;        /* */
 80  2        OUTPUT(PORT$2AH)=PORT$2AH$OUTPUT OR  ADC$IDLE;          /* */
 81  2        CALL TIME(1); /* A DELAY OF 100 MICROSECONDS UNTIL        */
                          /*     CONVERSION IS COMPLETED              */

                /* READ PORT 23H, MASK OFF 4 MSB'S, DOUBLE IT(i.e ADD 8 ZERO */
                /*      TO THE LEFT) AND THEN ROTATE LEFT 8 DIGITS.        */

 82  2        ADC$IN$WORD=SHL(DOUBLE(INPUT(PORT$23H) AND 0000$1111B),8);/* */

                /* ADD THIS VALUE TO THE INPUT OF PORT$21H AND THEN ROTATE  */
                /*      RIGHT 2 DIGITS (TO RID OFF 2 LSB'S).               */

 83  2        ADC$IN$WORD=SHR((ADC$IN$WORD+INPUT(PORT$21H)),2);        /* */

 84  2        RETURN  ADC$IN$WORD; /* RETURN 10-BIT VALUE (POSITION)      */
 85  2      END ADC$INPUT;                                             /* */
                                                                       /* */
              /*************************************************************/


              /*************************************************************/
              /* FUNCTION: ADVANCE THE ROLLER A GIVEN DISTANCE.            */
              /* PARAMETERS INPUT: ADVANCE DISTANCE IN ADC STEPS.          */
              /* OUTPUTS:NONE.                                             */
              /* PROCEDURE: LOCAL                                          */
              /* CALLS: ADC$INPUT.                                         */
              /* DESCRIPTION:                                              */
              /*            THE CURRENT TOOL POSITION IS INPUT AND TOOL    */
              /*           ADVANCE INITIATED IF REQUIRED. TOOL POSITION    */
              /*          IS CONTINUALLY MONITORED AND TOOL ADVANCE        */
              /*         TERMINATED WHEN THE SPECIFIED POSITION IS ACHIEVED. */
              /*************************************************************/

 86  1      TOOL$ADVANCE:
              PROCEDURE(ABSOLUTE$POSITION$ADC$STEPS);
 87  2          DECLARE ABSOLUTE$POSITION$ADC$STEPS ADDRESS;
 88  2          DECLARE PRESENT$POSITION$ADC$STEPS ADDRESS;
 89  2          DECLARE ADVANCE$DAC$VALUE LITERALLY '183';
 90  2          DECLARE STOP$DAC$VALUE    LITERALLY '128';
 91  2          DECLARE PORT$22H          LITERALLY '22H';

              /* IF ABSOLUTE POSITION IS LESS THAN THE MAXIMUM VALUE */
              /*       (03FFH), GO THROUGH THE ROUTINE.              */

 92  2      IF ABSOLUTE$POSITION$ADC$STEPS<03FFH  THEN
 93  2      DO;

                /* READ PRESENT POSITION */
 94  3          PRESENT$POSITION$ADC$STEPS=ADC$INPUT;

                /* ADVANCE TOOL */
```

```
 95   3          OUTPUT(PORT$22H)=ADVANCE$DAC$VALUE;

                 /* ADVANCE AND COMPARE THE TWO DISTANCES */
 96   3          DO WHILE PRESENT$POSITION$ADC$STEPS<
                       ABSOLUTE$POSITION$ADC$STEPS;

                       /* READ AGAIN */
 97   4           PRESENT$POSITION$ADC$STEPS=ADC$INPUT;

 98   4          END; /* DO WHILE */

                 /* STOP TOOL */
 99   3          OUTPUT(PORT$22H)=STOP$DAC$VALUE;

100   3      END; /* DO */

101   2      END TOOL$ADVANCE;

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

             /*****************************************************************/
             /* FUNCTION: RETRACT THE ROLLER A GIVEN DISTANCE.                */
             /* PARAMETERS INPUT: RETRACT DISTANCE IN ADC STEPS.              */
             /* OUTPUTS: NONE.                                                */
             /* PROCEDURE: LOCAL.                                             */
             /* CALLS: ADC$INPUT.                                             */
             /* DESCRIPTION:                                                  */
             /*            THE CURRENT TOOL POSITION IS INPUT AND TOOL        */
             /*           RETRACT INITIATED IF REQUIRED. TOOL POSITION IS     */
             /*          CONTINUALLY MONITORED AND TOOL RETRACT TERMINATED    */
             /*         WHEN THE SPECIFIED POSITION IS ACHIEVED.              */
             /*****************************************************************/

102   1      TOOL$RETRACT:
             PROCEDURE(ABSOLUTE$POSITION$ADC$STEPS);
103   2          DECLARE ABSOLUTE$POSITION$ADC$STEPS ADDRESS;
104   2          DECLARE PRESENT$POSITION$ADC$STEPS ADDRESS;
105   2          DECLARE RETRACT$DAC$VALUE LITERALLY '73';
106   2          DECLARE STOP$DAC$VALUE    LITERALLY '128';
107   2          DECLARE PORT$22H          LITERALLY '22H';

             /* READ PRESENT POSITION */
108   2      PRESENT$POSITION$ADC$STEPS=ADC$INPUT;

             /* RETRACT TOOL */
109   2      OUTPUT(PORT$22H)=RETRACT$DAC$VALUE;

             /* RETRACT AND COMPARE THE TWO DISTANCES */
110   2      DO WHILE PRESENT$POSITION$ADC$STEPS>
                     ABSOLUTE$POSITION$ADC$STEPS;

                    /* READ AGAIN */
```

```
111  3          PRESENT$POSITION$ADC$STEPS=ADC$INPUT;

112  3      END; /* DO WHILE */

            /* STOP TOOL */
113  2      OUTPUT(PORT$22H)=STOP$DAC$VALUE;

114  2      END TOOL$RETRACT;

            /*****************************************************************
          -  /

            /*****************************************************************/
            /* FUNCTION: CONE INTERPOLATION                                  */
            /* PARAMETERS INPUT: INTERPOLATION$ENCODER$PULSES(ADDRESS),      */
            /*                   INCREMENTS$ENCODER$PULSES (BYTE).           */
            /* OUTPUTS: CONICAL CONTOUR                                      */
            /* PROCEDURE:                                                    */
            /* CALLS: SHAFT$ENCODER, TOOL$RETRACT                            */
            /* DESCRITPION: A PROCEDURE TO DESCRIBE THE CONICAL CONTOUR.     */
            /*          THE PROCEDURE MOINTORS  THE CARRAIGE POSITION AND    */
            /*    AT PREDETERMINED INTERVALS (8,9,....16 ENCODER PULSES)     */
            /* CALCULATES THE ROLLER POSITION REQUIRED TO MAINTAIN THE       */
            /* CONICAL CONTOUR AND RETRACTS THE ROLLER.                      */
            /*****************************************************************/

115  1      CON$INTERPOLATION:
            PROCEDURE(INTERPOLATION$ENCODER$PULSES);

116  2         DECLARE INTERPOLATION$ENCODER$PULSES ADDRESS;
117  2         DECLARE MOVED$DISTANCE$ENCODER$PULSES ADDRESS;
118  2         DECLARE ABSOLUTE$ADC$STEPS ADDRESS;
119  2         DECLARE Y1$ADC$STEPS ADDRESS;
120  2         DECLARE Y1$TEMP      ADDRESS;

               /* INITIALISE THE MOVED DISTANCE */
121  2         MOVED$DISTANCE$ENCODER$PULSES=0;

            /* CALCULATE AND THEN RETRACT TO THE NEXT ABSOLUTE */
            /*  POSITION AND UPDATE THE CURRENT CARRIAGE       */
            /* POSITION FOR EACH REQUIRED INCREMENT OF CARRIAGE */
            /*  POSITION.                                      */

122  2         DO WHILE MOVED$DISTANCE$ENCODER$PULSES<=
                  INTERPOLATION$ENCODER$PULSES;

               /* UPDATE THE MOVED DISTANCE */
123  3         MOVED$DISTANCE$ENCODER$PULSES=
                                 MOVED$DISTANCE$ENCODER$PULSES
                                +INCREMENTS$ENCODER$PULSES;

124  3          DO WHILE CARRIAGE$POSITION$ENCODER$PULSES<
```

```
                        MOVED$DISTANCE$ENCODER$PULSES;
125   4                 CARRIAGE$POSITION$ENCODER$PULSES=SHAFT$ENCODER;
126   4               END; /* DO WHILE CARRIAGE POSITION */

                  /* CALCULATE THE NEXT RETRACT DISTANCE */
127   3               Y1$ADC$STEPS=(MOVED$DISTANCE$ENCODER$PULSES
                                *TAN$NUM(I))/TAN$DEN(I);
128   3               IF (2*((MOVED$DISTANCE$ENCODER$PULSES*TAN$NUM(I)) MOD
                        TAN$DEN(I)))>=TAN$DEN(I) THEN
129   3               Y1$ADC$STEPS=Y1$ADC$STEPS+1;

                  /* CONVERT FROM PULSES TO ADC$STEPS */
130   3               Y1$TEMP=(Y1$ADC$STEPS*32)/63;
131   3               IF (2*((Y1$ADC$STEPS*32) MOD 63))>=63 THEN
132   3                  Y1$TEMP=Y1$TEMP+1;
133   3             Y1$ADC$STEPS=Y1$TEMP;

                     /* CALCULATE THE NEXT ABSOLUTE POSITION */
134   3               ABSOLUTE$ADC$STEPS=ROLLER$POSITION$ADC$STEPS-Y1$ADC$STEPS
      -   ;

                     /* RETRACT TO THE NEXT ABSOLUTE POSITION */
135   3               CALL TOOL$RETRACT(ABSOLUTE$ADC$STEPS);

136   3             END; /* DO WHILE MOVED DISTANCE */

137   2     END CON$INTERPOLATION;

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

            /*****************************************************************/
            /*                                                               */
            /******      PUBLIC  PROCEDURES  DECLARATIONS            ******/
            /*                                                               */
            /*****************************************************************/

            /*****************************************************************/
            /* FUNCTION: MACHINE$CONE$CONTROL (CONTROL TOOL MOVEMENT)        */
            /* PARAMETERS INPUT: DEFINED IN PREVIOUS PROCEDURES              */
            /* OUTPUTS: DEFINED IN PREVIOUS PROCEDURES                       */
            /* PROCEDURE: PUBLIC                                             */
            /* CALLS: SHAFT$ENCODER, TOOL$ADVANCE, INTERPOLATION,            */
            /*        TOOL$RETRACT, CONSOLE$OUT, MESSAGE.                    */
            /* DESCRIPTION: TOOL MOVEMENT CAN BE SUBDIVIDED INTO THE         */
            /*           FOLLOWING SEGEMENTS:-                               */
            /*                                                               */
            /*  (4 TO 5) ADVANCE TOOL (LATHE IS OFF)                         */
            /*                   OPERATE LATHE, INITIALISE COUNTER           */
            /*  (5 TO 1) MOVEMENT WHILE TOOL IS HELD IN POSITION             */
            /*  (1 TO 2) CONTOUR SHAPE (CONICAL,ECT)                         */
            /*                   SWITCH OFF LATHE                            */
            /*  (2 TO 3) RETRACT TOOL                                        */
```




```
138   1     MACHINE$CONE$CONTROL:
            PROCEDURE PUBLIC;

139   2     DECLARE FORM$LENGTH$ENCODER$PULSES(45)ADDRESS DATA(472,488
                  ,504,519,535,551,567,582,598,614,630,645,661,677,693
                  ,708,724,740,756,771,787,803,819,835,850,866,882,898
                  ,913,929,945,961,976,992,1008,1024,1039,1055,1071
                  ,1087,1102,1118,1134,1150,1165);

140   2     DECLARE INTERPOLATION$ENCODER$PULSES ADDRESS;

141   2     DECLARE PORT$2AH LITERALLY '2AH';
142   2     DECLARE J BYTE; /* GET THE FORM$LENGTH$MM INDEX */

               /* MASKS TO CONTROL LATHE */
143   2     DECLARE PORT$2AH$START$LATHE LITERALLY '0000$0100B';
144   2     DECLARE PORT$2AH$STOP$LATHE  LITERALLY '1111$1011B';

             /* DAC CONTROL VALUES */
145   2     DECLARE RETRACT$DAC$VALUE    LITERALLY '73';
146   2     DECLARE STOP$DAC$VALUE       LITERALLY '128';

147   2     DECLARE PORT$2BH LITERALLY '2BH';
148   2     DECLARE PORT$2BH$INPUT BYTE;
149   2     DECLARE PORT$2BH$ADVANCED LITERALLY '0000$0001B';
150   2     DECLARE PORT$2BH$CARRIAGE LITERALLY '0000$0100B';

151   2     DECLARE OLD$POSITION$ENCODER$PULSES ADDRESS;
152   2     DECLARE NEW$POSITION$ENCODER$PULSES ADDRESS;
153   2     DECLARE COUNT$1 BYTE;
154   2     DECLARE COUNT$2 BYTE;
```

```
                /*****************************************************************/
                /*                                                               */
                /******        CALCULATIONS PRIOR TO THE PROCESS          ******/
                /*                                                               */
                /*****************************************************************/

155   2         I=CONE$ANGLE$DEGREES-30; /* GET THE CONE$ANGLE$DEGREES INDEX */
156   2         J=FORM$LENGTH$MM-30; /* GET THE FORM$LENGTH$MM INDEX */
157   2         INTERPOLATION$ENCODER$PULSES=FORM$LENGTH$ENCODER$PULSES(J);

158   2         TURNING$REQUIRED=TRUE;
159   2         DO WHILE TURNING$REQUIRED;

160   3         CALL CONSOLE$OUT(CLEAR$SCREEN);  /* CLEAR SCREEN */

161   3         CALL MESSAGE(.PAGE$7,LAST(PAGE$7));  /* PAGE 7  */

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

                /*** INHIBIT COUNTER UNTIL INTERPOLATION BEGINS, INITIALISE */
                /***   SET LINE TO TRUE (5V)                                   */

162   3             PORT$2AH$OUTPUT=PORT$2AH$OUTPUT OR COUNTER$RESET; .
163   3             OUTPUT(PORT$2AH)=PORT$2AH$OUTPUT;

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

                /*** INITIALISE SDK-85 KEYBOARD DISPLAY ***/

164   3           KBD$DPLY$CONTROL=KMODE;
165   3           KBD$DPLY$CONTROL=KBNIT;

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

                /*** CLEAR SDK-85 KEYBOARD DISPLAY ***/

166   3           CALL UPDDT(CLEAR$DATA$FIELD);
167   3           CALL UPDAD(CLEAR$ADDRESS$FIELD);

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

                /*** DAC INITIALISE ***/

168   3           OUTPUT(22H)=STOP$DAC$VALUE;

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

                /*** INITIALISE ADDRESS POSITION ***/

169   3         CARRIAGE$POSITION$ENCODER$PULSES=0;

                /*****************************************************************/
```

```
                /*                                                    */
                /****          THE MAIN PROGRAM                    ****/
                /*                                                    */
                /******************************************************/

                /*********************************************************************
              - /

                 /***          ADVANCE TOOL    (POSITION 4-5)              ***/

170   3          CALL TOOL$ADVANCE(ROLLER$POSITION$ADC$STEPS);            /* */
                                                                          /* */
                /*********************************************************************/

                 /***      SWITCH LATHE ON                                ***/

171   3          PORT$2AH$OUTPUT =PORT$2AH$OUTPUT OR PORT$2AH$START$LATHE; /* */
172   3          OUTPUT(PORT$2AH)=PORT$2AH$OUTPUT;                        /* */
                /*********************************************************************/

                 /***  CARRIAGE WILL START MOVING LEFT   (POSITION  5-1)   ***/

                 /* WAIT TILL CARRIAGE SWITCH IS CLOSED */

173   3          PORT$2BH$INPUT=INPUT(PORT$2BH); /* READ PORT */
174   3          DO WHILE (PORT$2BH$INPUT AND PORT$2BH$CARRIAGE)=0;
175   4             PORT$2BH$INPUT=INPUT(PORT$2BH); /* READ AGAIN */
176   4          END; /* DO WHILE */

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

                 /***  ENABLE COUNTER ****/
177   3          PORT$2AH$OUTPUT=PORT$2AH$OUTPUT AND COUNTER$ENABLE;
178   3          OUTPUT(PORT$2AH)=PORT$2AH$OUTPUT;

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

                 /***      CONICAL  CONTOUR   (POSITION  1-2)             ***/

                                                                          /* */
179   3          CALL CON$INTERPOLATION(INTERPOLATION$ENCODER$PULSES);     /* */
                                                                          /* */
                /*********************************************************************/
                                                                          /* */
                 /* SWITCH OFF THE LATHE                                   */
                                                                          /* */
180   3          PORT$2AH$OUTPUT =PORT$2AH$OUTPUT AND PORT$2AH$STOP$LATHE; /* */
181   3          OUTPUT(PORT$2AH)=PORT$2AH$OUTPUT;                        /* */

                /*********************************************************************/
```

```
                /***     RETRACT TOOL  (POSITION 2-3)                ***/
                /***     RETURN TO DATUM                             ***/

182   3     PORT$2BH$INPUT=INPUT(PORT$2BH); /* READ PORT */

            /* ADVANCE PISTON TILL IT REACHES DATUM */
            /* MOVE PISTON TILL IT HITS ADVANCED SWITCH */


183   3     IF (PORT$2BH$INPUT AND PORT$2BH$ADVANCED)=0 THEN
184   3     DO;
185   4        OUTPUT(22H)=RETRACT$DAC$VALUE; /* START ADVANCE TO DATUM */
186   4        PORT$2BH$INPUT=INPUT(PORT$2BH); /* READ PORT */

187   4        DO WHILE (PORT$2BH$INPUT AND PORT$2BH$ADVANCED)=0;
188   5           PORT$2BH$INPUT=INPUT(PORT$2BH); /* READ AGAIN */
189   5        END;   /* DO WHILE */
190   4     END;  /* DO */
191   3     OUTPUT(22H)=STOP$DAC$VALUE;  /* STOP PISTON ADVANCE */

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

            /* A ROUTINE TO DISPLAY THE CARRIAGE POSITION ON THE SDK-85 */
            /*  EVERY 0.5 SECONDS TILL LEADSCREW IS STATIONARY          */

192   3      OLD$POSITION$ENCODER$PULSES=SHAFT$ENCODER;
193   3      COUNT$1=0;
194   3      DO WHILE COUNT$1<10;
195   4       COUNT$2=1;
196   4       DO WHILE COUNT$2<=25;
197   5          CALL TIME(200);
198   5          COUNT$2=COUNT$2+1;
199   5       END; /* DO WHILE */
200   4       NEW$POSITION$ENCODER$PULSES=SHAFT$ENCODER;
201   4       IF NEW$POSITION$ENCODER$PULSES=OLD$POSITION$ENCODER$PULSES THEN
202   4          COUNT$1=COUNT$1+1;
              ELSE
203   4         DO;
204   5           COUNT$1=0;
205   5           OLD$POSITION$ENCODER$PULSES=NEW$POSITION$ENCODER$PULSES;
206   5         END; /* ELSE */
207   4       END; /* DO WHILE */

            /*************************************************************
          - /

208   3       CALL CONSOLE$OUT(CLEAR$SCREEN);  /* CLEAR SCREEN */

209   3       CALL MESSAGE(.PAGE$8,LAST(PAGE$8));  /* PAGE 8 */

210   3       ANSWER=CONSOLE$IN; /* GET REPLY */
211   3       DO WHILE ANSWER<>'Y' AND ANSWER<>'N';
```

```
 212   4           CALL CONSOLE$OUT(BELL);
 213   4           CALL MESSAGE(.ERROR$1,LAST(ERROR$1));
 214   4           ANSWER=CONSOLE$IN;
 215   4         END; /* DO WHILE */
 216   3         CALL MESSAGE(.DELETE$ERROR$1,LAST(DELETE$ERROR$1));
 217   3         CALL MESSAGE(.MESS$1,LAST(MESS$1));
 218   3         CALL CONSOLE$OUT(ANSWER);
 219   3         CALL GET(CARRIAGE$RETURN);
 220   3         IF ANSWER='N' THEN
 221   3           TURNING$REQUIRED=FALSE;
 222   3         END; /* DO WHILE */

 223   2         CALL CONSOLE$OUT(CLEAR$SCREEN);

 224   2       END MACHINE$CONE$$CONTROL;

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

 225   1     END MACHINE$CONE$CONTROL$MODULE;



MODULE INFORMATION:

    CODE AREA SIZE     = 0778H   1912D
    VARIABLE AREA SIZE = 0026H     38D
    MAXIMUM STACK SIZE = 0008H      8D
    685 LINES READ
    0 PROGRAM ERROR(S)

END OF PL/M-80 COMPILATION
```

```
ISIS-II PL/M-80 V3.1 COMPILATION OF MODULE PARAGENERATIONMODULE
OBJECT MODULE PLACED IN :F2:parage.OBJ
COMPILER INVOKED BY:  plm80 :F2:parage.plm DEBUG


                $WORKFILES(:F2:,:F2:)
                $PAGEWIDTH(80)
                $PAGELENGTH(55)

  1             PARA$GENERATION$MODULE:
                DO;
                    /* FILE$NAME  'PARAGE' */

                /**************************************************************/
                /*                                                            */
                /*******       EXTERNAL  VARIABLES  DECLARATIONS        ******/
                /*                                                            */
                /**************************************************************/

                    /* NONE */

                /**************************************************************/
                /*                                                            */
                /*******       PUBLIC  VARIABLES  DECLARATIONS          *******/
                /*                                                            */
                /**************************************************************/

  2   1           DECLARE FORM$LENGTH BYTE PUBLIC;
  3   1           DECLARE ROLLER$POSITION ADDRESS PUBLIC;

                /**************************************************************/
                /*                                                            */
                /******     LOCAL  VARIABLES  DECLARATIONS              ******/
                /*                                                            */
                /**************************************************************/

                    /* ASCII CODE CHARACTERS */

  4   1           DECLARE  ESC    LITERALLY '1BH';
  5   1           DECLARE  QUOTE  LITERALLY '27H';  /* SINGLE QUOTE */
  6   1           DECLARE  DQUOTE LITERALLY '22H';  /* DOUBLE QUOTE */
  7   1           DECLARE  SPACE  LITERALLY '20H';
  8   1           DECLARE  BACK$SPACE LITERALLY '08H';
  9   1           DECLARE  BELL LITERALLY '07H';
 10   1           DECLARE  CLEAR$SCREEN LITERALLY '1AH';

 11   1           DECLARE  ROLLER$ADJUST BYTE;
 12   1           DECLARE  INCREMENTS BYTE;
 13   1           DECLARE  DIRECTION BYTE;

                    /* 592 ADC STEPS IS EQUIVALENT TO 74 MM OF */
```

```
                /*      TRANSDUCER LENGTH                       */

 14   1        DECLARE  FORMER$TIP$ADC$STEPS LITERALLY '592';

 15   1        DECLARE  TEST$POSITION BYTE;

             /**************************************************************/
             /*                                                            */
             /*****      EXTERNAL  PROCEDURES  DECLARATIONS             *****/
             /*                                                            */
             /**************************************************************/

             /**************************************************************/
                                                                    /* */
 16   1          CONSOLE$OUT:                                      /* */
                 PROCEDURE(CHAR) EXTERNAL;                         /* */
 17   2                   DECLARE CHAR BYTE;                       /* */
 18   2          END CONSOLE$OUT;                                  /* */
                                                                    /* */
             /**************************************************************/
                                                                    /* */
 19   1          MESSAGE:                                          /* */
                 PROCEDURE(POINTER,LAST$ELEMENT) EXTERNAL;         /* */
 20   2                   DECLARE POINTER ADDRESS;                 /* */
 21   2                   DECLARE LAST$ELEMENT ADDRESS;            /* */
 22   2          END MESSAGE;                                      /* */
                                                                    /* */
             /**************************************************************/
                                                                    /* */
 23   1          CONSOLE$IN:PROCEDURE BYTE EXTERNAL;               /* */
 24   2          END CONSOLE$IN;                                   /* */
                                                                    /* */
             /**************************************************************/
                                                                    /* */
 25   1          GET:                                              /* */
                 PROCEDURE(TARGET$CHAR) EXTERNAL;                  /* */
 26   2              DECLARE TARGET$CHAR BYTE;                     /* */
 27   2          END GET;                                          /* */
                                                                    /* */
             /**************************************************************/
                                                                    /* */
 28   1          DECIMAL$VALUE$INPUT:                              /* */
                 PROCEDURE BYTE EXTERNAL;                          /* */
 29   2          END DECIMAL$VALUE$INPUT;                          /* */
                                                                    /* */
             /**************************************************************/

             /**************************************************************/
             /*                                                            */
             /*****      LOCAL   PROCEDURES   DECLARATIONS             *****/
             /*                                                            */
             /**************************************************************/
```

```
                /*****************************************************************/
                /* FUNCTION: BACK$DELETE$POSITION                                */
                /* PARAMETERS INPUT: NONE                                        */
                /* OUTPUTS: NONE                                                 */
                /* PROCEDURE:                                                    */
                /* CALLS: CONSOLE$OUT                                            */
                /* DESCRIPTION: A PROCEDURE TO BACKSPACE TWO POSITIONS, DELETE   */
                /*           TWO CHARACTERS FROM SCREEN, THEN BACKSPACE TWO      */
                /*         POSITIONS, WHERE IT WAS INITIALLY. THIS PROCEDURE     */
                /*       IS THEN REQUESTS OPERATOR TO ENTER THE CORRECT NUMBER   */
                /*    AFTER A NUMBER NOT WITH IN THE SHOWN RANGE HAS BEEN TYPED  */
                /*  IN.                                                          */
                /*****************************************************************/
                                                                        /* */
30   1            BACK$DELETE$POSITION:                                 /* */
                  PROCEDURE;                                          /* */
31   2              CALL  CONSOLE$OUT(BACK$SPACE);                     /* */
32   2              CALL  CONSOLE$OUT(BACK$SPACE);                     /* */
33   2              CALL  CONSOLE$OUT(SPACE);                          /* */
34   2              CALL  CONSOLE$OUT(SPACE);                          /* */
35   2              CALL  CONSOLE$OUT(BACK$SPACE);                     /* */
36   2              CALL  CONSOLE$OUT(BACK$SPACE);                     /* */
37   2            END BACK$DELETE$POSITION;                            /* */
                                                                        /* */
                /*****************************************************************/

                /*****************************************************************/
                /*                                                               */
                /*****       PUBLIC   PROCEDURES   DECLARATIONS            *****/
                /*                                                               */
                /*****************************************************************/

                /*****************************************************************/
                /* FUNCTION: PARA$GENERATION                                     */
                /* PARAMETERS INPUT: PARABOLA PARAMETERS FROM KEYBOARD           */
                /* OUTPUTS: NONE                                                 */
                /* PROCEDURE: PUBLIC                                             */
                /* CALLS: MESSAGE, DECIMAL$VALUE$INPUT, BACK$DELETE$POSITION,    */
                /*        CONSOLE$OUT                                            */
                /* DESCRIPTION: A PROCEDURE TO DISPLAY PARABOLA PARAMETES ON     */
                /*           SCREEN, THEN OPERATOR IS REQUIRED TO FILL IN THE    */
                /*         VARIABLES WITH APROPRIATE VALUES SELECTED FROM EACH   */
                /*      VARIABLE LIMITS SHOWN ON THE SCREEN. FAILING TO DO SO    */
                /*   WILL RING A BELL AND OPERATOR IS REQUESTED TO ENTER THE     */
                /* RIGHT VALUE AGAIN.                                            */
                /*****************************************************************/

38   1        PARABOLA$GENERATION:
                PROCEDURE PUBLIC;

                   /* PARABOLA CONTOUR PARAMETERS DISPLAY */
```

```
 39  2     DECLARE PAGE$6(*) BYTE DATA
           (ESC,'=','!','7',                    /* R2 C24 */
           'PARABOLA CONTOUR PARAMETERS',
           ESC,'=',DQUOTE,'7',                  /* R3 C24 */
           '--------------------------',
           ESC,'=','%',')',                     /* R6 C10 */

           'Form Length (mm)',
           ESC,'=','&',')',                     /* R7 C10 */
           '(30 to 74)',

           ESC,'=',QUOTE,',',       /*       R8 C13       */
           ESC,'j',                 /* START REVERSE VIDEO */
           ESC,'=',QUOTE,'/',       /*       R8 C16       */
           ESC,'k',                 /*  END REVERSE VIDEO  */

           ESC,'=','*','M',                     /* R11 C46 */
           '2',
           ESC,'=','+','8',                     /* R12 C25 */
           'Parabola equation   y= 4a*x',
           ESC,'=',',','A',                     /* R13 C34 */
           'and with   a= 68.45',

           ESC,'=','6','&',                 /* R23 C7 */
           'After each parameter press RETURN please.',
           ESC,'=','7','!',                 /* R24 C2 */
           'If incorrect value is entered, use DEL key to erase before RETURN
       -   .');

 40  2     DECLARE NOT$1(*) BYTE DATA(
           ESC,'=','7','&',                 /* R24 C7 */
           'Press SP to proceed please');

 41  2     DECLARE ERR$1(*) BYTE DATA
           (ESC,'=','5','&',                  /* R22 C7 */
           ESC,'^',   /* BLINKING CHARACTER */
           'Parameter not within limits, reenter please.',
           ESC,'q');  /* END BLINKING */

 42  2     DECLARE ERR$2(*) BYTE DATA
           (ESC,'=','5','&',              /* R22 C7 */
           ESC,'^',       /* BLINKING CHARACTER */
           'Enter either ''Y'' or ''N'' please.',
           ESC,'T');  /* ERASE TO END OF LINE */

 43  2     DECLARE ERR$3(*) BYTE DATA
           (ESC,'=','5','&',              /* R22 C7 */
           ESC,'^',       /* BLINKING CHARACTER */
           'Enter either ''F'' or ''R'' please.',
           ESC,'T');  /* ERASE TO END OF LINE */
```

```
44  2     DECLARE DEL$ERR(*) BYTE DATA
          (ESC,'=','5','&',              /* R22 C7 */
          ESC,'T');

45  2     DECLARE ADJ$1(*) BYTE DATA(
          ESC,'=','/',QUOTE,             /* R16 C8 */
          'Do you require roller adjustment ? (Y/N)');

46  2     DECLARE ADJ$2(*) BYTE DATA(
          ESC,'=','1',QUOTE,             /* R18 C8 */
          'How many increments ? (1-80) (1 increment=1/8 mm)');

47  2     DECLARE ADJ$3(*) BYTE DATA(
          ESC,'=','3',QUOTE,             /* R20 C8 */
          'Forward or reverse ? (F/R)');

48  2     DECLARE POS$1(*) BYTE DATA(
          ESC,'=',QUOTE,'-');            /* R8 C14 */

49  2     DECLARE POS$2(*) BYTE DATA(
          ESC,'=','0',')');              /* R17 C10 */

50  2     DECLARE POS$3(*) BYTE DATA(
          ESC,'=','2',')');              /* R19 C10 */

51  2     DECLARE POS$4(*) BYTE DATA(
          ESC,'=','4',')');              /* R21 C10 */

52  2     DECLARE POS$5(*) BYTE DATA(
          ESC,'=','1','&');              /* R18 C7  */

53  2     DECLARE TEST$1(*) BYTE DATA(
          ESC,'=','0','&',               /* R17 C7  */
          'Do you require the roller test position (Y/N) ?');

54  2     DECLARE DEL$T(*) BYTE DATA(
          ESC,'=','0','&',               /* R17 C7 */
          ESC,'T',
          ESC,'=','1','&',               /* R18 C7 */
          ESC,'T',
          ESC,'=','7',' ',               /* R24 C1 */
          ESC,'T');

55  2     DECLARE DEL$F(*) BYTE DATA(
          ESC,'=','6',' ',               /* R23 C1 */
          ESC,'T',
          ESC,'=','7',' ',               /* R24 C1 */
          ESC,'T');

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

56  2           CALL CONSOLE$OUT(CLEAR$SCREEN);
```

```
                /*************************************************************/

 57   2             CALL    MESSAGE(.PAGE$6,LAST(PAGE$6));

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

                /*       PARABOLA PARAMETERS INPUT                           */

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

                   /* INPUT FORM$LENGTH$MM */

 58   2             CALL MESSAGE(.POS$1,LAST(POS$1));
 59   2             FORM$LENGTH=DECIMAL$VALUE$INPUT; /* GET VALUE */
 60   2             DO WHILE  FORM$LENGTH<30 OR FORM$LENGTH>74; /* CHECK */
 61   3                 CALL  BACK$DELETE$POSITION;
 62   3                 CALL  MESSAGE(.ERR$1,LAST(ERR$1)); /* ERROR MESSAGE */
 63   3                 CALL  CONSOLE$OUT(BELL); /* RING A BELL */
 64   3                 CALL  MESSAGE(.POS$1,LAST(POS$1)); /* REPOSITION */
 65   3                 FORM$LENGTH=DECIMAL$VALUE$INPUT; /* GET NEW VALUE */
 66   3             END; /* WHILE */
 67   2             CALL    MESSAGE(.DEL$ERR,LAST(DEL$ERR));

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

 68   2              CALL  MESSAGE(.DEL$F,LAST(DEL$F));  /* DELETE FOOTNOTE */

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

                   /* ROLLER TEST POSITION REQUIRED ? */

 69   2             CALL MESSAGE(.TEST$1,LAST(TEST$1));
 70   2             TEST$POSITION=CONSOLE$IN;
 71   2             DO WHILE TEST$POSITION<>'Y' AND TEST$POSITION<>'N';
 72   3                CALL  CONSOLE$OUT(BELL);
 73   3                CALL  MESSAGE(.ERR$2,LAST(ERR$2));
 74   3                TEST$POSITION=CONSOLE$IN;
 75   3             END; /* DO WHILE */
 76   2             CALL MESSAGE(.DEL$ERR,LAST(DEL$ERR));
 77   2             IF TEST$POSITION='N' THEN
 78   2                DO;
 79   3                   CALL MESSAGE(.POS$5,LAST(POS$5));
 80   3                   CALL CONSOLE$OUT(TEST$POSITION);
 81   3                   CALL MESSAGE(.NOT$1,LAST(NOT$1));
 82   3                   CALL GET(' ');
 83   3                   CALL MESSAGE(.DEL$T,LAST(DEL$T));

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

                   /* ROLLER ADJUSTMENT REQUIRED ? */
```

```
 84   3          CALL MESSAGE(.ADJ$1,LAST(ADJ$1));
 85   3          ROLLER$ADJUST=CONSOLE$IN;
 86   3          DO WHILE ROLLER$ADJUST<>'Y' AND ROLLER$ADJUST<>'N';
 87   4              CALL  CONSOLE$OUT(BELL);
 88   4              CALL  MESSAGE(.ERR$2,LAST(ERR$2));
 89   4              ROLLER$ADJUST=CONSOLE$IN;
 90   4          END;
 91   3          CALL MESSAGE(.DEL$ERR,LAST(DEL$ERR));
 92   3          CALL MESSAGE(.POS$2,LAST(POS$2));
 93   3          CALL CONSOLE$OUT(ROLLER$ADJUST);

 94   3        IF ROLLER$ADJUST='Y' THEN
 95   3          DO;

                  /* INPUT ROLLER ADJUSTMENT */

 96   4              CALL MESSAGE(.ADJ$2,LAST(ADJ$2));
 97   4              CALL MESSAGE(.POS$3,LAST(POS$3));
 98   4              INCREMENTS=DECIMAL$VALUE$INPUT;
 99   4              DO WHILE INCREMENTS<1 OR INCREMENTS>80;
100   5                  CALL BACK$DELETE$POSITION;
101   5                  CALL MESSAGE(.ERR$1,LAST(ERR$1));
102   5                  CALL CONSOLE$OUT(BELL);
103   5                  CALL MESSAGE(.POS$3,LAST(POS$3));
104   5                  INCREMENTS=DECIMAL$VALUE$INPUT;
105   5              END;
106   4              CALL MESSAGE(.DEL$ERR,LAST(DEL$ERR));

                  /* INPUT ADJUSTMENT DIRECTION */

107   4          CALL MESSAGE(.ADJ$3,LAST(ADJ$3));
108   4          DIRECTION=CONSOLE$IN;
109   4          DO WHILE DIRECTION<>'F' AND DIRECTION<>'R';
110   5              CALL CONSOLE$OUT(BELL);
111   5              CALL MESSAGE(.ERR$3,LAST(ERR$3));
112   5              DIRECTION=CONSOLE$IN;
113   5          END;
114   4          CALL MESSAGE(.DEL$ERR,LAST(DEL$ERR));
115   4          CALL MESSAGE(.POS$4,LAST(POS$4));
116   4          CALL CONSOLE$OUT(DIRECTION);

117   4          IF DIRECTION='F' THEN
118   4          ROLLER$POSITION=FORMER$TIP$ADC$STEPS-INCREMENTS;

                  ELSE
119   4          ROLLER$POSITION=FORMER$TIP$ADC$STEPS+INCREMENTS;

120   4        END; /* END IF ROLLER$ADJUST= 'Y' */

                  ELSE
121   3              ROLLER$POSITION=FORMER$TIP$ADC$STEPS;
122   3        CALL MESSAGE(.NOT$1,LAST(NOT$1));
```

```
123   3         CALL GET(' ');
124   3         CALL CONSOLE$OUT(CLEAR$SCREEN);

125   3         END;  /* IF TEST$POSITION='N' */
                ELSE  /* IF TEST$POSITION='Y' THEN */
126   2          DO;
127   3            CALL MESSAGE(.POS$5,LAST(POS$5));
128   3            CALL CONSOLE$OUT(TEST$POSITION);
129   3            CALL MESSAGE(.NOT$1,LAST(NOT$1));
130   3            CALL GET(' ');

                   /* 552 ADC STEPS IS EQUIVALENT TO 69 MM OF */
                   /*          TRANSDUCER LENGTH              */

131   3            ROLLER$POSITION=552;
132   3          END;

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

133   2       END PARABOLA$GENERATION;

134   1       END PARA$GENERATION$MODULE;



MODULE INFORMATION:

     CODE AREA SIZE     = 0516H   1302D
     VARIABLE AREA SIZE = 0007H      7D
     MAXIMUM STACK SIZE = 0004H      4D
     379 LINES READ
     0 PROGRAM ERROR(S)

END OF PL/M-80 COMPILATION
```

```
ISIS-II PL/M-80 V3.1 COMPILATION OF MODULE MACHINEPARABOLACONTROLMODULE
OBJECT MODULE PLACED IN :F2:paramc.OBJ
COMPILER INVOKED BY:  plm80 :F2:paramc.plm DEBUG


            $WORKFILES(:F2:,:F2:)
            $PAGEWIDTH(80)
            $PAGELENGTH(55)

  1         MACHINE$PARABOLA$CONTROL$MODULE:
            DO;
               /* FILE$NAME  'PARAMC' */

            /*****************************************************************/
            /*                                                               */
            /******      EXTERNAL  VARIABLES  DECLARATIONS          ******/
            /*                                                               */
            /*****************************************************************/

  2   1        DECLARE FORM$LENGTH BYTE EXTERNAL;
  3   1        DECLARE ROLLER$POSITION ADDRESS EXTERNAL;
  4   1        DECLARE PORT$2AH$OUTPUT BYTE EXTERNAL;

            /*****************************************************************/
            /*                                                               */
            /******      PUBLIC  VARIABLES  DECLARATIONS            ******/
            /*                                                               */
            /*****************************************************************/

               /* NONE */

            /*****************************************************************/
            /*                                                               */
            /******      LOCAL  VARIABLES  DECLARATIONS             ******/
            /*                                                               */
            /*****************************************************************/

  5   1     DECLARE CARRIAGE$POSITION$ENCODER$PULSES ADDRESS;

             /* ASCII CODE CHARACTERS */

  6   1     DECLARE ESC LITERALLY '1BH';
  7   1     DECLARE QUOTE LITERALLY '27H';
  8   1     DECLARE CLEAR$SCREEN LITERALLY '1AH';
  9   1     DECLARE BELL LITERALLY '07H';
 10   1     DECLARE CARRIAGE$RETURN LITERALLY '0DH';

 11   1     DECLARE TURNING$REQUIRED BYTE;
 12   1     DECLARE TRUE LITERALLY '0FFH';
 13   1     DECLARE FALSE LITERALLY '0';
 14   1     DECLARE ANSWER BYTE;
```

```
 15  1     DECLARE COUNTER$RESET LITERALLY '0000$0001B';
 16  1     DECLARE COUNTER$ENABLE LITERALLY '1111$1110B';

 17  1     DECLARE CLEAR$DATA$FIELD LITERALLY '0';
 18  1     DECLARE CLEAR$ADDRESS$FIELD LITERALLY '0';

 19  1     DECLARE KBD$DPLY$CONTROL BYTE AT (1900H);
 20  1     DECLARE KMODE LITERALLY '0';
 21  1     DECLARE KBNIT LITERALLY '0CCH';

 22  1     DECLARE PAGE$9(*) BYTE DATA
             (ESC,'=',' ','8',                    /* R1  C25 */
             'Flow-turning roller movements',
              ESC,'=','!','8',                    /* R2  C25 */
             '----------------------------',
              ESC,'=','£','$',                    /* R4  C5  */
             '1- Roller advancing (4-5)',
              ESC,'=','$','$',                    /* R5  C5  */
             '2- Lathe switched on, carriage moving left (5-1)',
              ESC,'=','%','$',                    /* R6  C5  */
             '3- Parabolic contour path (1-2)',
              ESC,'=','&','$',                    /* R7  C5  */
             '4- Lathe switched off, roller retracted (2-3)',
              ESC,'=',QUOTE,'$',                  /* R8  C5  */
             '5- Manual movement to datum (3-4)',
              ESC,'=',')','=',                    /* R10 C30 */
             '1                              5',
              ESC,'=','+','?',                    /* R12 C32 */
             '++++++++++++++++++++++++++++',
              ESC,'=',',',';',                    /* R13 C28 */
             '+                             +',
              ESC,'=','-','8',                    /* R14 C25 */
             '+                            +',
              ESC,'=','.','5',                    /* R15 C22 */
             '+                             +',
              ESC,'=','/','3',                    /* R16 C20 */
             '2 +                             +',
              ESC,'=','0','3',                    /* R17 C20 */
             '+                               +',
              ESC,'=','1','3',                    /* R18 C20 */
             '+                               +',
              ESC,'=','2','3',                    /* R19 C20 */
             '+++++++++++++++++++++++++++++++++++++++',
              ESC,'=','4','3',                    /* R21 C20 */
             '3                                 4  DATUM ');

 23  1     DECLARE PAGE$10(*) BYTE DATA(
             ESC,'=','%',':',                     /* R6  C27 */
             'Flow-turning is completed',
             ESC,'=','&',':',                     /* R7  C27 */
             '------------------------',
```

```
                ESC,'=',')','$',                        /* R10 C5  */
                '1- Please remove the finished component.',
                ESC,'=','+','$',                        /* R12 C5  */
                '2- Disengage the carriage and return to datum by
                 the handwheel.',
                ESC,'=','3','$',                        /* R20 C5  */
                'Another identical parabola ?  (Y/N) then RETURN');
 24   1         DECLARE MESS$1(*) BYTE DATA(
                ESC,'=','5','$');                       /* R22 C5  */
 25   1         DECLARE ERROR$1(*) BYTE DATA(
                ESC,'^',   /* BLINKING STARTS  */
                ESC,'=','7','$',                        /* R24 C5  */
                'Enter either Y OR N please');
 26   1         DECLARE DELETE$ERROR$1(*) BYTE DATA(
                ESC,'=','7','$',                        /* R24 C5  */
                ESC,'T');  /* ERASE TO END OF LINE */

                /*****************************************************************/
                /*                                                               */
                /******      EXTERNAL  PROCEDURES  DECLARATIONS          ******/
                /*                                                               */
                /*****************************************************************/

 27   1            UPDDT:
                   PROCEDURE(PAR$1) EXTERNAL;
 28   2              DECLARE PAR$1 BYTE;
 29   2           END UPDDT;

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

 30   1            UPDAD:
                   PROCEDURE(PAR$2) EXTERNAL;
 31   2              DECLARE PAR$2 ADDRESS;
 32   2           END UPDAD;

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

 33   1            CONSOLE$OUT:
                   PROCEDURE(CHAR) EXTERNAL;
 34   2              DECLARE CHAR BYTE;
 35   2           END CONSOLE$OUT;

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

 36   1            MESSAGE:
                   PROCEDURE(POINTER,LAST$ELEMENT) EXTERNAL;
 37   2               DECLARE POINTER ADDRESS;
 38   2               DECLARE LAST$ELEMENT ADDRESS;
 39   2           END MESSAGE;

                /*****************************************************************/
```

```
 40  1          CONSOLE$IN:PROCEDURE BYTE EXTERNAL;
 41  2          END CONSOLE$IN;

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

 42  1          GET:
                PROCEDURE(TARGET$CHAR) EXTERNAL;
 43  2              DECLARE TARGET$CHAR BYTE;
 44  2          END GET;

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

             /*************************************************************/
             /*                                                           */
             /*******      LOCAL  PROCEDURES  DECLARATIONS            *******/
             /*                                                           */
             /*************************************************************/

             /*************************************************************/
             /* FUNCTION: SHAFT$ENCODER (ENCODER 8-BIT INPUT)             */
             /* PARAMETERS INPUT: NONE                                    */
             /* OUTPUTS: 16-BIT VALUE IN (.CARRIAGE$POSITION$ENCODER$PULSES) */
             /* PROCEDURE: LOCAL                                          */
             /* CALLS: NOTHING                                            */
             /* DESCRIPTION: THE ACCUMULATIVE VALUE OF THE COUNTER WILL BE  */
             /*            STORED IN TWO BYTES(16-BIT), THE HIGH AND THE  */
             /*          LOW. PORT$29H WILL BE READ, THE LOW BYTE WILL ALL */
             /*        THE TIME BE READ AND UPDATED, IF THE VALUE EXCEEDS  */
             /*     256 THEN 1 WILL BE ADDED TO THE HIGH BYTE. PULSE.HIGH  */
             /*   AND PULSE.LOW ARE STORED IN MEMORY WHERE THEY ARE REFERED */
             /* TO BY THE GLOBAL VARIABLE CARRIAGE$POSITION$ENCODER$PULSES. */
             /*************************************************************/
                                                                  /* */
 45  1       SHAFT$ENCODER:                                       /* */
             PROCEDURE ADDRESS;                                   /* */
 46  2          DECLARE  PULSE STRUCTURE(LOW BYTE,HIGH BYTE) AT  /* */
                          (.CARRIAGE$POSITION$ENCODER$PULSES);    /* */
 47  2          DECLARE READ$COUNT BYTE;                         /* */
 48  2          DECLARE NO$OF$READS LITERALLY '10';              /* */
 49  2          DECLARE  COUNTER BYTE;                           /* */
 50  2          DECLARE  COUNTER$1 BYTE;                         /* */
 51  2          DECLARE  PORT$29H LITERALLY '29H';               /* */
                                                                  /* */
             /* THE CODE DEBOUNCES COUNTER BY SOFTWARE */

 52  2       RETRY:
                READ$COUNT=NO$OF$READS;
 53  2          COUNTER=INPUT(PORT$29H);
 54  2          DO WHILE READ$COUNT>0;
 55  3            COUNTER$1=INPUT(PORT$29H);
 56  3            IF COUNTER<>COUNTER$1 THEN GOTO RETRY;
 58  3            READ$COUNT=READ$COUNT-1;
```

```
 59  3          END; /* WHILE */

 60  2          IF COUNTER<PULSE.LOW  THEN                          /* */
 61  2             PULSE.HIGH=PULSE.HIGH+1;                         /* */
 62  2          PULSE.LOW=COUNTER;                                  /* */

                /* DISPLAY COUNTER'S VALUE ON SDK-85 LED'S */
 63  2          CALL UPDDT(COUNTER);
 64  2          CALL UPDAD(CARRIAGE$POSITION$ENCODER$PULSES);

 65  2          RETURN CARRIAGE$POSITION$ENCODER$PULSES;            /* */
 66  2       END SHAFT$ENCODER;                                     /* */
                                                                    /* */
               /*****************************************************************/


               /*****************************************************************/
               /* FUNCTION: ADC$INPUT (12-BIT ADC)                              */
               /* PARAMETERS INPUT: ANALOGUE VOLTAGE (0-10) VOLTS               */
               /* OUTPUTS: DIGITAL VALUES (0 - 1023)                            */
               /* PROCEDURE: LOCAL                                              */
               /* CALLS: TIME                                                   */
               /* DESCRIPTION: OUTPUT PULSE LOW (00) AND THEN OUTPUT            */
               /*            PULSE HIGH (02) VIA PORT$2AH BIT$1 TO START        */
               /*          CONVERSION, WAITS TILL CONVERSION IS COMPLETED,      */
               /*        THEN INPUTS THE VALUES FROM PORTS                      */
               /*            21H (8-BIT)       8 LSB'S                        */
               /*            23H (9-12 BIT)    4 MSB'S                        */
               /*****************************************************************/
                                                                    /* */
 67  1       ADC$INPUT:                                             /* */
               PROCEDURE ADDRESS;                                   /* */
 68  2          DECLARE ADC$CONVERT LITERALLY '1111$1101B';        /* */
 69  2          DECLARE ADC$IDLE    LITERALLY '0000$0010B';        /* */
 70  2          DECLARE PORT$21H LITERALLY '21H'; /* ADC 1-8 BIT      */
 71  2          DECLARE PORT$23H LITERALLY '23H'; /* ADC 9-12 BIT     */
 72  2          DECLARE PORT$2AH LITERALLY '2AH'; /* START CONVERSION */
 73  2          DECLARE ADC$IN$WORD ADDRESS;                       /* */

 74  2          OUTPUT(PORT$2AH)=PORT$2AH$OUTPUT AND ADC$CONVERT;  /* */
 75  2          OUTPUT(PORT$2AH)=PORT$2AH$OUTPUT OR  ADC$IDLE;     /* */
 76  2          CALL TIME(1); /* A DELAY OF 100 MICROSECONDS UNTIL     */
                           /*     CONVERSION IS COMPLETED           */

               /* READ PORT 23H, MASK OFF 4 MSB'S, DOUBLE IT(i.e ADD 8 ZERO */
               /*      TO THE LEFT) AND THEN ROTATE LEFT 8 DIGITS.          */

 77  2          ADC$IN$WORD=SHL(DOUBLE(INPUT(PORT$23H) AND 0000$1111B),8);/* */

               /* ADD THIS VALUE TO THE INPUT OF PORT$21H AND THEN ROTATE   */
               /*      RIGHT 2 DIGITS (TO RID OFF 2 LSB'S).                 */
```

```
 78  2       ADC$IN$WORD=SHR((ADC$IN$WORD+INPUT(PORT$21H)),2);        /* */

 79  2       RETURN  ADC$IN$WORD; /* RETURN 10-BIT VALUE (POSITION)      */
 80  2     END ADC$INPUT;                                             /* */
                                                                      /* */
           /*********************************************************************/


           /*********************************************************************/
           /* FUNCTION: ADVANCE THE ROLLER A GIVEN DISTANCE.                    */
           /* PARAMETERS INPUT: ADVANCE DISTANCE IN ADC STEPS.                  */
           /* OUTPUTS:NONE.                                                     */
           /* PROCEDURE: LOCAL                                                  */
           /* CALLS: ADC$INPUT.                                                 */
           /* DESCRIPTION:                                                      */
           /*            THE CURRENT TOOL POSITION IS INPUT AND TOOL            */
           /*           ADVANCE INITIATED IF REQUIRED. TOOL POSITION IS         */
           /*          CONTINUALLY MONITORED AND TOOL ADVANCE TERMINATED        */
           /*         WHEN THE SPECIFIED POSITION IS ACHIEVED.                  */
           /*********************************************************************/

 81  1     TOOL$ADVANCE:
           PROCEDURE(ABSOLUTE$POSITION$ADC$STEPS);
 82  2        DECLARE ABSOLUTE$POSITION$ADC$STEPS ADDRESS;
 83  2        DECLARE PRESENT$POSITION$ADC$STEPS ADDRESS;
 84  2        DECLARE ADVANCE$DAC$VALUE LITERALLY '183';
 85  2        DECLARE STOP$DAC$VALUE    LITERALLY '128';
 86  2        DECLARE PORT$22H          LITERALLY '22H';

           /* IF ABSOLUTE POSITION IS LESS THAN THE MAXIMUM VALUE */
           /*       (03FFH), GO THROUGH THE ROUTINE.              */

 87  2     IF ABSOLUTE$POSITION$ADC$STEPS<03FFH  THEN
 88  2     DO;

             /* READ PRESENT POSITION */
 89  3        PRESENT$POSITION$ADC$STEPS=ADC$INPUT;

             /* ADVANCE TOOL */
 90  3        OUTPUT(PORT$22H)=ADVANCE$DAC$VALUE;

             /* ADVANCE AND COMPARE THE TWO DISTANCES */
 91  3        DO WHILE PRESENT$POSITION$ADC$STEPS<
                  ABSOLUTE$POSITION$ADC$STEPS;

                  /* READ AGAIN */
 92  4         PRESENT$POSITION$ADC$STEPS=ADC$INPUT;

 93  4        END; /* DO WHILE */

             /* STOP TOOL */
 94  3        OUTPUT(PORT$22H)=STOP$DAC$VALUE;
```

```
 95   3      END; /* DO */

 96   2      END TOOL$ADVANCE;

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

             /*****************************************************************/
             /* FUNCTION: RETRACT THE ROLLER A GIVEN DISTANCE.                */
             /* PARAMETERS INPUT: RETRACT DISTANCE IN ADC STEPS.              */
             /* OUTPUTS: NONE.                                                */
             /* PROCEDURE: LOCAL.                                             */
             /* CALLS: ADC$INPUT.                                             */
             /* DESCRIPTION:                                                  */
             /*            THE CURRENT TOOL POSITION IS INPUT AND TOOL        */
             /*          RETRACT INITIATED IF REQUIRED. TOOL POSITION IS      */
             /*         CONTINUALLY MONITORED AND TOOL RETRACT TERMINATED     */
             /*        WHEN THE SPECIFIED POSITION IS ACHIEVED.               */
             /*****************************************************************/

 97   1      TOOL$RETRACT:
             PROCEDURE(ABSOLUTE$POSITION$ADC$STEPS);
 98   2          DECLARE ABSOLUTE$POSITION$ADC$STEPS ADDRESS;
 99   2          DECLARE PRESENT$POSITION$ADC$STEPS ADDRESS;
100   2          DECLARE RETRACT$DAC$VALUE LITERALLY '73';
101   2          DECLARE STOP$DAC$VALUE    LITERALLY '128';
102   2          DECLARE PORT$22H          LITERALLY '22H';

             /* READ PRESENT POSITION */
103   2      PRESENT$POSITION$ADC$STEPS=ADC$INPUT;

             /* RETRACT TOOL */
104   2      OUTPUT(PORT$22H)=RETRACT$DAC$VALUE;

             /* RETRACT AND COMPARE THE TWO DISTANCES */
105   2      DO WHILE PRESENT$POSITION$ADC$STEPS>
                     ABSOLUTE$POSITION$ADC$STEPS;

                    /* READ AGAIN */
106   3          PRESENT$POSITION$ADC$STEPS=ADC$INPUT;

107   3      END; /* DO WHILE */

             /* STOP TOOL */
108   2      OUTPUT(PORT$22H)=STOP$DAC$VALUE;

109   2      END TOOL$RETRACT;

             /*****************************************************************
           - /

             /*****************************************************************/
```

```
            /* FUNCTION: PARABOLA INTERPOLATION
          -  */
            /* PARAMETERS INPUT: INTERPOLATION$ENCODER$PULSES(ADDRESS)     */
            /* OUTPUTS: PARABOLIC CONTOUR                                  */
            /* PROCEDURE:                                                  */
            /* CALLS: SHAFT$ENCODER, TOOL$RETRACT                          */
            /* DESCRITPION: A PROCEDURE TO DESCRIBE THE PARABOLA CONTOUR.  */
            /*          THE PROCEDURE MONITORS THE CARRIAGE POSITION AND   */
            /*    AT INTERVALS OF 16 ENCODER PULSES EXTRACTS THE ROLLER    */
            /* POSITION REQUIRED TO MAINTAIN THE PARABOLIC CONTOUR FROM THE */
            /* LOOK-UP TABLE AND RETRACTS THE ROLLER TO THAT POSITION.     */
            /*************************************************************/

110   1     PAR$INTERPOLATION:
            PROCEDURE(INTERPOLATION$ENCODER$PULSES);

111   2       DECLARE INTERPOLATION$ENCODER$PULSES ADDRESS;
112   2       DECLARE MOVED$DISTANCE$ENCODER$PULSES ADDRESS;
113   2       DECLARE ABSOLUTE$ADC$STEPS ADDRESS;
114   2       DECLARE Y1$ADC$STEPS ADDRESS;
115   2       DECLARE ADC$RESOLUTION LITERALLY '8';
116   2       DECLARE J BYTE;  /* PARABOLA RADUIS, NUMERATOR AND */
                               /*      DENOMENATOR INDEX         */
117   2       DECLARE INCREMENTS$ENCODER$PULSES LITERALLY '16' ;

118   2       DECLARE PARA$NUM(72) ADDRESS DATA(131,524,651,45,
                  961,704,806,90,528,961,937,405,295,852,123,360,
                  961,193,377,656,685,177,355,810,205,813,863,869,
                  929,492,471,637,583,292,956,386,160,969,367,187,
                  995,685,718,708,710,710,533,443,869,820,961,367,
                  646,1000,479,934,49,799,958,95,491,942,823,803,
                  223,739,897,523,359,351,19,899);

119   2       DECLARE PARA$DEN(72) ADDRESS DATA(34747,34747,
                  19186,746,10196,5187,4363,373,1729,2549,2054,
                  746,463,1153,145,373,882,158,277,435,412,97,178,
                  373,87,319,314,294,293,145,130,165,142,67,207,
                  79,31,178,64,31,157,103,103,97,93,89,64,51,96,
                  87,98,36,61,91,42,79,4,63,73,7,35,65,55,52,14,
                  45,53,30,20,19,1,46);

                /* VALUES OF THE RADIUS COMPENSATION ALONG THE */
                /*            PARABOLIC CONTOUR (ADC STEPS)   */

120   2       DECLARE RAD$COM(72) BYTE DATA(0,0,0,0,0,0,0,0,0,0,
                  0,0,0,0,0,0,0,0,0,0,0,0,0,0,
                  1,1,1,1,1,1,1,1,1,1,
                  1,1,2,2,2,2,2,2,3,3,3,3,3,3,3,
                  4,4,4,4,4,4,5,5,5,5,5,5,
                  6,6,6,6,6,7,7,7,7,7,7);

                    /* INITIALISE THE MOVED DISTANCE */
```

```
121  2            MOVED$DISTANCE$ENCODER$PULSES=0;
122  2                                    J=0;

                /* CALCULATE AND THEN RETRACT TO THE NEXT ABSOLUTE */
                /*  POSITION AND UPDATE THE CURRENT CARRIAGE       */
                /* POSITION FOR EACH REQUIRED INCREMENT OF CARRIAGE */
                /*  POSITION.                                      */

123  2         DO WHILE MOVED$DISTANCE$ENCODER$PULSES<=
                   INTERPOLATION$ENCODER$PULSES;

                    /* UPDATE THE MOVED DISTANCE */
124  3              MOVED$DISTANCE$ENCODER$PULSES=
                                    MOVED$DISTANCE$ENCODER$PULSES
                                   +INCREMENTS$ENCODER$PULSES;

125  3           DO WHILE CARRIAGE$POSITION$ENCODER$PULSES<
                        MOVED$DISTANCE$ENCODER$PULSES;
126  4             CARRIAGE$POSITION$ENCODER$PULSES=SHAFT$ENCODER;
127  4           END; /* DO WHILE CARRIAGE POSITION */

                /* CONVERT FROM MILLIMETERS TO ADC$STEPS */
128  3          Y1$ADC$STEPS=PARA$NUM(J)*ADC$RESOLUTION/PARA$DEN(J);

129  3          IF (2*((PARA$NUM(J)*ADC$RESOLUTION) MOD PARA$DEN(J)))
                                >=PARA$DEN(J) THEN
130  3          Y1$ADC$STEPS=Y1$ADC$STEPS+1;

                    /* CALCULATE THE NEXT ABSOLUTE POSITION */
131  3              ABSOLUTE$ADC$STEPS=ROLLER$POSITION-Y1$ADC$STEPS+RAD$COM(J
       -  );

                    /* RETRACT TO THE NEXT ABSOLUTE POSITION */
132  3              CALL TOOL$RETRACT(ABSOLUTE$ADC$STEPS);

133  3          J=J+1;  /* UPDATE THE PARABOLA INDEX */

134  3           END; /* DO WHILE MOVED DISTANCE */

135  2       END PAR$INTERPOLATION;

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

            /*****************************************************************/
            /*                                                               */
            /******      PUBLIC  PROCEDURES  DECLARATIONS            ******/
            /*                                                               */
            /*****************************************************************/

            /*****************************************************************/
            /* FUNCTION: MACHINE$PARABOLA$CONTROL (CONTROL TOOL MOVEMENT)    */
            /* PARAMETERS INPUT: DEFINED IN PREVIOUS PROCEDURES              */
```

```
                /* OUTPUTS: DEFINED IN PREVIOUS PROCEDURES                    */
                /* PROCEDURE: PUBLIC                                          */
                /* CALLS: SHAFT$ENCODER, TOOL$ADVANCE, PAR$INTERPOLATION,     */
                /*        TOOL$RETRACT, CONSOLE$OUT, MESSAGE.                 */
                /* DESCRIPTION: TOOL MOVEMENT CAN BE SUBDIVIDED INTO THE      */
                /*           FOLLOWING SEGEMENTS:-                            */
                /*                                                            */
                /*  (4 TO 5) ADVANCE TOOL (LATHE IS OFF)                      */
                /*                   OPERATE LATHE, INITIALISE COUNTER        */
                /*  (5 TO 1) MOVEMENT WHILE TOOL IS HELD IN POSITION          */
                /*  (1 TO 2) CONTOUR SHAPE (PARABOLA)                         */
                /*                   SWITCH OFF LATHE                         */
                /*  (2 TO 3) RETRACT TOOL                                     */
                /************************************************************/
                /*                                                            */
                /*                       1 ++++++++++++++++++++++++ 5         */
                /*                       +                        +           */
                /*                       +                        +           */
                /*                      +                         +           */
                /*                    +                           +           */
                /*                  +                             +           */
                /*                +                               +           */
                /*              +                                 +           */
                /*          2 +                                   +           */
                /*            +                                   +           */
                /*            +                                   +           */
                /*            +                                   +           */
                /*            +                                   +           */
                /*            +                                   +           */
                /*            +                                   +           */
                /*          3 ++++++++++++++++++++++++++++++++++++ 4           */
                /*                                                            */
                /*                                                            */
                /************************************************************/

136   1     MACHINE$PARABOLA$CONTROL:
            PROCEDURE PUBLIC;

137   2     DECLARE FORM$LENGTH$ENCODER$PULSES(45)ADDRESS DATA(472,488
                  ,504,519,535,551,567,582,598,614,630,645,661,677,693
                  ,708,724,740,756,771,787,803,819,835,850,866,882,898
                  ,913,929,945,961,976,992,1008,1024,1039,1055,1071
                  ,1087,1102,1118,1134,1150,1165);

138   2     DECLARE INTERPOLATION$ENCODER$PULSES ADDRESS;

139   2     DECLARE PORT$2AH LITERALLY '2AH';
140   2     DECLARE J BYTE; /* GET THE FORM$LENGTH$MM INDEX */

              /* MASKS TO CONTROL LATHE */
141   2     DECLARE PORT$2AH$START$LATHE LITERALLY '0000$0100B';
142   2     DECLARE PORT$2AH$STOP$LATHE  LITERALLY '1111$1011B';

              /* DAC CONTROL VALUES */
```

```
143  2     DECLARE RETRACT$DAC$VALUE    LITERALLY '73';
144  2     DECLARE STOP$DAC$VALUE       LITERALLY '128';

145  2     DECLARE PORT$2BH LITERALLY '2BH';
146  2     DECLARE PORT$2BH$INPUT BYTE;
147  2     DECLARE PORT$2BH$ADVANCED LITERALLY '0000$0001B';
148  2     DECLARE PORT$2BH$CARRIAGE LITERALLY '0000$0100B';

149  2     DECLARE OLD$POSITION$ENCODER$PULSES ADDRESS;
150  2     DECLARE NEW$POSITION$ENCODER$PULSES ADDRESS;
151  2     DECLARE COUNT$1 BYTE;
152  2     DECLARE COUNT$2 BYTE;

           /*************************************************************/
           /*                                                           */
           /******        CALCULATIONS PRIOR TO THE PROCESS         ******/
           /*                                                           */
           /*************************************************************/

153  2     J=FORM$LENGTH-30; /* GET THE FORM$LENGTH$MM INDEX */
154  2     INTERPOLATION$ENCODER$PULSES=FORM$LENGTH$ENCODER$PULSES(J);

155  2     TURNING$REQUIRED=TRUE;
156  2     DO WHILE TURNING$REQUIRED;

157  3     CALL CONSOLE$OUT(CLEAR$SCREEN);  /* CLEAR SCREEN */

158  3     CALL MESSAGE(.PAGE$9,LAST(PAGE$9));  /* PAGE 9  */

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

           /*** INHIBIT COUNTER UNTIL INTERPOLATION BEGINS, INITIALISE */
           /***   SET LINE TO TRUE (5V)                               */

159  3       PORT$2AH$OUTPUT=PORT$2AH$OUTPUT OR COUNTER$RESET;
160  3       OUTPUT(PORT$2AH)=PORT$2AH$OUTPUT;

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

           /*** INITIALISE SDK-85 KEYBOARD DISPLAY ***/

161  3       KBD$DPLY$CONTROL=KMODE;
162  3       KBD$DPLY$CONTROL=KBNIT;

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

           /*** CLEAR SDK-85 KEYBOARD DISPLAY ***/

163  3       CALL UPDDT(CLEAR$DATA$FIELD);
164  3       CALL UPDAD(CLEAR$ADDRESS$FIELD);

           /*************************************************************/
```

```
            /*** DAC INITIALISE ***/

165   3       OUTPUT(22H)=STOP$DAC$VALUE;

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

            /*** INITIALISE ADDRESS POSITION ***/

166   3     CARRIAGE$POSITION$ENCODER$PULSES=0;

            /*************************************************************/
            /*                                                           */
            /****          THE MAIN PROGRAM                         ****/
            /*                                                           */
            /*************************************************************/

            /*************************************************************
          -  /

              /***         ADVANCE TOOL    (POSITION 4-5)              ***/

167   3       CALL TOOL$ADVANCE(ROLLER$POSITION);                    /* */
                                                                     /* */
            /*************************************************************/

              /***       SWITCH LATHE ON                             ***/

168   3       PORT$2AH$OUTPUT =PORT$2AH$OUTPUT OR PORT$2AH$START$LATHE;  /* */
169   3       OUTPUT(PORT$2AH)=PORT$2AH$OUTPUT;                      /* */
            /*************************************************************/

              /***  CARRIAGE WILL START MOVING LEFT   (POSITION  5-1)  ***/

             /* WAIT TILL CARRIAGE SWITCH IS CLOSED */

170   3      PORT$2BH$INPUT=INPUT(PORT$2BH); /* READ PORT */
171   3      DO WHILE (PORT$2BH$INPUT AND PORT$2BH$CARRIAGE)=0;
172   4         PORT$2BH$INPUT=INPUT(PORT$2BH); /* READ AGAIN */
173   4      END; /* DO WHILE */

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

             /***  ENABLE COUNTER ****/

174   3       PORT$2AH$OUTPUT=PORT$2AH$OUTPUT AND COUNTER$ENABLE;
175   3       OUTPUT(PORT$2AH)=PORT$2AH$OUTPUT;

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

              /***    PARABOLIC  CONTOUR   (POSITION  1-2)           ***/
```

```
                                                                        /* */
176   3       CALL PAR$INTERPOLATION(INTERPOLATION$ENCODER$PULSES);      /* */
                                                                        /* */
              /*************************************************************/
                                                                        /* */
               /* SWITCH OFF THE LATHE                                   */
                                                                        /* */
177   3        PORT$2AH$OUTPUT =PORT$2AH$OUTPUT AND PORT$2AH$STOP$LATHE; /* */
178   3        OUTPUT(PORT$2AH)=PORT$2AH$OUTPUT;                         /* */

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

               /***      RETRACT  TOOL   (POSITION  2-3)                ***/
               /***      RETURN TO DATUM                                ***/

179   3       PORT$2BH$INPUT=INPUT(PORT$2BH); /* READ PORT */

              /* ADVANCE PISTON TILL IT REACHES DATUM */
              /* MOVE PISTON TILL IT HITS ADVANCED SWITCH */


180   3       IF (PORT$2BH$INPUT AND PORT$2BH$ADVANCED)=0 THEN
181   3       DO;
182   4          OUTPUT(22H)=RETRACT$DAC$VALUE; /* START ADVANCE TO DATUM */
183   4          PORT$2BH$INPUT=INPUT(PORT$2BH); /* READ PORT */

184   4          DO WHILE (PORT$2BH$INPUT AND PORT$2BH$ADVANCED)=0;
185   5             PORT$2BH$INPUT=INPUT(PORT$2BH); /* READ AGAIN */
186   5          END;   /* DO WHILE */
187   4       END;  /* DO */
188   3       OUTPUT(22H)=STOP$DAC$VALUE;  /* STOP PISTON ADVANCE */

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

              /* A ROUTINE TO DISPLAY THE CARRIAGE POSITION ON THE SDK-85 */
              /*  EVERY 0.5 SECONDS TILL LEADSCREW IS STATIONARY          */

189   3        OLD$POSITION$ENCODER$PULSES=SHAFT$ENCODER;
190   3        COUNT$1=0;
191   3        DO WHILE COUNT$1<10;
192   4         COUNT$2=1;
193   4         DO WHILE COUNT$2<=25;
194   5            CALL TIME(200);
195   5            COUNT$2=COUNT$2+1;
196   5         END; /* DO WHILE */
197   4         NEW$POSITION$ENCODER$PULSES=SHAFT$ENCODER;
198   4         IF NEW$POSITION$ENCODER$PULSES=OLD$POSITION$ENCODER$PULSES THEN
199   4            COUNT$1=COUNT$1+1;
                 ELSE
200   4           DO;
201   5             COUNT$1=0;
```

```
202   5           OLD$POSITION$ENCODER$PULSES=NEW$POSITION$ENCODER$PULSES;
203   5         END; /* ELSE */
204   4       END; /* DO WHILE */

          /*****************************************************************
        -  /

205   3         CALL CONSOLE$OUT(CLEAR$SCREEN);  /* CLEAR SCREEN */

206   3         CALL MESSAGE(.PAGE$10,LAST(PAGE$10));  /* PAGE 10 */

207   3         ANSWER=CONSOLE$IN; /* GET REPLY */
208   3         DO WHILE ANSWER<>'Y' AND ANSWER<>'N';
209   4           CALL CONSOLE$OUT(BELL);
210   4           CALL MESSAGE(.ERROR$1,LAST(ERROR$1));
211   4           ANSWER=CONSOLE$IN;
212   4         END; /* DO WHILE */
213   3         CALL MESSAGE(.DELETE$ERROR$1,LAST(DELETE$ERROR$1));
214   3         CALL MESSAGE(.MESS$1,LAST(MESS$1));
215   3         CALL CONSOLE$OUT(ANSWER);
216   3         CALL GET(CARRIAGE$RETURN);
217   3         IF ANSWER='N' THEN
218   3           TURNING$REQUIRED=FALSE;
219   3         END; /* DO WHILE */

220   2         CALL CONSOLE$OUT(CLEAR$SCREEN);

221   2       END MACHINE$PARABOLA$CONTROL;

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

222   1     END MACHINE$PARABOLA$CONTROL$MODULE;



MODULE INFORMATION:

    CODE AREA SIZE     = 08A5H   2213D
    VARIABLE AREA SIZE = 0024H     36D
    MAXIMUM STACK SIZE = 0008H      8D
    697 LINES READ
    0 PROGRAM ERROR(S)

END OF PL/M-80 COMPILATION
```

```
asm80 :f2:disply.asm debug


ISIS-II 8080/8085 MACRO ASSEMBLER, V4.1        DISPLA   PAGE    1


 LOC  OBJ        LINE       SOURCE STATEMENT

                    1 $PAGEWIDTH(80)
                    2 $PAGELENGTH(55)
                    3
                    4 ;*****************************************************
                    5
                    6         NAME    DISPLAY
                    7
                    8 ;*****************************************************
                    9
                   10         PUBLIC  UPDDT
                   11         PUBLIC  UPDAD
                   12
                   13 ;*****************************************************
                   14
 20F9              15         OBUFF   EQU     20F9H
 0001              16         DTFLD   EQU     01H
 0090              17         ADISP   EQU     90H
 0094              18         DDISP   EQU     94H
 1900              19         CNTRL   EQU     1900H
 0008              20         DTMSK   EQU     08H
 1800              21         DSPLY   EQU     1800H
                   22
                   23 ;*****************************************************
                   24         CSEG
                   25 ;*****************************************************
                   26 HXDSP:
 0000 7A           27         MOV     A,D     ; GET FIRST DATA BYTE
 0001 0F           28         RRC             ; CONVERT 4 HIGH ORDER BITS
 0002 0F           29         RRC             ; /TO A SINGLE CHARACTER
 0003 0F           30         RRC             ;
 0004 0F           31         RRC             ;
 0005 E60F         32         ANI     0FH     ;
 0007 21F920       33         LXI     H,OBUFF ; GET ADDRESS OR OUTPUT BUFFER
 000A 77           34         MOV     M,A     ; STORE CHARACTER IN OUTPUT
                   35                         ; BUFFER
 000B 7A           36         MOV     A,D     ; GET FIRST DATA BYTE AND
                   37                         ; CONVERT 4 LOW ORDER
 000C E60F         38         ANI     0FH     ; /BITS TO A SINGLE CHARACTER
 000E 23           39         INX     H       ; NEXT BUFFER POSITION
 000F 77           40         MOV     M,A     ; STORE CHARACTER IN BUFFER
 0010 7B           41         MOV     A,E     ; GET SECOND DATA BYTE
 0011 0F           42         RRC             ; CONVERT 4 HIGH ORDER BITS
 0012 0F           43         RRC             ; /TO A SINGLE CHARACTER
```

```
LOC  OBJ                SOURCE STATEMENT

0013 0F                         RRC                 ;
0014 0F                         RRC                 ;
0015 E60F                       ANI     0FH         ;
0017 23                         INX     H           ; NEXT BUFFER POSITION
0018 77                         MOV     M,A         ; STORE CHARACTER IN BUFFER
0019 7B                         MOV     A,E         ; GET SECOND DATA BYTE AND
                                                    ;  CONVERT
                                                    ; LOW ORDER
001A E60F                       ANI     0FH         ; /4 BITS TO A SINGLE CHARACTER
001C 23                         INX     H           ; NEXT BUFFER POSITION
001D 77                         MOV     M,A         ; STORE CHARACTER IN BUFFER
001E 21F920                     LXI     H,OBUFF     ; RETURN ADDRESS OF OUTPUT
                                                    ; BUFFER IN H & L
0021 C9                         RET                 ;

                        ;***************************************************
                        OUTPT:
0022 0F                         RRC                 ; USE DATA FIELD ?
0023 DA2D00      C              JC      OUT05       ; YES-GO SET UP TO USE DATA
                                                    ; FIELD
0026 0E04                       MVI     C,4         ; NO-COUNT FOR ADDRESS FIELD
0028 3E90                       MVI     A,ADISP     ; CONTROL CHARACTER FOR OUTPUT
                                                    ; TO/ADDRESS FIELD OF DISPLAY
002A C33100      C              JMP     OUT10       ;
002D 0E02               OUT05:  MVI     C,2         ; COUNT FOR DATA FIELD
002F 3E94                       MVI     A,DDISP     ; CONTROL CHARACTER FOR OUTPUT
                                                    ; TO DATA FIELD/ OF DISPLAY
0031 320019             OUT10:  STA     CNTRL       ;
0034 7E                 OUT15:  MOV     A,M         ; GET OUTPUT CHARACTER
0035 EB                         XCHG                ; SAVE OUTPUT CHARACTER ADDRESS
                                                    ; IN D & E
0036 216600      C              LXI     H,DSPTB     ; GET DISPLAY FORMAT TABLE
                                                    ; ADDRESS
0039 85                         ADD     L           ; USE OUTPUT CHARACTER AS A
                                                    ; POINTER TO
003A 6F                         MOV     L,A         ; /DISPLAY FORMAT TABLE
003B 7E                         MOV     A,M         ; GET DISPLAY FORMAT CHARACTER
                                                    ; FROM TABLE
003C 61                         MOV     H,C         ; TEST COUNTER WITHOUT CHANGING
                                                    ; IT
003D 25                         DCR     H           ; IS THIS THE LAST CHARACTER ?
003E C24700      C              JNZ     OUT20       ; NO-GO OUTPUT CHARACTER AS IS
0041 05                         DCR     B           ; YES-IS DOT FLAG SET ?
0042 C24700      C              JNZ     OUT20       ; NO-GO OUTPUT CHARACTER AS IS
```

```
LOC  OBJ          LINE      SOURCE STATEMENT

0045 F608          88        ORI     DTMSK   ; YES-OR IN MASK TO DISPLAY DOT
                   89                        ; WITH/ LAST CHARACTER
0047 2F            90 OUT20: CMA             ; COMPLEMENT OUTPUT CHARACTER
0048 320018        91        STA     DSPLY   ; SEND CHARACTER TO DISPLAY
004B EB            92        XCHG            ; RETRIEVE OUTPUT CHARACTER
                   93                        ; ADDRESS
004C 23            94        INX     H       ; NEXT OUTPUT CHARACTER
004D 0D            95        DCR     C       ; ANY MORE OUTPUT CHARACTERS ?
004E C23400    C   96        JNZ     OUT15   ; YES-GO PROCESS ANOTHER
                   97                        ; CHARACTER
0051 C9            98        RET             ; NO - RETURN
                   99
                  100 ;##################################################
                  101 UPDDT:
0052 51           102        MOV     D,C     ;
0053 CD0000    C  103        CALL    HXDSP   ;
0056 3E01         104        MVI     A,DTFLD ;
0058 CD2200    C  105        CALL    OUTPT   ;
005B C9           106        RET             ;
                  107
                  108 ;##################################################
                  109 UPDAD:
005C 50           110        MOV     D,B     ;
005D 59           111        MOV     E,C     ;
005E CD0000    C  112        CALL    HXDSP   ;
0061 AF           113        XRA     A       ;
0062 CD2200    C  114        CALL    OUTPT   ;
0065 C9           115        RET             ;
                  116
                  117 ;##################################################
                  118 ;
                  119 DSPTB:  ; TABLE FOR TRANSLATING CHARACTERS FOR OUTPUT
                  120 ;
                  121 ;                     DISPLAY
                  122 ;                     FORMAT    CHARACTER
                  123 ;                     ======    =========
                  124 ;
0066 F3           125          DB        0F3H    ;    0
0067 60           126          DB         60H    ;    1
0068 B5           127          DB        0B5H    ;    2
0069 F4           128          DB        0F4H    ;    3
006A 66           129          DB         66H    ;    4
006B D6           130          DB        0D6H    ;    5 AND S
006C D7           131          DB        0D7H    ;    6
```

```
 LOC  OBJ       LINE       SOURCE STATEMENT

 006D 70         132          DB        70H    ;   7
 006E F7         133          DB       0F7H    ;   8
 006F 76         134          DB        76H    ;   9
 0070 77         135          DB        77H    ;   A
 0071 C7         136          DB       0C7H    ;   B (LOWER CASE)
 0072 93         137          DB        93H    ;   C
 0073 E5         138          DB       0E5H    ;   D (LOWER CASE)
 0074 97         139          DB        97H    ;   E
 0075 17         140          DB        17H    ;   F
                 141
                 142 ;##################################################
                 143        END            ;


PUBLIC SYMBOLS
UPDAD  C 005C    UPDDT  C 0052

EXTERNAL SYMBOLS


USER SYMBOLS
ADISP  A 0090    CNTRL  A 1900    DDISP  A 0094    DSPLY  A 1800
DSPTB  C 0066    DTFLD  A 0001    DTMSK  A 0008    HXDSP  C 0000
OBUFF  A 20F9    OUT05  C 002D    OUT10  C 0031    OUT15  C 0034
OUT20  C 0047    OUTPT  C 0022    UPDAD  C 005C    UPDDT  C 0052

ASSEMBLY COMPLETE,  NO ERRORS
```

```
asm80 :f2:inro.asm debug


ISIS-II 8080/8085 MACRO ASSEMBLER, V4.1        INRO     PAGE    1


 LOC  OBJ        LINE       SOURCE STATEMENT

                    1 $PAGEWIDTH(80)
                    2 $PAGELENGTH(55)
                    3
                    4 ;*****************************************
                    5
                    6     NAME      INRO
                    7
                    8 ;*****************************************
                    9
                   10     PUBLIC    INRO
                   11
                   12 ;*****************************************
                   13
                   14     CSEG      ; CODE RELOCATABLE
                   15
                   16 ;*****************************************
                   17 ;
                   18 ;    RETRACT TOOL TO DATUM
                   19 ;
                   20 ;*****************************************
                   21
0000 3E49          22 INRO: MVI  A,49H    ; OUTPUT TOOL
0002 D322          23       OUT  22H      ; RETRACT COMMAND
0004 DB2B          24 LAB1: IN   2BH      ; WAIT TILL TOOL
0006 E601          25       ANI  1H       ; FULLY RETRACTED
0008 CA0400    C   26       JZ   LAB1
000B 3E80          27       MVI  A,80H    ; STOP
000D D322          28       OUT  22H      ; TOOL
                   29
                   30 ;*****************************************
                   31 ;
                   32 ;    SWITCH OFF THE LATHE AND THE OIL PUMP
                   33 ;
                   34 ;*****************************************
                   35
000F 3E00          36     MVI  A,0H      ; OUTPUT TOOL RETRACT
0011 D32A          37     OUT  2AH       ; COMMAND
                   38
                   39 ;*****************************************
                   40 ;
                   41 ;    RETURN TO MONITOR
                   42 ;
                   43 ;*****************************************
```

```
  LOC  OBJ        LINE       SOURCE STATEMENT

                    44
  0013 CF           45     RST 1
                    46
                    47 ;##############################################
                    48
                    49     END


PUBLIC SYMBOLS
INR0   C 0000

EXTERNAL SYMBOLS


USER SYMBOLS
INR0   C 0000    LAB1   C 0004

ASSEMBLY COMPLETE,  NO ERRORS
```

```
asm80 :f2:invect.asm debug


ISIS-II 8080/8085 MACRO ASSEMBLER, V4.1        INVECT   PAGE    1


  LOC  OBJ        LINE       SOURCE STATEMENT

                     1 $PAGEWIDTH(80)
                     2 $PAGELENGTH(55)
                     3
                     4 ;##########################################
                     5
                     6      NAME      INVECT
                     7
                     9 ;##########################################
                     9
                    10      EXTRN    INRO
                    11
                    12 ;##########################################
                    13
                    14 ; RST 6.5 INTERRUPT VECTOR
  20C8              15     ORG  20C8H
  20C8 C30000   E   16     JMP  INRO       ; JUMP TO INTERRUPT
                    17                     ; SERVICE ROUTINE
                    18 ; RST 7.5 INTERRUPT VECTOR
  20CE              19     ORG  20CEH
  20CE C30000   E   20     JMP  INRO       ; JUMP TO INTERRUPT
                    21                     ; SERVICE ROUTINE
                    22
                    23 ;##########################################
                    24
                    25      END


PUBLIC SYMBOLS


EXTERNAL SYMBOLS
INRO   E 0000

USER SYMBOLS
INRO   E 0000

ASSEMBLY COMPLETE,   NO ERRORS
```

```
as80 :f2:init.asm debug


ISIS-II 8080/8085 MACRO ASSEMBLER, V4.1        INIT     PAGE    1


  LOC  OBJ        LINE       SOURCE STATEMENT

                     1 $PAGEWIDTH(80)
                     2 $PAGELENGTH(55)
                     3 $MOD85
                     4
                     5 ;*************************************************
                     6
                     7         NAME      INIT
                     8
                     9 ;*************************************************
                    10
                    11         PUBLIC    INIT
                    12
                    13 ;*************************************************
                    14
                    15        CSEG         ; CODE RELOCATABLE
                    16
                    17 ;*************************************************
                    18
                    19  INIT:
                    20
  0000 3E09         21        MVI    A,09H
  0002 30           22        SIM           ; UNMASK RST 7.5 AND 6.5
  0003 FB           23        EI            ; ENABLE ALL INTERRUPTS
  0004 C9           24        RET           ; RETURN TO INITIALISATION MODULE
                    25
                    26 ;*************************************************
                    27
                    28        END


PUBLIC SYMBOLS
INIT   C 0000

EXTERNAL SYMBOLS


USER SYMBOLS
INIT   C 0000

ASSEMBLY COMPLETE,   NO ERRORS
```

```
10 LPRINT "Y VALUE","X VALUE","X RATIO"
20 LPRINT "-------","-------","-------"
30 FOR Y=0 TO 74 STEP 1.016
40 X=(Y^2)/273.8
50 IF Y=0 THEN LPRINT Y,X: GOTO 180
60 FOR I=1 TO 1000
70 J=I/X
80 J(I)=INT(J+.5)
90 D(I)=ABS(I/J(I)-X)
110 NEXT I
120 T=999:TI=0
130 FOR I=1 TO 1000
140 IF D(I)<T THEN T=D(I):TI=I
150 NEXT I
160 I=TI
170 LPRINT Y,X,I;"/";J(I)
180 NEXT Y
190 END
```

Y VALUE	X VALUE	X RATIO	
0	0		
1.016	3.77011E-03	131 / 34747	
2.032	1.508044E-02		524 / 34747
3.048	3.393099E-02		651 / 19186
4.064	6.032176E-02		45 / 746
5.08	9.425274E-02		961 / 10196
6.096	.1357239	704 / 5187	
7.112	.1847353	806 / 4363	
8.128	.241287	90 / 373	
9.143999	.3053788	528 / 1729	
10.16	.3770109	961 / 2549	
11.176	.4561832	937 / 2054	
12.192	.5428957	405 / 746	
13.208	.6371484	295 / 463	
14.224	.7389413	852 / 1153	
15.24	.8482745	123 / 145	
16.256	.9651479	360 / 373	
17.272	1.089562	961 / 882	
18.288	1.221516	193 / 158	
19.304	1.36101	377 / 277	
20.32	1.508044	656 / 435	
21.336	1.662619	685 / 412	
22.352	1.824734	177 / 97	
23.36801	1.994389	355 / 178	
24.38401	2.171584	810 / 373	
25.40001	2.35632	205 / 87	
26.41601	2.548595	813 / 319	
27.43201	2.748412	863 / 314	
28.44801	2.955768	869 / 294	
29.46401	3.170664	929 / 293	
30.48001	3.393101	492 / 145	
31.49601	3.623078	471 / 130	
32.51201	3.860595	637 / 165	

A3.1: Basic program to find the integer ratios (1)

33.52801	4.105652	583 / 142
34.54401	4.358248	292 / 67
35.56001	4.618386	956 / 207
36.57601	4.886063	386 / 79
37.592	5.161281	160 / 31
38.608	5.444039	969 / 178
39.624	5.734337	367 / 64
40.64	6.032175	187 / 31
41.656	6.337554	995 / 157
42.672	6.650473	685 / 103
43.688	6.970932	718 / 103
44.704	7.298931	708 / 97
45.72	7.634471	710 / 93
46.73599	7.97755	710 / 89
47.75199	8.32817	533 / 64
48.76799	8.68633	443 / 51
49.78399	9.052029	869 / 96
50.79999	9.425269	820 / 87
51.81599	9.80605	961 / 98
52.83199	10.19437	367 / 36
53.84799	10.59023	646 / 61
54.86399	10.99363	1000 / 91
55.87998	11.40458	479 / 42
56.89598	11.82306	934 / 79
57.91198	12.24908	49 / 4
58.92798	12.68264	799 / 63
59.94398	13.12374	958 / 73
60.95998	13.57238	95 / 7
61.97598	14.02857	491 / 35
62.99198	14.49229	942 / 65
64.00798	14.96355	823 / 55
65.02397	15.44236	803 / 52
66.03997	15.9287	223 / 14
67.05597	16.42258	739 / 45
68.07197	16.92401	897 / 53
69.08797	17.43297	523 / 30
70.10397	17.94947	359 / 20
71.11996	18.47352	351 / 19
72.13596	19.0051	19 / 1
73.15196	19.54423	899 / 46

A3.2: Basic program to find the integer ratios (2)

Sample Number	Maximum thickness (mm)	Minimum thickness (mm)	Mean thickness (mm)	Range in thickness reduction (mm)	Microscope reading at 10 Kg load (mm)	Hardness VHN	Reduction in thickness (percentage) %
1	1.120	0.850	1.012	0.270	0.973	19.59	36.75
2	1.198	0.841	1.019	0.357	0.888	23.95	36.31
3	1.205	0.842	1.039	0.363	0.886	23.62	35.06
4	1.675	1.178	1.382	0.497	0.871	24.44	13.62
5	1.734	1.156	1.417	0.578	0.878	24.06	11.43
6	1.646	1.166	1.364	0.480	0.916	22.10	14.75
7	1.675	1.308	1.492	0.367	0.897	23.05	6.75
8	1.678	1.295	1.457	0.383	0.886	23.62	8.93
9	1.664	1.267	1.446	0.397	0.872	24.39	9.62
10	1.670	1.416	1.551	0.254	0.892	23.31	3.06
11	1.626	1.425	1.509	0.201	1.018	17.89	5.68
12	1.585	1.428	1.501	0.157	1.023	17.72	6.18
13	1.611	1.360	1.449	0.251	1.022	17.75	9.43
14	1.625	1.403	1.496	0.222	1.018	17.89	6.50
15	1.621	1.395	1.489	0.226	1.016	17.96	6.93
16	1.608	1.370	1.488	0.238	1.013	18.07	7.00
17	1.689	1.320	1.465	0.369	0.883	23.78	8.43
18	1.694	1.342	1.474	0.352	0.882	23.84	7.87
19	1.643	1.318	1.468	0.325	0.814	27.99	8.25
20	1.621	1.294	1.467	0.327	0.816	27.85	8.31
21	1.655	1.296	1.463	0.359	0.815	27.92	8.56
22	1.250	0.966	1.084	0.284	0.789	29.79	32.25
23	1.296	0.969	1.106	0.327	0.792	29.56	30.87
24	1.279	1.006	1.104	0.273	0.780	30.48	31.00
25	1.685	1.180	1.405	0.505	0.877	24.11	12.18
26	1.660	1.165	1.369	0.495	0.894	23.20	14.43
27	1.688	1.114	1.368	0.574	0.887	23.57	14.50
28	1.631	1.290	1.440	0.341	0.893	23.25	10.00
29	1.669	1.359	1.511	0.310	0.886	23.62	5.56
30	1.680	1.343	1.488	0.337	0.879	24.00	7.00
31	1.649	1.408	1.531	0.241	0.875	24.22	4.31
32	1.700	1.437	1.568	0.263	0.884	23.73	2.00
33	1.631	1.411	1.526	0.220	0.828	27.05	4.62
34	1.266	1.000	1.135	0.266	0.875	24.22	29.06
35	1.333	0.903	1.124	0.430	0.893	23.25	29.75
36	1.305	1.010	1.150	0.295	0.864	24.84	28.12
37	1.686	1.188	1.392	0.498	0.873	24.33	13.00
38	1.658	1.100	1.359	0.558	0.877	24.11	15.06
39	1.654	1.140	1.351	0.514	0.872	24.39	15.56
40	1.620	1.283	1.454	0.337	0.888	23.52	9.12
41	1.650	1.276	1.433	0.374	0.896	23.10	10.43
42	1.627	1.294	1.457	0.333	0.874	24.28	8.93
43	1.714	1.412	1.537	0.302	0.907	22.54	3.93

Table 1: Results from the test samples (1)

Sample Number	Maximum thickness (mm)	Minimum thickness (mm)	Mean thickness (mm)	Range in thickness reduction (mm)	Microscope reading at 10 Kg load (mm)	Hardness VHN	Reduction in thickness (percentage) %
44	1.656	1.387	1.533	0.269	0.895	23.15	4.18
45	1.681	1.390	1.521	0.291	0.884	23.73	4.93
46	1.214	0.893	1.036	0.321	0.873	24.33	35.25
47	1.290	0.914	1.077	0.376	0.869	24.56	32.68
48	1.292	0.998	1.144	0.294	0.887	23.57	28.50
49	1.701	1.086	1.352	0.615	0.890	23.41	15.50
50	1.738	1.134	1.379	0.604	0.885	23.68	13.81
51	1.731	1.123	1.367	0.608	0.890	23.41	14.56
52	1.694	1.304	1.474	0.390	0.879	24.00	7.87
53	1.682	1.248	1.435	0.434	0.900	22.89	10.30
54	1.724	1.332	1.476	0.392	0.884	23.73	7.75
55	1.678	1.386	1.516	0.292	0.885	23.68	5.25
56	1.676	1.377	1.509	0.299	0.884	23.73	5.68
57	1.659	1.366	1.511	0.293	0.883	23.78	5.56
58	1.549	1.249	1.416	0.300	0.892	23.31	11.50
59	1.739	1.410	1.546	0.329	0.884	23.73	3.37
60	1.747	1.432	1.586	0.315	0.893	23.25	0.87
61	1.756	1.523	1.612	0.233	0.897	23.05	1.20
62	1.476	1.095	1.282	0.381	0.889	23.46	19.87
63	1.706	1.365	1.493	0.341	0.891	23.36	6.68
64	1.699	1.424	1.556	0.275	0.889	23.46	2.75
65	1.674	1.450	1.557	0.224	0.876	24.17	2.68
66	1.430	0.937	1.175	0.493	0.885	23.68	26.56
67	1.664	1.170	1.367	0.494	0.917	22.05	14.56
68	1.701	1.257	1.458	0.444	0.870	24.50	8.87
69	1.826	1.381	1.524	0.445	0.892	23.31	4.75
70	1.527	0.960	1.185	0.567	0.873	24.33	25.93
71	1.406	1.027	1.206	0.379	0.879	24.00	24.62
72	1.418	1.061	1.230	0.357	0.869	24.56	23.12
73	1.683	1.099	1.371	0.584	0.878	24.06	14.31
74	1.670	1.149	1.376	0.521	0.880	23.95	14.00
75	1.763	1.163	1.406	0.600	0.874	24.28	12.12
76	1.733	1.273	1.451	0.460	0.891	23.36	9.31
77	1.693	1.314	1.467	0.379	0.874	24.28	8.31
78	1.641	1.535	1.473	0.288	0.879	24.00	7.93
79	1.724	1.341	1.472	0.383	0.886	23.62	8.00
80	1.661	1.342	1.480	0.320	0.883	23.78	7.50
81	1.693	1.297	1.445	0.396	0.672	41.10	9.68
82	1.695	1.303	1.469	0.392	0.670	41.30	8.18
83	1.670	1.295	1.472	0.375	0.668	41.60	8.00
84	1.728	1.315	1.483	0.413	0.664	42.10	7.31
85	1.714	1.271	1.459	0.443	0.661	42.40	8.81
86	1.726	1.282	1.464	0.444	0.672	41.10	8.50

Table 2: Results from the test samples (2)

Sample Number	Maximum thickness (mm)	Minimum thickness (mm)	Mean thickness (mm)	Range in thickness reduction (mm)	Microscope reading at 10 Kg load (mm)	Hardness VHN	Reduction in thickness (percentage) %
87	1.756	1.339	1.500	0.417	0.658	42.8	6.25
88	1.756	1.246	1.437	0.510	0.659	42.7	10.18
89	1.724	1.315	1.482	0.409	0.662	42.3	7.37
90	1.442	1.010	1.229	0.432	0.650	43.9	23.18
91	1.458	1.010	1.221	0.448	0.671	41.2	23.68
92	1.515	0.974	1.234	0.541	0.656	43.1	22.87
93	1.772	1.176	1.390	0.596	0.658	42.8	13.12
94	1.805	1.208	1.437	0.597	0.654	43.4	10.18
95	1.713	1.150	1.375	0.563	0.650	43.9	14.06
96	1.659	1.258	1.421	0.401	0.670	41.3	11.18
97	1.757	1.308	1.466	0.449	0.665	41.9	8.37
98	1.697	1.230	1.438	0.467	0.685	39.5	10.12
99	1.772	1.421	1.547	0.351	0.679	40.2	3.31
100	1.761	1.388	1.521	0.373	0.662	42.3	4.93
101	1.735	1.366	1.494	0.369	0.667	41.7	6.62
102	1.262	0.949	1.121	0.313	0.638	45.6	29.93
103	1.366	0.905	1.168	0.461	0.653	43.5	27.00
104	1.372	1.016	1.186	0.356	0.644	44.7	25.87
105	1.732	1.102	1.376	0,630	0.665	41.9	14.00
106	1.664	1.143	1.362	0.521	0.649	44.0	14.87
107	1.687	1.149	1.387	0.538	0.658	42.8	13.31
108	1.734	1.263	1.456	0.471	0.653	43.5	9.00
109	1.652	1.295	1.474	0.357	0.653	43.5	7.87
110	1.689	1.271	1.460	0.418	0.654	43.4	8.75
111	1.719	1.355	1.511	0.364	0.664	42.1	5.56
112	1.663	1.401	1.520	0.262	0.673	40.9	5.00
113	1.683	1.398	1.546	0.285	0.661	42.4	3.37
114	1.393	0.975	1.188	0.418	0.647	44.3	25.75
115	1.415	0.989	1.188	0.426	0.647	44.3	25.75
116	1.393	0.934	1.152	0.459	0.653	43.5	28.00
117	1.701	1.094	1.334	0.607	0.653	43.5	16.62
118	1.685	1.125	1.362	0.560	0.651	43.8	14.87
119	1.702	1.173	1.404	0.529	0.656	43.1	12.25
120	1.672	1.278	1.453	0.394	0.663	42.2	9.18
121	1.702	1.275	1.474	0.427	0.676	40.6	7.87
122	1.747	1.272	1.466	0.475	0.670	41.3	8.37
123	1.665	1.418	1.558	0.247	0.659	42.7	2.62
124	1.704	1.349	1.493	0.355	0.671	41.2	6.68
125	1.669	1.390	1.521	0.279	0.658	42.8	4.93
126	1.547	1.225	1.384	0.322	0.666	41.8	13.5
127	1.519	1.307	1.384	0.212	0.678	40.3	13.5
128	1.472	1.249	1.357	0.223	0.722	35.6	15.18
129	1.732	1.406	1.541	0.326	0.729	34.9	3.68

Table 3: Results from the test samples (3)

Sample Number	Maximum thickness (mm)	Minimum thickness (mm)	Mean thickness (mm)	Range in thickness reduction (mm)	Microscope reading at 10 Kg Load (mm)	Hardness VHN	Reduction in thickness (percentage) %
130							
131	1.731	1.366	1.553	0.365	0.742	33.68	2.93
132	1.742	1.388	1.566	0.354	0.715	36.30	3.40
133	1.727	1.518	1.634	0.209	0.729	34.90	2.12
134	1.785	1.469	1.599	0.316	0.662	42.30	0.06
135	1.741	1.463	1.590	0.278	0.681	40.00	0.62
136	1.795	1.496	1.596	0.299	0.663	42.20	0.25
137	1.674	1.503	1.585	0.171	0.672	41.10	0.93
138	1.728	1.526	1.626	0.202	0.668	41.60	1.62
139	1.452	1.126	1.287	0.326	0.653	43.50	19.56
140	1.538	1.156	1.320	0.382	0.658	42.80	17.50
141	1.514	1.161	1.310	0.353	0.654	43.40	18.12
142	1.701	1.286	1.463	0.415	0.648	44.20	8.56
143	1.724	1.302	1.503	0.422	0.652	43.60	6.06
144	1.774	1.297	1.482	0.477	0.704	37.40	7.37
145	1.710	1.424	1.543	0.286	0.653	43.50	3.56
146	1.753	1.408	1.546	0.345	0.665	41.90	3.37
147	1.719	1.492	1.608	0.227	0.664	42.10	0.50
148	1.704	1.463	1.562	0.241	0.658	42.80	2.37
149	1.695	1.483	1.571	0.212	0.661	42.40	1.81
150	1.689	1.460	1.576	0.229	0.648	44.20	1.50
151	1.433	0.992	1.208	0.441	0.641	45.10	24.5
152	1.311	0.997	1.150	0.314	0.644	44.70	28.12
153	1.452	0.911	1.148	0.541	0.648	44.20	28.25
154	1.726	1.123	1.342	0.603	0.646	44.40	16.12
155	1.685	1.161	1.416	0.524	0.643	44.90	11.50
156	1.657	1.120	1.377	0.537	0.657	43.00	13.93
157	1.694	1.252	1.448	0.442	0.654	43.40	9.50
158	1.670	1.296	1.483	0.374	0.652	43.60	7.31
159	1.667	1.258	1.454	0.409	0.663	42.20	9.12
160	1.728	1.352	1.489	0.376	0.658	42.80	6.93
161	1.681	1.409	1.522	0.272	0.665	41.90	4.87
162	1.662	1.362	1.505	0.300	0.666	41.80	4.87

Table 4: Results from the test samples (4)

Sample Number	Input Angle (degrees)	Measured Angle (degrees)		Mean value for the two sides	Mean value for the three samples	Angle difference
		side 1	side 2	(degrees)	(degrees)	(degrees)
1	30	33.50	32.25	32.87		
2	30	34.00	32.00	33.00	33.41	3.41
3	30	33.50	35.25	34.37		
4	35	34.80	37.60	36.20		
5	35	34.90	37.60	36.25	36.20	1.20
6	35	34.90	37.40	36.15		
7	40	42.50	39.80	41.15		
8	40	42.00	39.80	40.90	40.97	0.97
9	40	42.00	39.75	40.87		
10	45	44.00	46.50	45.25		
11	45	43.50	46.50	45.00	44.95	0.05
12	45	43.50	45.70	44.60		
13	40	40.90	38.50	39.70		
14	40	41.00	38.50	39.75	39.73	0.27
15	40	41.00	38.50	39.75		
16	40	38.50	41.25	39.87		
17	40	39.50	42.00	40.75	40.62	0.62
18	40	40.00	42.50	41.25		
19	40	42.50	40.50	41.50		
20	40	42.00	40.00	41.00	41.50	1.50
21	40	43.00	41.00	42.00		
22	30	33.00	35.50	34.25		
23	30	32.70	35.00	33.85	34.06	4.06
24	30	32.70	35.50	34.10		
25	35	36.80	34.75	35.77		
26	35	36.80	34.00	35.40	35.51	0.51
27	35	36.50	34.25	35.37		
28	40	40.50	43.00	41.75		
29	40	40.25	43.25	41.75	41.55	1.55
30	40	39.80	42.50	41.15		
31	45	46.25	44.50	45.37		
32	45	46.50	44.50	45.50	45.84	0.84
33	45	48.80	44.50	46.65		
34	30	32.25	33.50	33.37		
35	30	34.50	32.50	33.50	33.45	3.45
36	30	34.50	32.50	33.50		
37	35	37.00	34.50	35.75		
38	35	37.50	35.00	36.25	36.08	1.08
39	35	37.50	35.00	36.25		
40	40	39.25	42.50	40.87		
41	40	39.00	42.50	40.75	40.99	0.99
42	40	40.00	42.75	41.37		

Table 5: Results from the test samples (5)

Sample Number	Input Angle (degrees)	Measured Angle (degrees)		Mean value for the two sides	Mean value for the three samples	Angle difference
		side 1	side 2	(degrees)	(degrees)	(degrees)
43	45	47.50	43.60	45.55		
44	45	46.50	44.50	45.50	45.47	0.47
45	45	46.50	44.25	45.37		
46	30	33.00	35.00	34.00		
47	30	32.50	35.00	33.75	33.83	3.83
48	30	32.50	35.00	33.75		
49	35	37.25	35.00	36.12		
50	35	36.50	34.50	35.50	35.70	0.70
51	35	36.50	34.50	35.50		
52	40	40.00	41.80	40.90		
53	40	39.75	41.80	40.77	40.90	0.90
54	40	39.60	42.50	41.05		
55	45	46.50	45.00	45.75		
56	45	46.75	44.50	45.62	45.62	0.62
57	45	46.75	44.25	45.50		
58	30	33.00	35.40	34.20		4.20
59	35	38.50	35.25	36.87		1.87
60	40	40.50	43.50	42.00		2.00
61	45	47.00	46.50	46.75		1.75
62	30	32.50	35.00	33.75		3.75
63	35	37.25	35.50	36.37		1.37
64	40	41.00	42.50	41.75		1.75
65	45	48.25	45.00	46.62		1.62
66	30	32.50	34.80	33.65		3.65
67	35	37.25	35.00	36.12		1.12
68	40	41.00	42.50	41.75		1.75
69	45	46.75	44.75	45.75		0.75
70	30	32.70	34.75	33.72		
71	30	32.50	34.75	33.62	33.78	3.78
72	30	32.50	35.50	34.00		
73	35	37.25	34.50	35.87		
74	35	37.00	35.00	36.00	36.04	1.04
75	35	37.00	35.50	36.25		
76	40	40.50	42.50	41.50		
77	40	40.00	42.50	41.25	40.91	0.91
78	40	41.00	39.00	40.00		
79	45	46.00	47.50	46.75		
80	45	45.40	47.50	46.45	46.48	1.48
81	45	45.00	47.50	46.25		
82	40	41.75	40.00	40.87		
83	40	40.80	42.70	41.75	41.34	1.34
84	40	42.30	40.50	41.40		

Table 6: Results from the test samples (6)

Sample Number	Input Angle (degrees)	Measured Angle (degrees)		Mean value for the two sides	Mean value for the three samples	Angle difference
		side 1	side 2	(degrees)	(degrees)	(degrees)
85	40	40.75	42.50	41.62		
86	40	42.00	40.40	41.20	41.52	1.52
87	40	40.50	43.00	41.75		
88	40	43.00	40.25	41.62		
89	40	40.00	42.80	41.40	41.26	1.26
90	40	41.80	39.75	40.77		
91	30	33.00	35.00	34.00		
92	30	35.00	33.00	34.00	33.83	3.83
93	30	32.50	34.50	33.50		
94	35	37.00	35.50	36.25		
95	35	35.50	37.50	36.50	36.33	1.33
96	35	37.00	35.50	36.25		
97	40	40.50	42.50	41.50		
98	40	42.60	40.50	41.55	41.51	1.51
99	40	40.50	42.50	41.50		
100	45	46.70	45.00	45.85		
101	45	45.30	46.50	45.90	46.15	1.15
102	45	48.00	45.40	46.70		
103	30	32.80	34.70	33.75		
104	30	34.50	32.70	33.60	33.73	3.73
105	30	32.70	35.00	33.85		
106	35	36.50	35.50	36.00		
107	35	35.50	37.00	36.25	36.25	1.25
108	35	37.00	35.00	36.00		
109	40	40.50	42.50	41.50		
110	40	42.50	40.00	41.25	41.41	1.41
111	40	40.00	43.00	41.50		
112	45	46.00	44.00	45.00		
113	45	44.00	47.25	45.62	45.62	0.62
114	45	47.50	45.00	46.25		
115	30	32.50	35.00	33.75		
116	30	34.50	32.50	33.50	33.70	3.70
117	30	32.70	35.00	33.85		
118	35	37.50	35.00	36.25		
119	35	35.00	37.50	36.25	36.33	1.33
120	35	37.50	35.50	36.50		
121	40	40.50	42.50	41.50		
122	40	42.25	39.75	41.00	41.41	1.41
123	40	40.00	43.50	41.75		
124	45	46.50	45.50	46.00		
125	45	45.50	47.50	46.50	46.08	1.08
126	45	46.5	45.00	45.75		

Table 7: Results from the test samples (7)

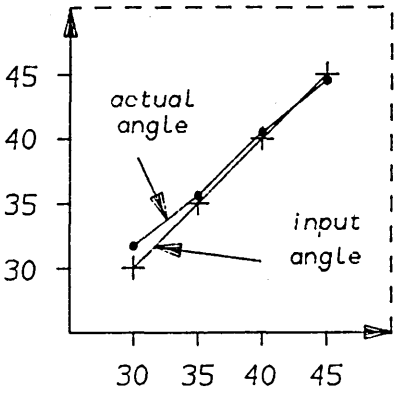
Sample Number	Input Angle (degrees)	Measured Angle (degrees)		Mean value for the two sides	Mean value for the three samples	Angle difference
		side 1	side 2	(degrees)	(degrees)	(degrees)
127	30	33.50	34.80	34.15		
128	30	35.00	32.50	33.75	33.80	3.80
129	30	33.25	33.75	33.50		
130	35	38.00	35.50	36.75		
131	35	34.50	37.50	36.00	36.00	1.00
132	35	36.00	34.50	35.25		
133	40	40.75	42.50	41.62		
134	40	43.70	40.00	41.85	41.74	1.74
135	40	40.00	43.50	41.75		
136	45	47.00	45.50	46.25		
137	45	45.00	48.25	46.62	46.45	1.45
138	45	48.00	45.00	46.50		
139	30	33.00	35.00	34.00		
140	30	35.00	32.50	33.75	33.83	3.83
141	30	32.50	35.00	33.75		
142	35	36.00	35.00	35.50		
143	35	35.00	36.50	35.75	35.83	0.83
144	35	36.50	36.00	36.25		
145	40	40.50	42.50	41.50		
146	40	42.00	40.50	41.25	41.00	1.00
147	40	39.00	41.50	40.25		
148	45	46.00	45.50	45.75		
149	45	45.50	47.50	46.50	45.95	0.95
150	45	46.50	44.75	45.62		
151	30	32.80	33.50	33.15		
152	30	32.50	31.50	32.00	32.75	2.75
153	30	32.25	34.00	33.12		
154	35	36.50	35.50	36.00		
155	35	35.00	37.50	36.25	36.16	1.16
156	35	37.00	35.50	36.25		
157	40	40.00	42.80	41.40		
158	40	43.00	40.00	41.50	41.30	1.30
159	40	40.00	42.00	41.00		
160	45	47.00	44.50	45.75		
161	45	45.00	47.00	46.00	45.92	0.92
162	45	46.80	45.25	46.02		

Table 8: Results from the test samples (8)

Actual
semi-cone angle
(degrees)

First Test

8 encoder puises

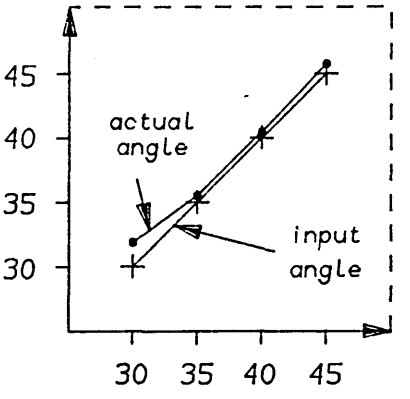


Input
semi-cone angle
(degrees)

Actual
semi-cone angle
(degrees)

First Test

16 encoder pulses



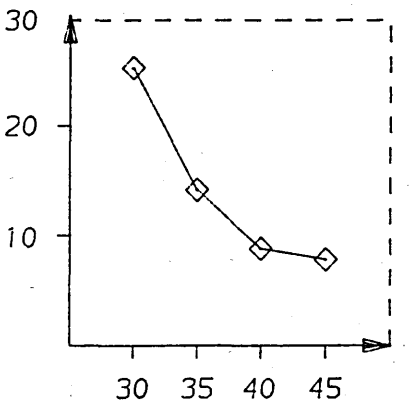
Input
semi-cone angle
(degrees)

Graph 1: The actual against the input cone angle

Reduction in thickness %

First Test

8 encoder pulses

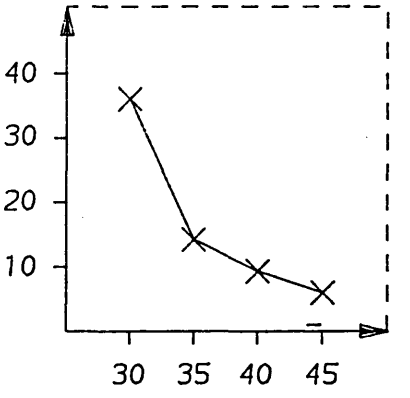


Semi-cone angle (degrees)

Reduction in thickness %

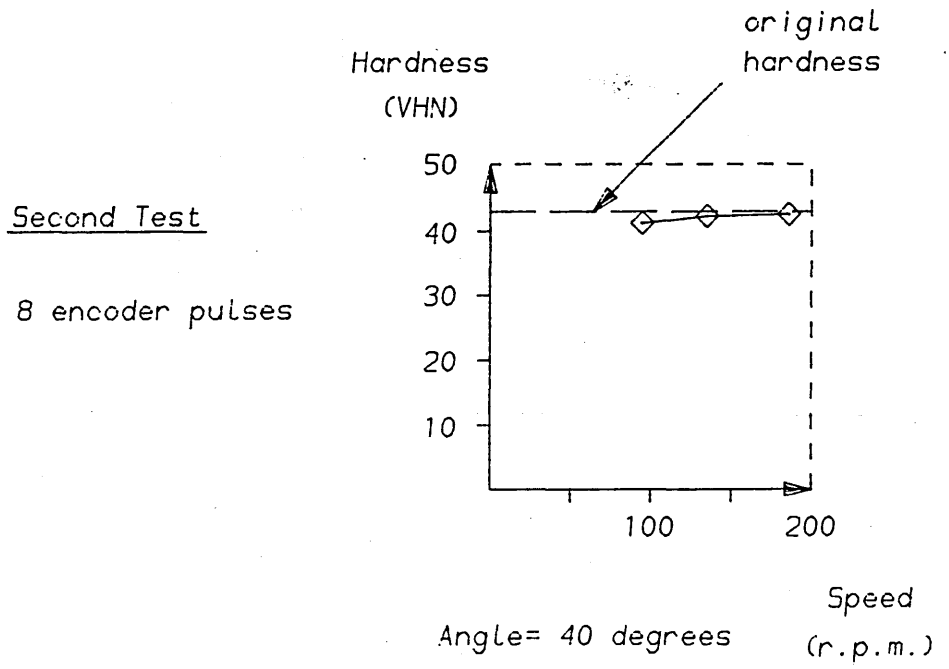
First Test

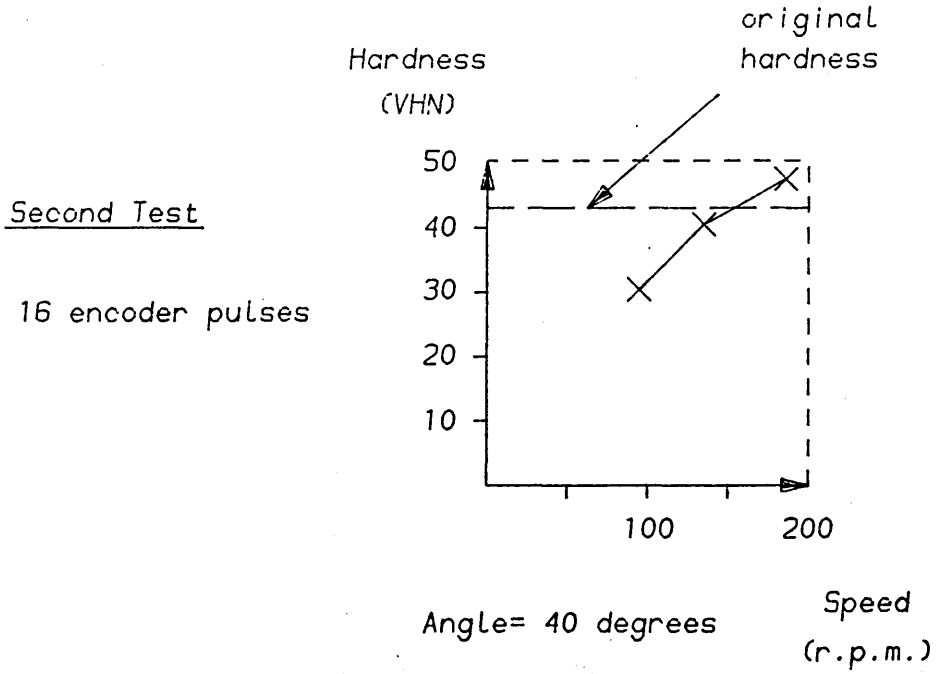
16 encoder pulses



Semi-cone angle (degrees)

Graph 2: The reduction in thickness against the cone angle

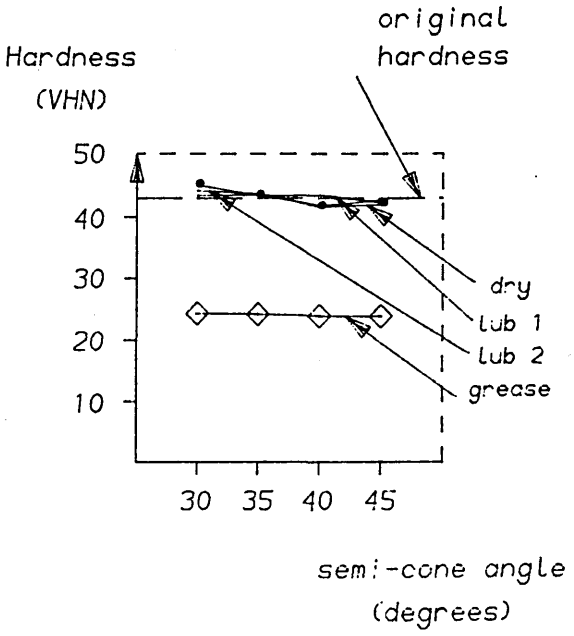




Graph 3: The hardness against the speed

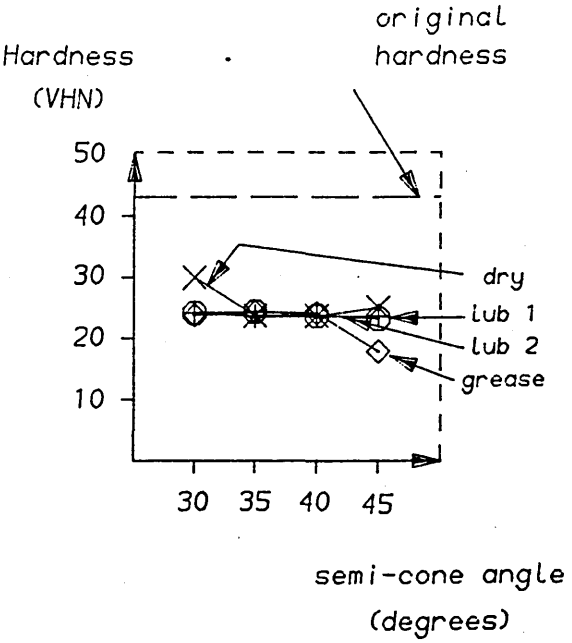
First and Third Tests

8 encoder puises

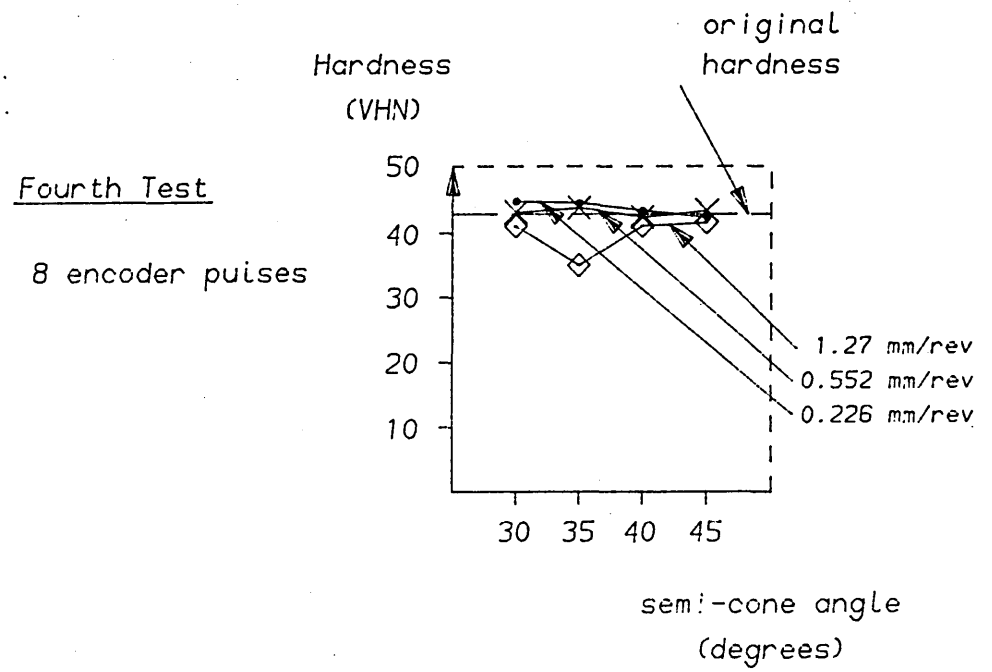


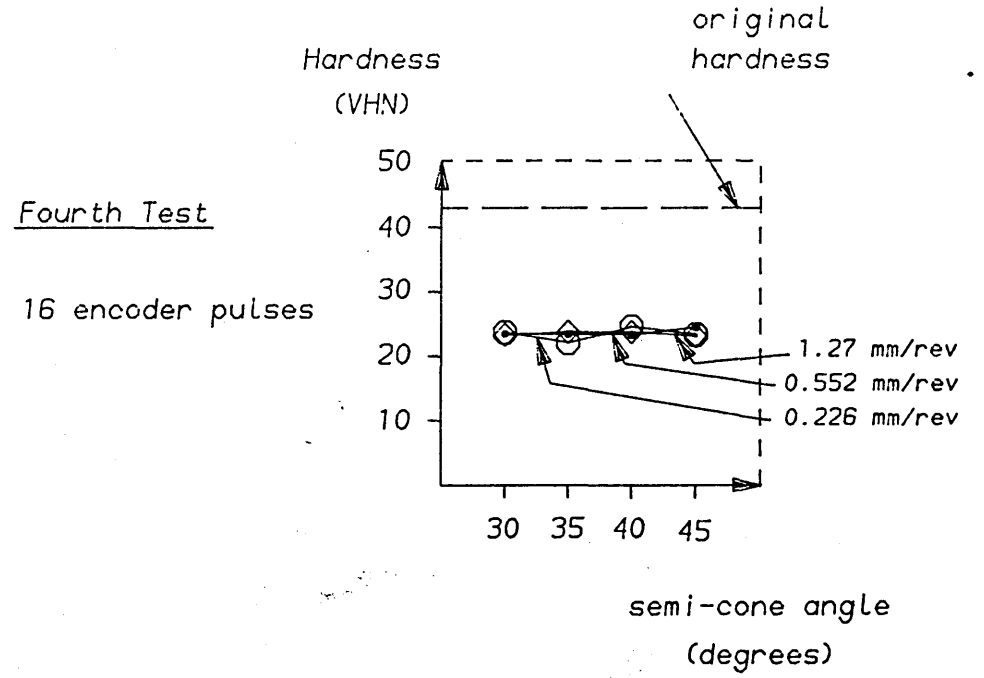
First and Third Tests

16 encoder pulses



Graph 4: The hardness against the cone angle with different lubricants





Graph 5: The hardness against the cone angle with different feeds