# Intelligent Control For The Centreless Grinding Of Compliant, Thin Walled Cylinders

S.K. Kelly, B.Eng

A thesis submitted as partial fulfilment of the requirements of Liverpool John Moores University for the degree of Doctor of Philosophy

This research programme was carried out in collaboration with The Liverpool University, GKN (Sheepbridge) and OSAI A-B

June 1994

**Acknowledgements**

I would like to thank the following for their help during the course of the work described in this thesis.

My supervisor Prof. W.B Rowe for his help, guidance and patience. My second supervisor Prof. J.L Moruzzi for his time, encouragement and helpful suggestions. Dr. Gibson for advice and support.

My co-workers on the project J.A Pettit, D. Allanson, P. Wright and Dr. S.R Terry for the long hours spent working with the machine.

SERC, GKN Sheepbridge and OSAI A-B for both support and finance during the course of the project work.

And finally, but not least, my parents, brothers, Denise Purser, Simon Moruzzi, Ruth Millard and Phil Adams for their constant ~~nagging~~ encouragement that led to the eventual presentation of this thesis.

## Abstract

An advanced system was developed for the intelligent, self-optimising computer numerically controlled grinding of compliant, thin walled cylinders. A study was made of previous developments in the adaptive control of grinding processes. Requirements imposed on the design of computer control systems by the implementation of adaptive control techniques were considered. A manually controlled centreless grinding machine was automated and the mechanical capabilities of the machine were extended so that productivity was limited by process rather than machine constraints. In-process measurement was implemented to allow a control system to account for variations in process conditions.

Experimental work performed with highly compliant workpieces exposed problems of process control that were accounted for by machine cycle design and process optimisation. It was considered that the improvements in productivity achieved by maximising material removal rates were limited for highly compliant workpieces. Therefore, a requirement for improving productivity through the design of appropriate machine cycles was identified. The design of a novel machine cycle eliminated the need for separate rough and finish grinding operations whilst allowing the control system to compensate for the high level of system compliance that was experienced. The benefits and disadvantages of the new machine cycle design were revealed by comparing the performance of the automated machine with that of manually operated machine tools.

A conceptual framework for an intelligent control system was created that combined process models, learning strategies, process measurement and adaptive control techniques with an economic data base. The conceptual framework was used as a basis for the development of strategies that improved productivity when grinding highly compliant workpieces by the centreless grinding process. A control system was developed which successfully demonstrated the benefits of the approach.

The control system was implemented within a commercial C.N.C. system and was put into production on the shop floor. A modular approach to the design of intelligent control systems for grinding processes is defined.

## Nomenclature

| | |
|---|---|
| $\alpha$ | Workplate angle (°), average heat transfer coefficient ($J/m^2/s/°C$) |
| $\alpha_s$ | Grinding wheel thermal diffusivity (m/s) |
| $\alpha_w$ | Workpiece wheel thermal diffusivity (m/s) |
| $\beta$ | Included angle (°), vertical control wheel skew angle (°) |
| $\partial$ | Horizontal control wheel skew angle (°) |
| $\lambda_s$ | Grinding thermal conductivity (W/mK) |
| $\lambda_w$ | Workpiece thermal conductivity (W/mK) |
| $\theta$ | Workpiece temperature rise (°C) |
| $\theta_m^*$ | Critical temperature (°C) |
| $\tau$ | System time constant (s) |
| $\omega(t)$ | Rotational speed (rad/s) |
| $\Omega$ | Cooled workpiece area ($m^{2)}$ |
| $a_r$ | Chip shape ratio (dimensionless) |
| a | Depth of cut (mm) |
| b | Wheel width (mm) |
| $d_s$ | Grinding wheel diameter (mm) |
| $d_w$ | Workpiece diameter (mm) |
| $d_e$ | Effective diameter (mm) |
| $e_c$ | Specific energy ($J/mm^3$) |
| $e_c^*$ | Critical specific energy ($J/mm^3$) |
| $F_n$ | Normal force (N) |
| $F_t$ | Tangential force (N) |
| k | Filter constant (dimensionless) |
| K | Percentage grinding energy entering the workpiece (dimensionless) |
| $K_t$ | Total system stiffness (N/μm) |
| $K_m$ | Machine stiffness (N/μm) |
| $K_w$ | Workpiece stiffness (N/μm) |
| $l_d$ | Grit spacing (mm) |

$l_g$ Geometric contact length (mm)

$l_e$ True contact length (mm)

$P$ Power (W)

$P_{max}$ Maximum machine power (W)

$P_{rms}$ R.M.S. power (W)

$R$ Workpiece energy partition ratio (dimensionless)

$t_g$ Total grinding time (s)

$t_d$ Dwell time (s)

$v_m$ Mean chip volume ($mm^3$)

$v_f$ Infeed rate (mm/s)

$v_{fp}$ Programmed infeed rate (mm/s)

$v_{fi}$ True infeed rate (mm/s)

$v_s$ Grinding wheel speed (m/s)

$v_{total}$ Total volume of material removed ($mm^3$)

$v_w$ Work speed (m/s)

$\Delta v_w$ Work speed increment (m/s)

$V_{rms}$ R.M.S. voltage (V)

$X_d$ Diameter reduction during dwell time (mm)

$x_i$ System deflection (mm)

$X_i$ True infeed position (mm)

$X_{OS}$ Overshoot infeed position (mm)

$X_T$ Target workpiece diameter (mm)

$Z$ Removal rate ($mm^3/s$)

$Z'$ Specific removal rate ($mm^3/mm/s$)

## Contents

# Chapter 1. Introduction

## 1.1 Project background

Workpiece quality and production rate vary due to changes in grinding conditions [1]. Grinding conditions vary primarily due to grinding wheel wear. Other factors affecting the process include the varying condition of the coolant, workpiece material and workpiece dimensions. The purpose of the project was to test the concept of an intelligent control system that responded to changes in grinding conditions through incorporation of suitable control strategies. The basic structure of an intelligent control system was proposed by Rowe [2] and a schematic diagram of this structure is given in Figure 1. The proposal for a control system structure was referred to as the 'conceptual framework'.

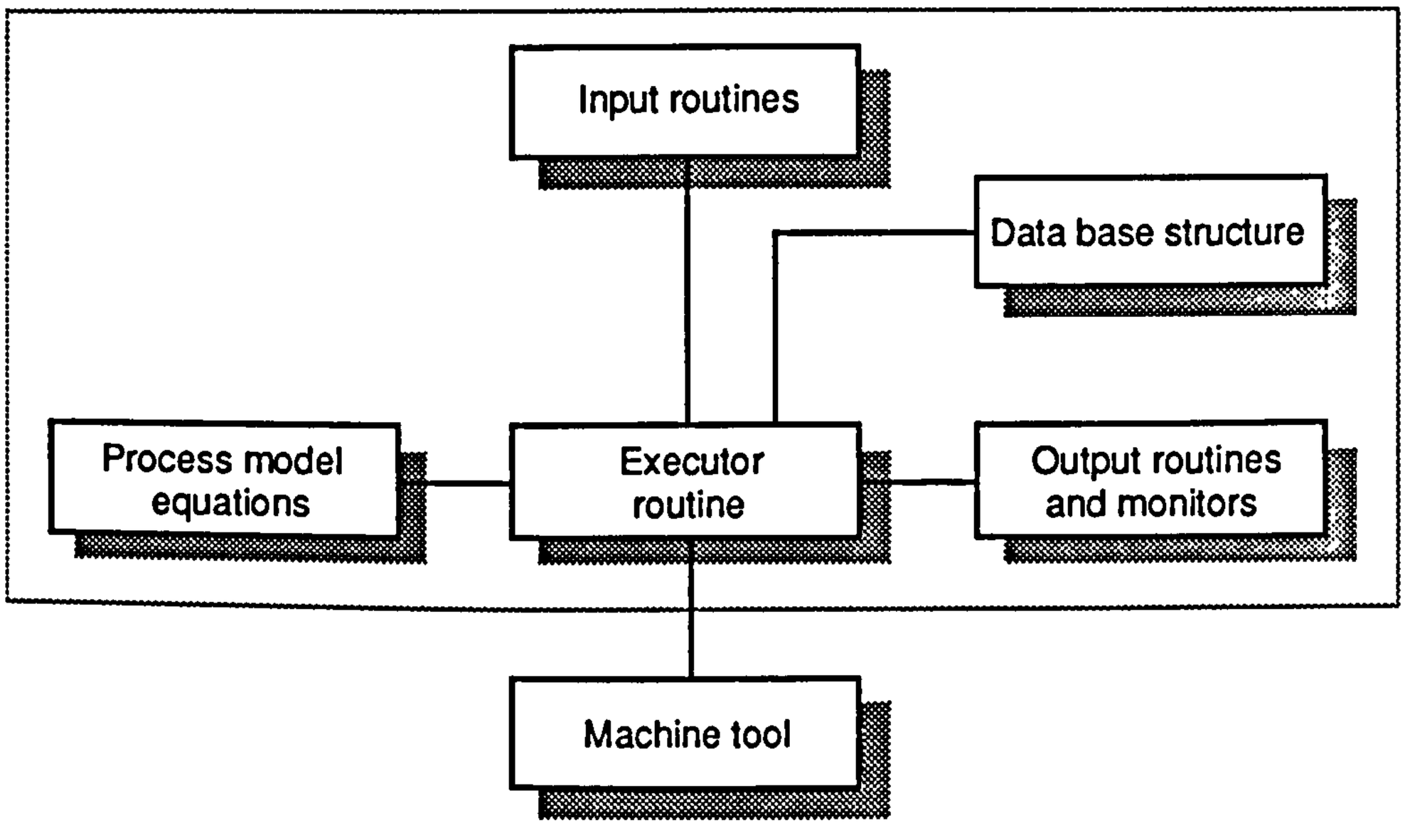



Figure 1. Preliminary outline of the conceptual framework

The conceptual framework provided a starting point for the development of ideas and concepts that related to intelligent control systems. A template for the design of intelligent control systems was provided by the module framework definition. A key feature of the project proposal was the requirement for the integration of a data base with process modelling, real-time data acquisition, learning strategies and adaptive

control. Specification of a communications protocol for control system modules was required in order to simplify development and testing of new modules.

Other workers have investigated the application of adaptive control techniques to metal cutting processes in order to provide process control and error compensation [3-4]. Improvements in grinding efficiency, workpiece size and form accuracy were benefits claimed from adaptive control of the grinding process [5-11]. A variety of models of the grinding process have been proposed as a basis for adaptive control. Gao [12] suggested that these models were based on either physical laws [12-15] or observed grinding data [16]. However, there has been little consideration given to the creation of a modular control system framework that permits flexible integration of new or improved process models and adaptive strategies.

The industrial collaborator, GKN Sheepbridge, is a large scale manufacturer of cast iron cylinder liners for use in the automotive industry. The external and internal diameters of the cylinder liners were rough turned before separate rough and finish grinding of the external diameters on centreless grinding machines. Due to the large element of the manufacturing process that incorporated centreless grinding and the number of centreless grinding operations required it was reasoned that productivity would be greatly improved by the proposed control strategies. The thin walled cylinder liners produced by GKN were highly compliant and thus the manufacturing process provided a demanding test for the developed strategies. It was also found that the lathes used in the rough turning process permitted large variations in size and compliance between individual workpieces prior to grinding.

GKN provided a manually controlled Cincinnati No. 3 centreless grinding machine for the project of the type on which the cylinder liners were currently produced. Experience within the University had been gained previously from the mechanical development of a Wickman research machine tool for high speed grinding. This experience formed the basis for believing that significant improvements in productivity could be achieved.

As GKN intended to use the developed machine tool for production it was essential that the control system was capable of performing reliably in a hostile shop floor environment and that support was available at the end of the three year project. Commercial control system equipment to industrial specification was therefore selected.

OSAI A-B supplied two 8600 MC/TC C.N.C. systems for the project. The 8600 C.N.C. was a general purpose machine tool control system. Such control systems are fitted to a large number of C.N.C. controlled lathes and milling machines. One 8600 C.N.C. system was provided for the Cincinnati No. 3 machine tool. The second 8600 C.N.C. system was provided for the Wickman machine and for the development of control system software.

## 1.2 Aims and objectives

The aim of the project was to design and develop an intelligent control system structure employing strategies that could be applied to a range of grinding processes. The control strategies were to be selected with a view to optimising process productivity. Advantages arising from the integration of process modelling and real-time data acquisition with learning features were to be investigated. Particular requirements were to compensate for changes in grinding conditions arising from variations in workpiece material and grinding wheel wear. It was required to determine whether an intelligent control system could improve productivity over that achieved by a manually controlled machine tool whilst maintaining or improving standards of quality. A constraint on the control strategies to be employed was that any machine control and monitoring systems employed were to be economically justified.

## 1.3 The scope of the investigation

A modular control system for the intelligent control of grinding processes was designed and implemented. The control system was used for the development of appropriate strategies for improving productivity when grinding highly compliant workpieces by the centreless grinding process. The work formed part of an SERC collaborative research project involving The Liverpool John Moores University, The University of Liverpool, GKN Sheepbridge and OSAI A-B.

A study was made of previous developments in adaptively controlled grinding. Decisions were taken as to the relevance of existing technology for general application to grinding processes including the particular case of the centreless grinding process. The specification of the computer control system was considered. Of particular importance was the ability to integrate the intelligent control system structure within the computer control system. It was concluded that to be successful the control system must be capable of being embodied in a product that could be commercially supported world wide.

From a consideration of the requirements of the centreless grinding process and available technology, a basic conceptual framework was proposed for the intelligent control system. In the light of experience gained during the project this concept was further developed and forms a basis for transfer of the technology to other grinding processes. The essential attributes of a conceptual framework are discussed and the framework is used as the basis for the design and implementation of the intelligent control system developed. The conceptual framework was considered and defined in terms of the essential elements. Further developments were proposed. A substantial part of the work was concerned with structuring the system elements so that the strategies could be accommodated within the capability of the controller. A set of priorities was established for the various controller functions. This set provides a

guide that can be applied by future designers of intelligent controllers for grinding process control.

The experience gained from earlier work on the Wickman research machine was used to extend the mechanical capabilities of the GKN machine so that productivity was limited by process rather than machine constraints. The machine modifications included the provision of a high power grinding wheel drive, a variable speed control wheel drive and a servo-controlled infeed mechanism. GKN also provided a large number of cylinder liners to allow system trials to be performed on workpieces of the type manufactured.

Initially the two machine tools were re-wired and interfaced to the computer control system. System trials were undertaken at subsequent stages of system development. Experimental work exposed problems of process control that were largely overcome by appropriate feed cycle design and optimisation for highly compliant workpieces. To compensate for the large deflections experienced when grinding cylinder liners an assessment was made of possible approaches using a combination of compliance modelling and data acquisition techniques. A pecking cycle was proposed and found to be successful in eliminating the need for separate rough and finish grinding operations. The pecking cycle also made it possible to overcome the problems of in-process size measurement and in-process temperature rise. The machine tool and control system were introduced to the shop floor. An evaluation of the automated machine performance revealed the benefits and disadvantages compared to manual operation.

Technological advances achieved in this work include identification of the requirements for a commercially successful, intelligent automation system and the translation of a basic conceptual framework into an essentially realisable structure for centreless grinding of highly compliant workpieces. These advances required an analysis of control system requirements and a specification of functional priorities. Furthermore, a novel approach to the design and implementation of a feed cycle

capable of coping with the technological difficulties of in-process size measurement and temperature growth of the workpiece was produced. Implementation and evaluation of the above principles were performed in a practical system.

# Chapter 2. Review of the centreless grinding process

## 2.1 Centreless grinding machine geometry

Generally, it is necessary to clamp a workpiece to the machine tool on which it is to be machined. Therefore, a computer control system is able to control the size of a workpiece by positioning the machine tool axes relative to a machine datum position. In the case of cylindrical grinding machines the workpiece is supported between centres and the relationship between the machine datum position and the centre of the workpiece is fixed.

However, on a centreless grinding machine the workpiece is supported by a tungsten carbide tipped work support plate and a rubber control wheel. The control wheel holds the workpiece against the grinding wheel surface and the surface speed of the control wheel governs the speed of workpiece rotation. Grinding and control wheel speed and direction are represented by '$v_s$' and '$v_w$' respectively. A diagram of the relationship between grinding wheel, control wheel, workpiece and workplate is given in Figure 2.

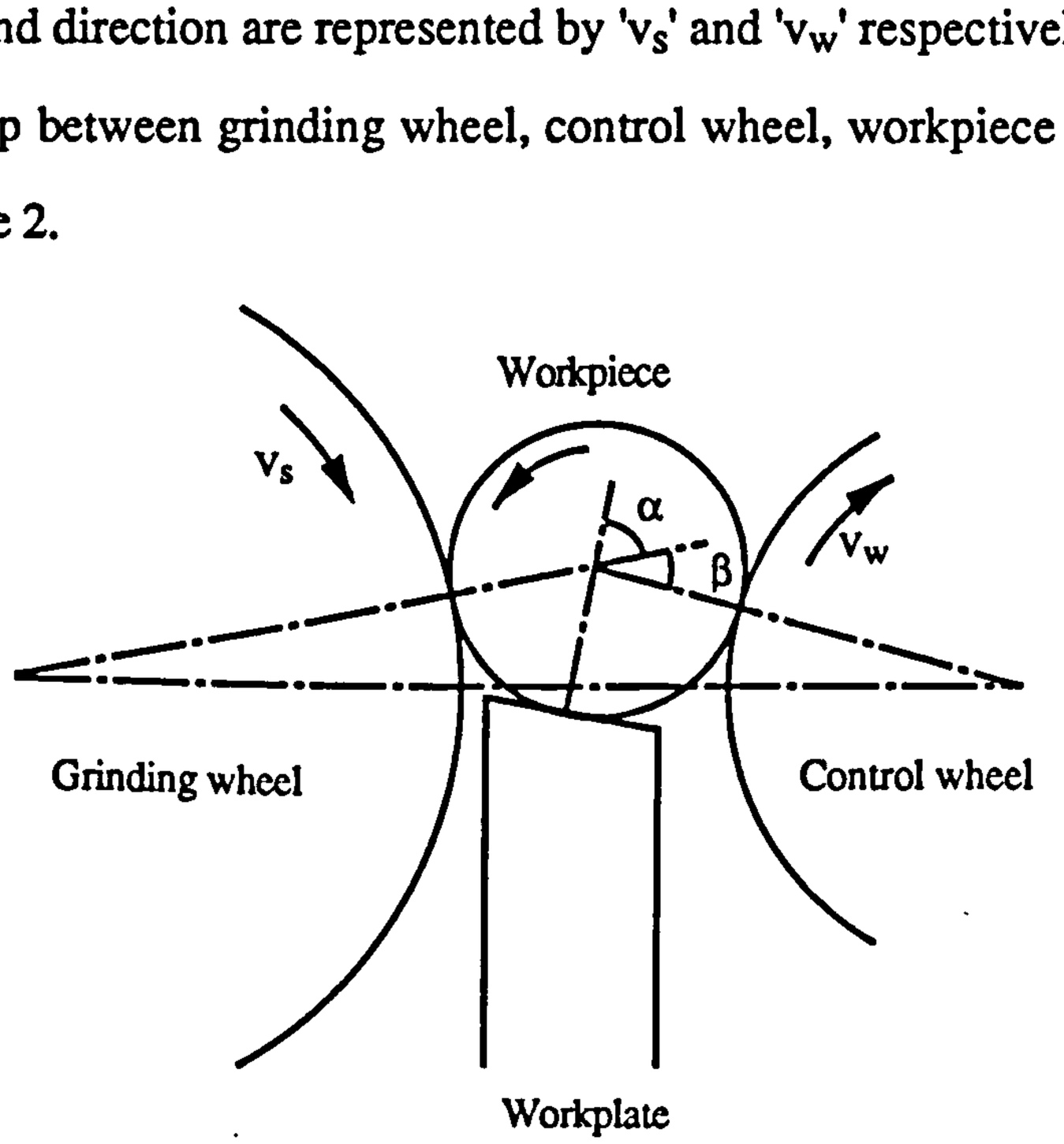



Figure 2. A schematic diagram of the centreless grinding machine arrangement.

The position of the workpiece centre depends on the relationship between the grinding wheel, control wheel and workplate. The centre position of the workpiece changes during a grinding cycle because material is removed from the workpiece and from the grinding wheel. Also, deflection of the workpiece resulting from the workpiece compliance and grinding forces causes the workpiece centre position to change. There is no simple technique for deriving the nominal position of the workpiece centre relative to a machine datum position. Therefore, there is no reliable machine datum position from which a control system can accurately control workpiece size. Consequently, there is a requirement for in-process gauging of workpiece diameter. Kaliszer [17] concluded that in-process measurement of workpiece diameter was required to provide a control system with the ability to adjust and correct process parameters. Also, Rao [18] suggested that it was possible to derive grinding wheel wear and system time constant from in-process workpiece size measurement.

In comparison with machine tools that require a workpiece to be mounted between centres there are three main advantages available to users of centreless grinding machines. Firstly, the loading and unloading time is typically very low and is easily automated. Secondly, hollow workpieces may be produced that would otherwise require an insert to allow clamping between centres. Finally, the centreless grinding process is amenable to high material removal rates. The support provided by the control wheel and the workplate allows heavy cuts that would deflect or damage a workpiece that relied on centres for positional location and support.

Workpiece roundness is affected by the workplate arrangement due to geometrical effects on the stability of this highly complex process. If the machine geometry causes instability, lobes can appear on the workpiece surface. Although the process by which a particular machine geometry either encourages or corrects lobing is reasonably well understood, the result of the process is not easily predictable. The effects of system deflections, machine setting and grinding conditions on workpiece roundness for grinding processes is well documented [1,8,19,20]. Rowe [21,22]

analysed the roundness characteristics of the centreless grinding process and proposed a computer simulation for predicting workpiece roundness [14]. However, general rules for setting the geometry of a centreless grinding machine to avoid roundness problems are well documented [14,23-25]. The geometry of the process is determined by workpiece centre height, workplate angle and work speed. The height and position of the workplate is usually adjusted manually according to the workpiece dimensions by a machine setter. Careful machine setting ensures that the workpiece centre is above the centres of the grinding and control wheels. It was shown [14,24] that a workplate angle '$\alpha$' of 30° and an included angle '$\beta$' between 6° and 8° was suitable for most purposes.

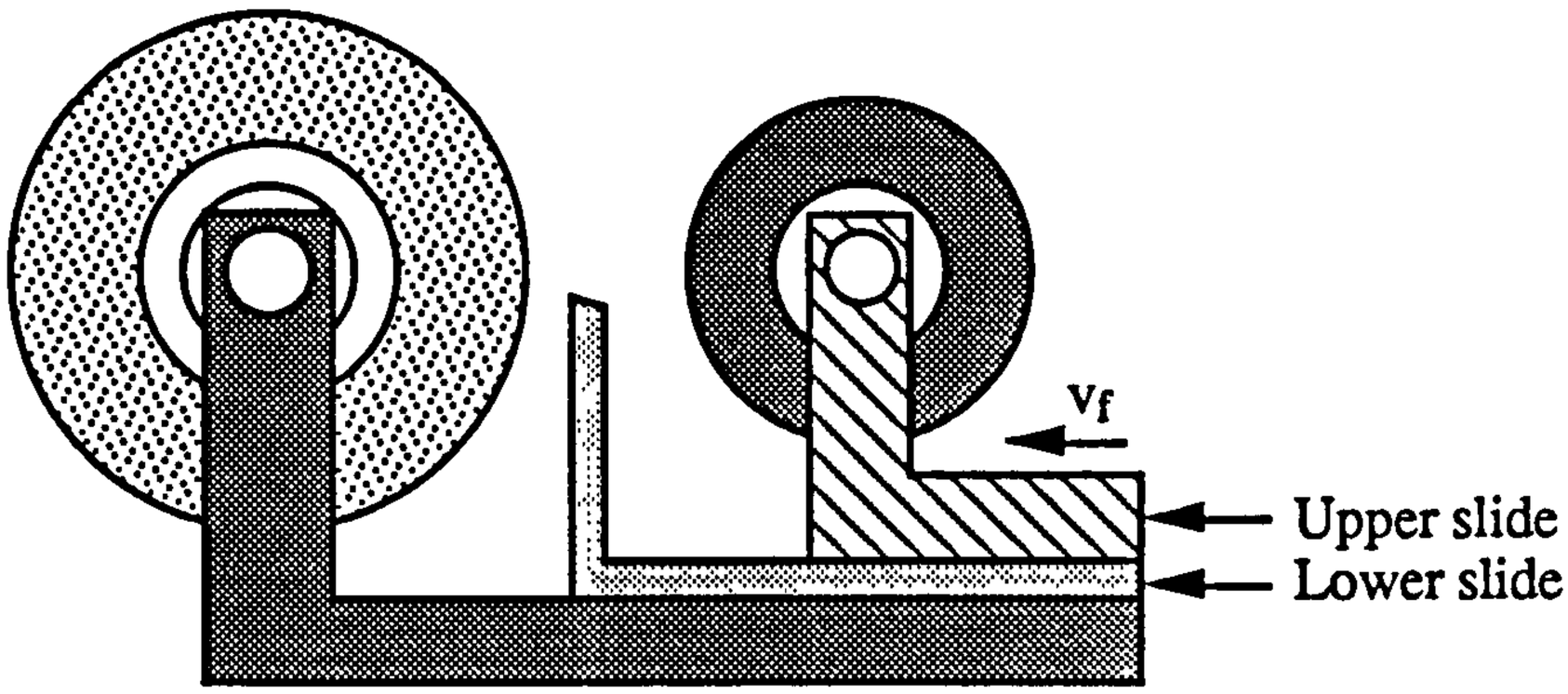



Figure 3. Schematic diagram of the Cincinnati No. 3 slideway arrangement.

In common with the design of most centreless grinding machines, the Cincinnati No. 3 supplied by GKN featured the compound slideway arrangement illustrated in Figure 3. The control wheel was mounted on an upper slideway that was itself carried on the lower slideway supporting the workplate. Conventionally the lower slide supporting the workplate is clamped in position and the infeed mechanism positions the upper slide so that the advancing control wheel rolls the workpiece up the workplate until it meets the grinding wheel. The infeed rate of the control wheel is represented by '$v_f$'. The geometry between the workpiece, grinding wheel and control wheel changes as the workpiece rolls up the workplate. The variation in geometry during a grinding cycle causes difficulties for in-process measurement of workpiece size.

## 2.2 The grinding mechanism

A major factor affecting the performance of the grinding process in terms of workpiece quality and production rate is the type and condition of the grinding wheel. In order to maintain the required process performance a control system must compensate for the effects of grinding wheel wear and cater for different grades of wheel composition. The principal effects of the grinding wheel type and condition on the grinding process are on workpiece surface roughness and material removal rate. A grinding wheel is selected according to the requirements of the particular process.

Grinding wheels used in the centreless grinding process comprise a large number of small, extremely hard and brittle grits held in a matrix of bonding material and separated by pores. The bonding material encases individual grits and forms bond bridges that separate adjacent grits. Pores between grits provide clearance for the coolant and the metal chips produced by the grinding action. The grade of a grinding wheel is referred to as 'hardness' and is a measure of bond strength and wheel durability. If the amount of bonding material is increased, the thickness of bond bridges increases and the size of pores between grits reduces. Thicker bond bridges hold grits together more rigidly leading to a harder wheel grade.

To generate a cutting action from the grits the grinding wheel is rotated at a high speed. During the grinding cycle the workpiece is forced against the grinding wheel and the cutting action of the grits causes material to be removed. Each active grit on the wheel surface acts as a small single point tool removing a chip from the workpiece surface every revolution of the wheel. A grinding wheel rotating at 30 rev/s expends a large amount of energy as this causes the production of 30 workpiece chips per second per grinding wheel grit. Chips formed by this action are typically slightly smaller than the grits causing them [26]. Aluminium oxide grits vitrified or bonded in resin are the most common form of grinding wheel material. However, both composition and bonding technique are varied to suit the workpiece material.

Extended use blunts individual grits causing the abrasive surface to dull. A degree of self-sharpening occurs as friction, heat and forces increase [27] until a new cutting edge is formed. The new cutting edge is produced as either the grit fractures or becomes dislodged to uncover a fresh grit. Tough grits used for high material removal rates tend to dull until the grit dislodges. However, the point at which a grit becomes dislodged depends on the bond strength. The ability of a wheel to resist wear and remove material also depends on grit type, size and spacing. Hard grinding wheels tend to have small fine grits with tough bonds that give wear resistance. These types of wheels are normally used when high quality surface texture is required and material removal rate is not critical. Conversely, soft wheels with large grits and weak bonds are used for high removal rate applications. Although a soft wheel will wear more rapidly than a hard wheel, material may be removed at a higher rate and with less power consumption.

A grinding wheel is normally sharpened by dressing the wheel surface with a diamond tipped tool. Grits are either sharpened or fresh grits are exposed by drawing the sharp diamond tip across the width of the rotating grinding wheel surface. The power and forces whilst grinding depend on a statistical average of the numbers of dull and sharp grits at a given point in time.

In certain cases the rate of grinding wheel wear is of a similar magnitude to the material removal rate. It is therefore necessary for a computer control system to adjust the infeed position according to the reduction in grinding wheel size in order to maintain workpiece size. In-process measurement of the workpiece may be used in computer controlled grinding machines to account for small amounts of wheel wear [17,18].

## 2.3 Effects of compliance

The majority of computer control systems fail to make provision for machine and workpiece compliance. For a rigid machine tool and workpiece, removal rate is proportional to infeed rate and workpiece size is directly related to infeed position. However, machine tools and workpieces are not infinitely stiff and deflections arise due to cutting forces [16,28,29]. Machine elements that determine rigidity include the machine bed, slideways, spindles and spindle bearings. Careful machine tool design reduces compliance and the use of optimum grinding conditions minimises the cutting forces that cause deflections. Figure 4 is a schematic representation of the sources of compliance in the centreless grinding process.

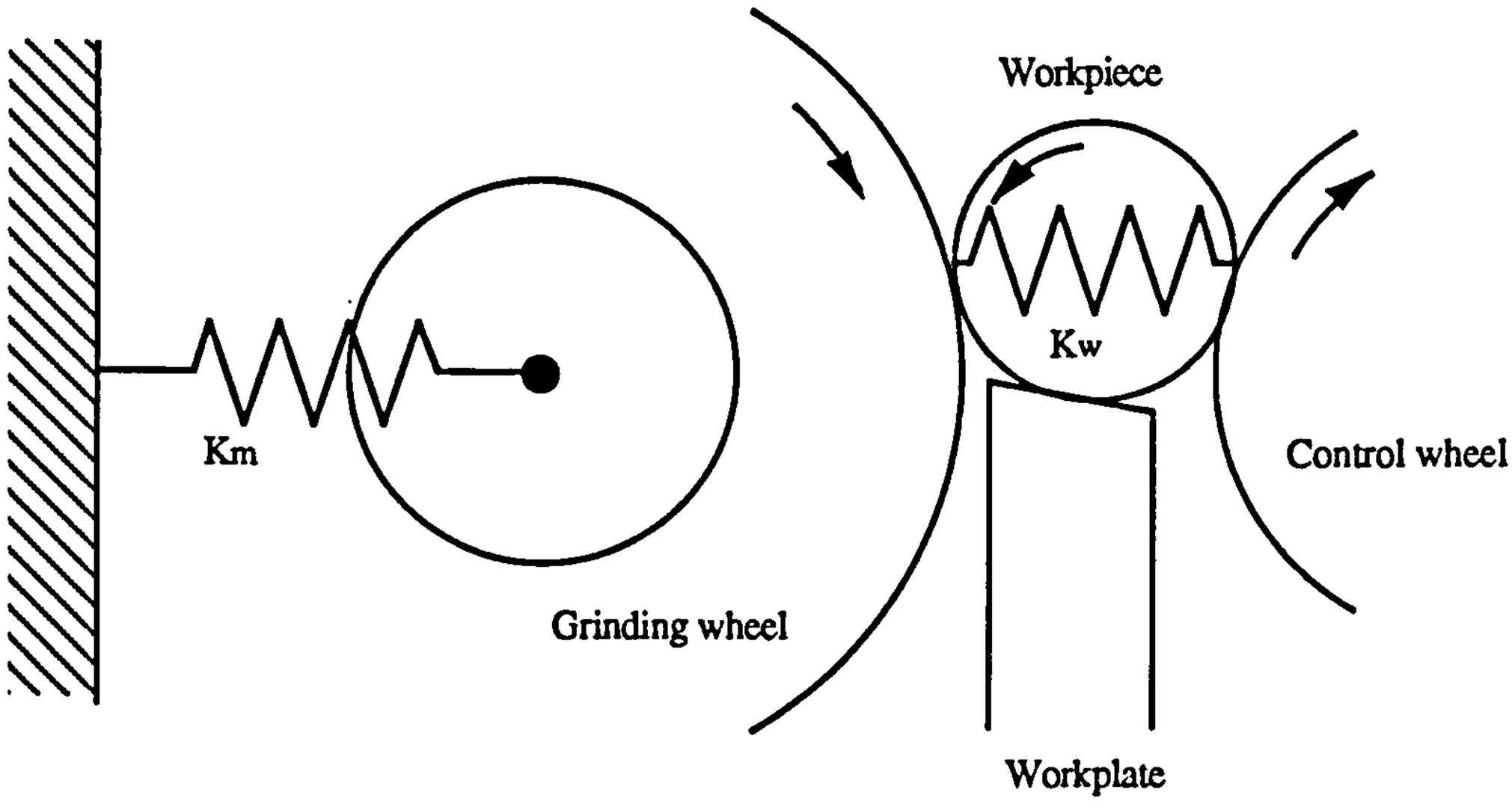



Figure 4. Sources of compliance for the centreless grinding process.

Compliance leads to the requirement for a dwell or 'spark out' period in the grinding process. Use of a dwell time at the end of the infeed cycle allows machine and workpiece deflections to relax as grinding forces fall. An increase in compliance causes greater deflection and requires longer dwell times in order to ensure size and roundness accuracy. Trmal [16] indicates that the amount of deflection experienced for a given value of system compliance depends upon the grinding force. Grinding

conditions change over time as the grinding wheel wears [1] and system deflection varies due to the resultant change in grinding forces. An adaptive control system is therefore required to account for varying system deflections whilst maintaining satisfactory production rates.

## 2.4 Process capability

The centreless grinding machine is suitable for high volume, high precision production of workpieces in a wide variety of materials [27]. Typically metal, glass and ceramic workpieces are produced on such machines. Understanding of the grinding process is available to produce adequate quality for most workpieces. However, the rate of material removal is often low in comparison to alternative processes such as turning and milling. Generally, modern manufacturing industry requires that both the accuracy and production rate of workpieces are continually improved. This requires improvements in manufacturing techniques. Often such improvement involves the incorporation of computer control systems.

The purpose of grinding is normally to produce a workpiece with a higher degree of size accuracy, geometric accuracy and surface quality at a lower cost than is possible with other manufacturing processes. Size accuracy relates to the variations in workpiece diameter achieved whilst geometric accuracy primarily relates to roundness and straightness tolerances. Centreless grinding is capable of high degrees of geometric and surface texture accuracy. For example, Miyashita quoted values of 0.2 μm roundness and 0.12 μm Ra surface roughness [30]. It is not unusual to achieve a size accuracy and repeatability of 2.5 μm. However, increasing quality requirements may have a significant detrimental effect on the time taken to produce a workpiece.

Workpiece size accuracy depends on a number of factors including the positioning capability of the infeed mechanism. With suitable axis drives and positional feedback it is possible for a modern computer control system to position machine tool axes with a resolution of less than 1 μm. However, size accuracy is

affected by compliance of the machine elements and the workpiece. Also, thermal distortion of the machine tool introduces size control difficulties and errors occur due to thermal expansion of the workpiece during the grinding cycle. An adaptive control strategy was therefore required which would compensate for these factors.

A tapered workpiece is produced if the control wheel is not set parallel to the grinding wheel when viewed from a plan view. Adjustment of the control wheel angle is performed either by an operator or, in some cases, by the machine tool control system. However, taper may be produced on compliant workpieces for reasons other than machine setting. The compliance of a workpiece may not be constant over its length due to variations in wall thickness. Therefore, taper results if grinding force and hence material removal varies over the length of the workpiece. Taper may be reduced by adjusting control wheel angle so that grinding forces are equalised over the length of the workpiece.

Experimental data from the grinding of cast-iron workpieces such as those produced by GKN was scarce in comparison to the large volume of data relating to the machining of high-speed steel. However, King [31] suggested that cast-iron was an easy to grind material when compared with high-speed steel. To achieve the equivalent of cast-iron material removal rates with high-speed steel required approximately four times as much grinding force. Hahn [32] indicated that specific material removal rates of approximately 12 $mm^3/mm/s$ were possible for high-speed steel. It was therefore possible to deduce that specific material removal rates in the region of 48 $mm^3/mm/s$ were possible for cast-iron. Interpreting the limit chart for cast-iron proposed by Rowe [33] confirmed this deduction as specific material removal rates of approximately 37 $mm^3/mm/s$ were achieved with appropriate grinding parameters.

# Chapter 3. Previous work directed towards intelligent control

Whilst a wide range of computer control systems are dedicated to milling and turning processes there has been limited commercial application of such systems to the grinding process [34]. Also, although there have been many investigations into the adaptive control of machining processes, adaptive control has not been applied previously to the centreless grinding process. However, work has been performed in developing techniques for intelligent control of other grinding processes. It was necessary to consider the relevance of this work to the centreless grinding process.

## 3.1 Control of grinding processes

Much research [4-6] has been concerned with adaptive computer control systems that were designed to maximise material removal rates. Improvements in machine tool design [34-37] have increased rigidity and material removal rates. However, despite these advances many centreless grinding machines remain manually controlled. This may be due to the fact that current automation systems for the centreless grinding process do not significantly reduce costs in many areas of work. Consequently, the sources of process variability and process optimisation techniques for manually controlled centreless grinding machines were considered prior to the design of the intelligent grinding control system.

Typically, the following process features are considered to be relevant in relation to workpiece quality:

1. Workpiece size variability
2. Workpiece surface texture
3. Noise level
4. Chatter occurrence
5. Power and force levels
6. Temperature levels

A manual machine operator produces workpieces of satisfactory quality by continually adjusting grinding parameters and monitoring the process conditions. An operator with skill and experience is therefore required to produce satisfactory results from a manually operated grinding machine. However, operators cannot sustain high levels of output and productivity is lower than theoretically possible. Although process optimisation was considered too complex a procedure to perform intuitively, work performed by Shaw [10] suggested that it was not practicable to optimise the centreless grinding process without an operator. This premise resulted from recognising the need for in-process measurement of machine parameters. However, suitable sensors were not commercially available. Shaw proposed that an operator entered observed changes in process conditions into a programmable calculator. A set of pre-programmed rules within the calculator suggested suitable alterations to the controlled grinding parameters. This approach provided a manual adaptive control system. The technique developed was based on the optimisation of work speed in the pursuit of minimum cost per workpiece.

An off-line process optimisation system for centre and internal grinding proposed by Malkin [38] optimised grinding and dressing parameters with a program running on a personal computer. The objective was to achieve maximum material removal rate within the constraints of workpiece burn and surface texture. The computer program derived new values of grinding parameters and dressing conditions from the operator's recordings of surface texture and grinding power. Effects of high material removal rates on grinding wheel wear were also considered with a view to either minimising costs or maximising production rate. Malkin suggested optimum grinding conditions involved a compromise between shorter grinding cycle times and more frequent wheel dressing arising from increased removal rates. Thus maximum material removal rate was not necessarily the optimum target for an adaptive control system. Increased removal rates were achieved by further work [6] that applied these techniques to an on-line adaptive computer control system. The developed system comprised a PDP-11/40 mini-computer interfaced to a cylindrical grinding machine.

Process optimisation was achieved through adaptive control of infeed rate in response to measured grinding power.

Malkin's concept of an adaptive control system that provided process optimisation was unusual in that optimisation targets were not fixed but varied according to the particular process. It was considered that this concept was particularly valuable in the design of a generic, intelligent control system. Therefore, a conclusion derived from this work was that a modular control system structure should provide the machine tool user with a selection of suitable optimisation strategies.

The grinding process is widely regarded as a finishing process [27] and hence accuracy is often the prime requirement. Forces during the grinding cycle lead to wear of the grinding wheel and a reduction in wheel diameter. For many grinding processes the effect on workpiece size accuracy of wheel diameter reduction is significant. Although this problem is less important in centreless grinding where large diameter wheels are used, accuracy is important and methods are needed to compensate for changes in grinding conditions that arise from wheel wear. In-process gauging systems are used to maintain workpiece size within close tolerances despite grinding wheel size reduction [17,18]. Gap elimination techniques permit reduced cycle times by removing the non-productive grinding time at low infeed rate that is otherwise necessary with a grinding wheel of indeterminate size. Automatic wheel changing and dynamic balancing are sometimes used in order to reduce manning levels and down time [34]. The application of computer control to grinding machines has become expensive due to the cost of these and other requirements. Also, technological advances in tooling for use on milling machines and lathes led to improvements in surface texture and accuracy that in some cases eliminated the need for finish grinding.

A control system that relied on in-process measurement of force from a dynamometer mounted on a specially built cylindrical grinding machine was proposed by Stoten [39]. The aim of this work was to determine the 'grindability' of various materials including tungsten carbide and silicon nitride. System identification was used

to produce transfer functions for force and infeed response that were implemented on a personal computer based control system. Stoten determined gain constants for a 'proportional plus integral' controller that allowed constant force grinding to be achieved by continually varying the infeed rate. The performance of the system was responsive and stable for a variety of workpiece materials. Although the closed loop control approach did not provide a true adaptive system, valuable data concerning the grinding of ceramics was collected. The requirement for a special purpose machine and force dynamometer inhibited commercial adoption of the ideas proposed.

Work performed by Hahn [29] identified normal force and grinding wheel sharpness as two of the most important variables in the grinding process. This was due to the effect of normal force and wheel sharpness on the following variables:

1. Material removal rate
2. Instantaneous surface texture
3. Incidence of thermal damage
4. Wheel wear rate
5. Machine deflection

Hahn noted that system deflection caused problems in the size, roundness and taper of internally ground fuel injection nozzles. It was proposed that control of normal force would allow a system to compensate for system deflection. A degree of artificial rigidity was achieved by using adaptive techniques to adjust the wheelhead position in response to changes in measured normal force. The wheelhead advanced to compensate for the grinding wheel deflection. Hahn considered that monitoring wheel sharpness made it possible to avoid thermal damage. The research suggested calculation of the work removal parameter as a quantitative way of measuring grinding wheel sharpness in real time. Work removal rate was defined as the rate of change of material removal rate with normal force and was measured on-line by a computer control system.

Various researchers detailed the potential benefits of controlled force grinding [29,32,39-41] where force was controlled either after each workpiece revolution or dynamically. The first approach was pursued because dynamic force control systems that reacted to instantaneous peaks do not correct workpiece roundness errors. Controlled force grinding of the centreless grinding process was discussed by Romanov [40]. Romanov devised a range of operating conditions for which controlled force grinding was claimed to correct initial workpiece out-of-roundness from 20% to 50% faster than a conventional fixed infeed machine.

A computer controlled adaptive force system was developed by Tönshoff [41] for the internal grinding process. The aim of the control system was to produce accurate bearing rings from workpieces that were initially out of round with reduced manufacturing costs. This was achieved by reducing non-productive time by gap elimination and optimising infeed rate. Gap elimination was used to switch from rapid approach to controlled feed after detecting an increase in force when the grinding wheel contacted the workpiece. A force control algorithm was developed that varied infeed rate in order to achieve constant average force per workpiece revolution. The force control algorithm relied on an identification cycle to determine control coefficients. As digital control systems can compensate for large dead times Tönshoff suggested that it was possible to achieve similar results more economically using grinding wheel power measurement as opposed to force. Clough [64] proposed such a scheme for in-line adaptive dead time compensation.

A variety of plunge grinding problems were studied by Kaliszer [9,17,42] in the development of a multi-processor control system. Kaliszer proposed a modular, bus-based adaptive control system for the grinding process [42]. The aim of the research was to minimise machining costs for a given workpiece specification. The bus-based approach to computer control system design provided great flexibility for the specification of suitable input and output (I/O), memory requirements and processing power. In-process measurement of normal force, workpiece size, roundness and

surface texture were incorporated as feedback to the control system. Also included was an automatic grinding wheel balancing system driven by analog signals from a wheelhead mounted vibration sensor. In-process measurements were used by the control system to determine suitable grinding conditions and to modify the infeed cycle. This work was of particular relevance to the current project as it addressed problems of system deflection by modifying infeed cycles according to measured compliance. It was considered likely that such techniques would be useful for the development of the control strategy for the grinding of the highly compliant cylinder liners.

Investigations into a number of key areas were also carried out at Liverpool Polytechnic [33,35] prior to the start of the project on which this thesis is based. The main purpose of studies by earlier researchers was the pursuit of high material removal rate grinding. The emphasis of the work performed involved the development of operation strategies suitable for use in computer control systems and was to form the basis of intelligent control system strategies.

Limit charts [33] were the result of an investigation into high rate grinding of steel and cast-iron. The controlled grinding variables of infeed rate and work speed provided the limit chart axes. This was to be achieved using values of parameters that were derived experimentally. Typically, a limit chart illustrated the process boundaries represented by workpiece burn, chatter and the available grinding wheel power. It was suggested that as the optimum grinding condition was within the region enclosed by the boundaries, knowledge of the shape of the boundaries would allow a computer control system to optimise the process. This conclusion was consistent with a study of the optimisation process which found that specific energy should be minimised for maximum material removal rate and avoidance of thermal damage. The study also showed that when centreless grinding ferrous materials with aluminium oxide wheels, the optimum wheel speed lay in the region 45 m/s to 60 m/s. Limiting wheel speed to

45 m/s eliminated the need for high velocity coolant delivery. Kinematic analysis was proposed to duplicate optimal grinding conditions for workpieces of varying size.

Much of the work undertaken at Liverpool in the identification of both kinematic parameters and limit charts was performed on a Wickman 2K centreless grinding machine. The machine featured high stiffness hydrostatic bearings on the grinding and control wheel heads and was fitted with a high power (75 kW) grinding wheel drive motor to allow high material removal rates. Grinding force feedback was achieved through differential pressure transducers connected to the hydrostatic bearings. The infeed mechanism was changed from a hydraulic system to a d.c. servomotor. Position and speed control of the infeed axis were achieved by a computer control system based around a BBC micro-computer. The large number of peripherals available for the BBC such as disk drives and I/O cards were intended to provide a basis for a modular control system. However, a decision was taken to develop the intelligent control system on a commercially available C.N.C. which led to the current project.

Production machines have increasingly incorporated a variety of sensors as used in the research techniques described above. TI Coventry produced an angle head grinding machine whose features included automatic wheel balancing, gap elimination and a touch probe for workpiece identification. Wheel speeds in the order of 60 m/s were achieved on machines like the Danex Cylindrical Plunge Grinder through the use of hydrostatic bearings. Rolls-Royce operated an unmanned cell based on a Hauni-Blohm GC1200 creep feed grinding machine that featured continuous wheel dressing, automatic wheel changing and balancing, automatic workpiece loading and unloading together with wheel breakage detection. Adaptive control of grinding wheel speed, work speed and infeed rate was included on the Overbeck 600 Internal Grinding Machine. Granitan filled beds and hydrostatic slideways were used in attempts to improve the damping properties of the machine and reduce vibration. Internal cooling was used to stabilise the operating temperature of the machine.

Although there has been much research in the field of centreless grinding, to date the benefits have only slowly filtered through to industry and are mainly limited to expensive special purpose machines. For the technology to be accepted in industry it is important that new control system strategies are incorporated on commercial equipment with acceptable levels of reliability and support. Manufacturers of new grinding machines use C.N.C. systems from such companies as Fanuc, OSAI-AB and Siemens. Consultation with GKN confirmed that it was unlikely that a manufacturer would risk using a novel control system that was not based on controls from a supplier that could provide suitable support. Increasingly end users specify the manufacturer of the C.N.C. to be fitted to a new machine so as to limit the number of different control systems on-site. This factor further limits the number of controllers which can be employed. The definition of a flexible control system framework that can be incorporated on the hardware of a number of control systems was required.

## 3.2 Process optimisation

The intelligent control system was to comprise a modular framework capable of incorporating a range of process optimisation techniques. In accordance with the project objectives a requirement for particular process models was to be established. The selected process models were to be incorporated within the control system and strategies devised for their operation.

Kinematic modelling was developed by Rowe [33] to account for the geometric effects of the grinding process and provide similar grinding conditions for a workpiece material with a range of workpiece and grinding wheel geometries. Control of grinding conditions was to be through selection of the controlled grinding parameters of infeed rate and work speed. Prior to a grinding cycle, values of kinematic parameters that described the required grinding conditions for the workpiece material were input to the kinematic model resource along with workpiece and grinding wheel geometry. On the basis of the input data the kinematic model suggested values

of the controlled grinding parameters of infeed rate and work speed that would reproduce the specified grinding conditions.

From Rowe, kinematic parameters that described grinding conditions for a material at a particular grinding wheel speed were chip shape ratio '$a_r$' and mean chip volume '$v_m$'. The grinding wheel geometry was defined by its diameter '$d_s$' and grit spacing '$l_d$'. Workpiece geometry data required by the kinematic model was diameter '$d_w$'. The kinematic model was described by the following expressions:

$$v_f = \frac{2\, v_m\, v_s}{\pi\, d_w\, l_d^2} \qquad 1.$$

$$d_e = \frac{d_s\, d_w}{d_s + d_w} \qquad 2.$$

$$v_w = \frac{d_e\, v_s}{2\, l_d\, a_r} \qquad 3.$$

where '$d_e$' is a parameter known as the effective diameter, '$v_f$' is the infeed rate, '$v_w$' is the work speed and '$v_s$' is the grinding wheel speed.

Depending on the requirements of the machine tool user, process optimisation usually involves either increasing production rate or decreasing workpiece cost. Both production rate and minimum cost objectives are affected by the rate of grinding wheel wear, dressing interval and grinding time produced by the current grinding conditions. In the centreless grinding process employed by GKN the rate of grinding wheel wear was small and the dressing intervals large. Therefore, the costs and time involved in replacing and dressing grinding wheels were small in comparison with the costs and time taken by workpiece machining and handling. For the control system that was developed for use by GKN, optimum grinding conditions were chosen as those that maximised removal rate.

The efficiency of the grinding process is defined by specific energy. Specific energy '$e_c$' is the energy required to remove a unit volume of material and is proportional to the grinding force [43]. A high specific energy implies a low material

removal rate and hence low production rate. Power 'P' and removal rate 'Z' define specific energy:

$$e_c = \frac{P}{Z} \qquad 4.$$

Grinding power is related to tangential force '$F_t$' by grinding wheel speed '$v_s$':

$$P = F_t\, v_s \qquad 5.$$

Removal rate is determined by depth of cut 'a', wheel width 'b' and work speed, '$v_w$':

$$Z = a\, b\, v_w \qquad 6.$$

Depth of cut for the centreless grinding process is given by:

$$a = \frac{\pi\, d_w\, v_f}{2\, v_w} \qquad 7.$$

where '$d_w$' is the workpiece diameter and '$v_f$' the infeed rate. It should be noted that the factor of 2 in the diameter is unique to centreless grinding and results from the work holding arrangement. Relating tangential grinding force to specific energy by combining equations 4 to 7 produces:

$$e_c = \frac{2\, F_t\, v_s}{\pi\, d_w\, b\, v_f} \qquad 8.$$

This relationship shows for a given set of grinding conditions that specific energy is proportional to tangential grinding force and remains so for proportional increases in grinding wheel speed, work speed and infeed rate. Increasing grinding wheel speed in isolation reduces the depth of cut required and so reduces grinding forces. However, as mentioned previously process efficiency, as indicated by values of specific energy achieved, may increase or reduce depending on other physical effects.

Experiments performed in the centreless grinding of steel by Rowe [33] indicated that process efficiency is maximised with a grinding wheel speed of 50 m/s independently of changes in other process parameters. Limit charts described the

shape of the boundaries of the centreless grinding process in terms of the controlled grinding parameters of infeed rate and work speed. Figure 5 shows a limit chart for cast iron with the process boundaries of machine power, thermal damage and chatter. A study of the optimisation process showed that for maximum removal rate and avoidance of thermal damage, specific energy was minimised. For the centreless grinding process maximum removal rate occurred at the junction of the machine power and thermal damage boundaries.

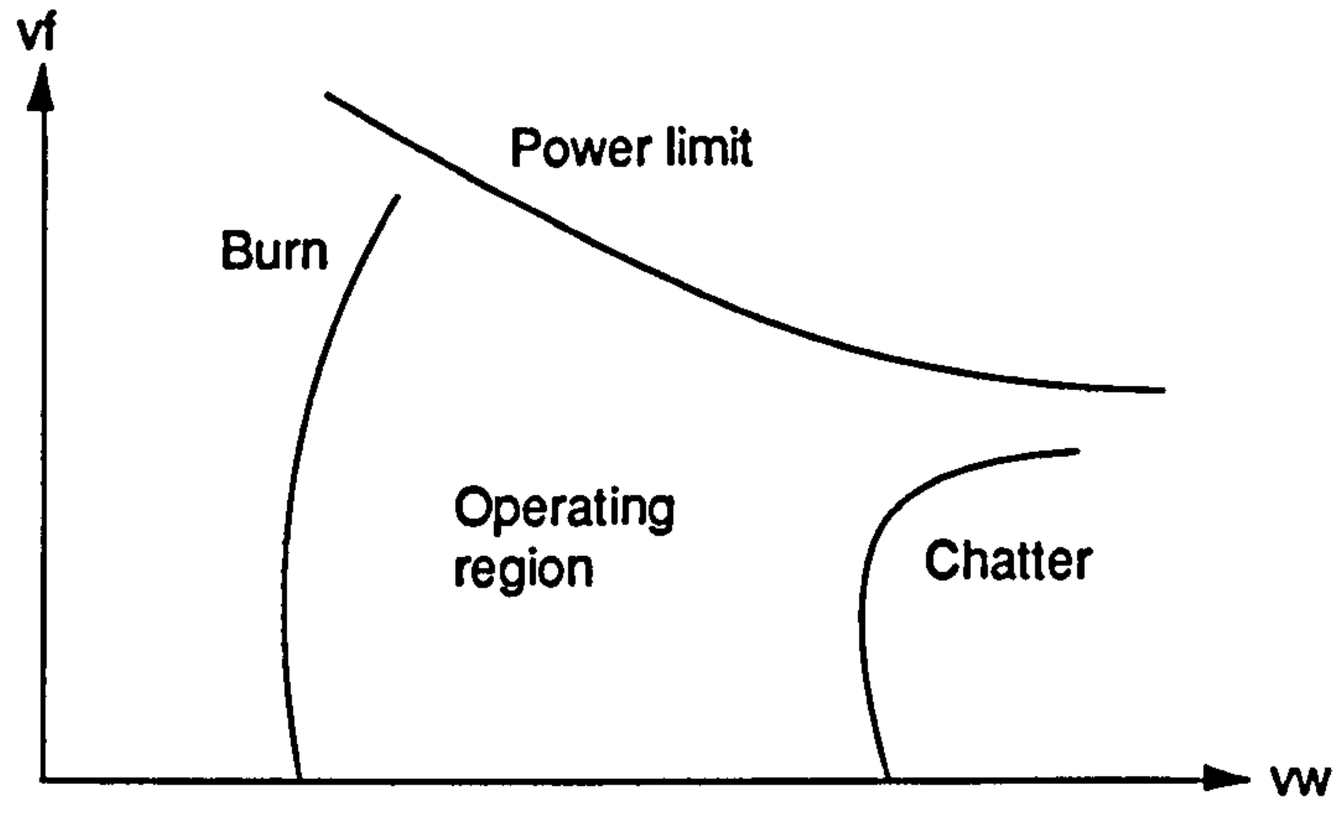



Figure 5. Process limit chart for the centreless grinding of cast iron.

Many papers suggested a relationship between the critical value of grinding zone temperature and the onset of thermal damage [43-45]. Thermal damage, which in some cases is evidenced by visible workpiece burn, is characterised in steel by discoloration due to oxide formation or the transformation of the surface material to austenite. Cracks may also appear on or beneath the workpiece surface. Material damage occurs under the surface and residual stresses resulting from softening or hardening of the workpiece can reduce service life of a machine part. To avoid burn it is generally necessary to increase workpiece speed, reduce infeed rate or reduce specific energy by using a coarser wheel dressing or by changing to a softer grinding wheel. It has also been shown that grinding wheel wear accelerates at the point of burn due to changes in grinding forces and temperature. Hence, given an optimal grinding wheel speed, the material removal rates will be limited directly by the available machine power and indirectly by thermal damage or surface roughness.

For the developed computer control system to avoid thermal damage it was considered necessary to incorporate constraints based on a process model. Use of a model of grinding zone heat dissipation allowed conditions to be selected which prevented workpiece burn when provided with sufficient in-process and material information. Thermal modelling has been the subject of much research to date [43-50] with the result that the following elements are considered to be important in the dissipation of heat from the grinding zone:

1. Workpiece conduction
2. Grinding wheel conduction
3. Heat removed by grinding chips
4. Coolant convection

Heat sinks which are considered to be negligible in comparison to the above are:

5. Kinetic energy of grinding chips
6. Energy required to generate a new surface
7. Residual energy imparted to the new surface

The purpose of a thermal model is to predict the proportion of grinding energy dissipated through these various heat sinks and hence determine the heat flux into the workpiece so as to determine the maximum zone temperature. The grinding energy is imparted by the grinding wheel and so may be determined from the grinding wheel drive power after making allowances for electrical and mechanical losses. It has been argued that it is possible to determine the workpiece surface temperature and hence the energy required to burn the material [45]. Many researchers have different ideas about the importance of the heat sinks listed above. Whilst Outwater and Shaw [49] neglected beneficial effects of convective cooling in the prediction of mean surface temperature for dry grinding, DesRuisseaux [46] showed that such coolant effects can reduce grinding temperatures. Malkin [43] suggested that the specific energy of the grinding process consisted of chip formation, ploughing and sliding elements. Investigations performed by Shafto [50] predicted that in the creep feed grinding

process under 5% of grinding energy dissipated as heat entered the workpiece. Lavine [47] predicted the convective heat transfer at the workpiece surface, the proportion of grinding energy entering the workpiece and the resultant workpiece surface temperature. An experimental method of measuring grinding zone temperature in-process was proposed by Jianshe [48] who used the thermo-couple effects of a graphite impregnated wheel and workpiece to measure temperature.

An improved model developed by Rowe and Pettit [44] was incorporated within the control system in order to define the position of the burn boundary of the limit chart. The new model was an improvement over previous work as it accounted for partitioning of grinding heat flux between the workpiece and grinding wheel. The thermal model employed was described by the following expressions:

$$\frac{1}{R} = \left(\frac{\alpha_w v_s}{\alpha_s v_w}\right)\frac{\lambda_s}{\lambda_w} + 1 \qquad 9.$$

$$l_g = \sqrt{a D_e} \qquad 10.$$

$$l_e = l_g\left[4.95\left(\frac{v_s}{v_w}\right)^{-0.216}\exp\left(-0.0205\left(\frac{v_s}{v_w}\right)^{0.33}\ln(a)\right)\right] \qquad 11.$$

$$e_c^* = \theta_m^* \sqrt{\left(\frac{l_e}{a\, v_w}\right)}\frac{\lambda_w}{0.887\, R\, a} \qquad 12.$$

Critical specific energy is '$e_c^*$' and critical temperature '$\theta_m^*$'. The 'R' factor represents the proportion of the energy partitioned to the workpiece, '$l_g$' is the geometric contact length and '$l_e$' the true contact length. Thermal diffusivity is represented by '$\alpha_s$' and '$\alpha_w$' whilst thermal conductivity is '$\lambda_s$' and '$\lambda_w$' for wheel and workpiece respectively.

# Chapter 4. Specification of process requirements and controller functionality

## 4.1 Process requirements

The intelligent control system structure was to be applied to the grinding of highly compliant cylinder liners and the developed machine tool and control system were to be returned to GKN for shop floor trials. It was important to consider the performance of the manually operated machine tools that were in use at GKN in order to determine the effectiveness required of the control system strategies that were to be employed. The design specification of a plain cylinder liner produced at GKN by a manually operated Cincinnati No. 3 centreless grinding machine is given in Figure 6.

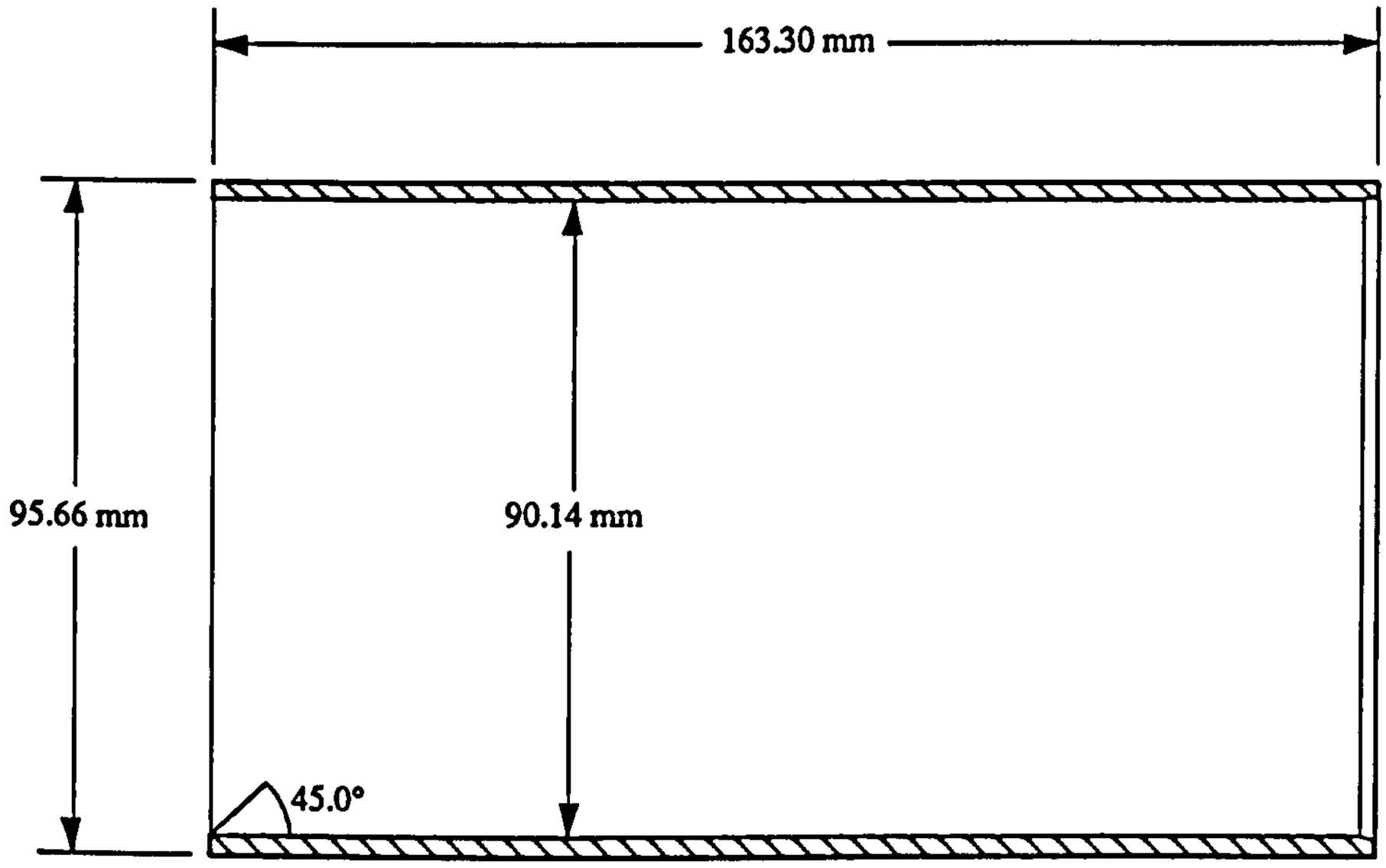



Figure 6. Plain cylinder liner OP6555/A dimensions

Table 1 lists the tolerances that were specified by GKN for production of both plain and flanged cast iron cylinder liners on centreless grinding machines. It was therefore decided that the system must be capable of producing parts within these tolerances but with a higher production rate.

| | |
|---|---|
| Nominal diameter | ± 12.5 μm |
| Roundness | 25 μm |
| Surface texture | 0.5-1.5 μm Ra |

Table 1. Workpiece production tolerances.

A two stage grinding operation was employed in the manual process to achieve the tolerances specified by GKN for a finished workpiece. The grinding wheel was dressed before each batch of 50 rough turned cylinder liners. The settings employed for grinding wheel dressing were chosen to be appropriate for rough grinding of the cylinder liners. After rough grinding the batch of cylinders, the grinding wheel was dressed with settings that provided satisfactory finish grinding conditions. During a finish grinding cycle the operator ensured correct workpiece size by repeatedly removing the liner from the machine for measurement on a set of vee blocks. Approximately 0.5 mm of stock was removed during the rough grinding cycle and 0.1 mm during the finish grinding cycle.

The production rate of a manually operated centreless grinding machine can be used as a basis for comparison with the production rates achieved by the machine with the computer control system developed in this work. The production rate for a manually operated Cincinnati No. 3 centreless grinding machine was based on the time taken to produce a batch of 50 cylinder liners. Table 2 illustrates the calculation of production rates. The total time to produce 50 liners was 6800 s and the component time was thus one fiftieth of this time.

| Operation | Time/component (s) | Batch time (s) |
|---|---|---|
| Rough wheel dress | | 300 |
| Rough floor to floor | 52 | 2600 |
| Actual rough grinding | 17 | 850 |
| Finish wheel dress | | 300 |
| Finish floor to floor | 72 | 3600 |
| Actual finish grinding | 17 | 850 |
| | | |
| Total wheel dress | | 600 |
| Total floor to floor | | 800 |
| Total grinding time | | 1700 |
| Liner floor to floor time | 136 | 6800 |

Table 2. Calculation of floor to floor time.

The total time taken to grind a liner was therefore 136 seconds. However, only 34 seconds (25%) of the production time was spent grinding material whilst the remainder of the time was used for loading, dressing, measurement and machine setting. The study of process conditions revealed that process improvements should aim to increase the efficiency of both productive and non-productive parts of the cycle. It was concluded that automation has the potential to effect such improvements by increasing removal rates and reducing the requirements for loading, unloading, measuring and dressing.

The average material removal rate for a cylinder liner may be determined from the data illustrated in Table 3.

| | |
|---|---|
| Start diameter (mm) | 103.6 |
| Finish diameter (mm) | 103.0 |
| Length (mm) | 220 |
| Average time taken (s) | 34 |

Table 3. Machine cycle data for a typical cylinder liner.

The total volume of material removed '$v_{total}$' is simply:

$$v_{total} = \frac{\pi}{4}(103.6^2 - 103.0^2)\,220 = 21400\ mm^3 \qquad 13.$$

The average material removal rate 'Z' is derived from the volume of material removed and the total grinding time '$t_g$' :

$$Z = \frac{v_{total}}{t_g} = \frac{21400}{34} = 629\ mm^3/s \qquad 14.$$

Using workpiece length 'b' specific removal rate 'Z" may be calculated:

$$Z' = \frac{Z}{b} = \frac{629}{220} = 2.86\ mm^3/mm/s \qquad 15.$$

Limit charts for cast-iron produced by Rowe [33] suggested that specific material removal rates of approximately 37 $mm^3/mm/s$ were possible before thermal damage occurred. From the above table it can be seen that the cylinder liners produced on the manually operated Cincinnati No. 3 at GKN were produced with less than one tenth of the material removal rate possible for the workpiece material. Thus, if it was possible to increase material removal rate on the manually operated Cincinnati No. 3 to 37 $mm^3/mm/s$, total grinding time for a liner would reduce from 34 seconds to 2.6 seconds. The total time taken to produce a liner would then reduce by 31.4 seconds (23%) to 104.6 seconds. Dressing, loading and handling times would account for the remaining 102 seconds (97.5%) of the total time taken to produce a liner. Therefore, in order to improve the production rate of a manually operated Cincinnati No. 3 grinding machine it was not only necessary to improve material removal rate but, more importantly, to reduce the dressing, loading, measurement and handling times.

## 4.2 Controller functionality

A prime consideration was that the control system and machine tool were to be used in production by GKN (Sheepbridge) after the end of the project. Therefore, it was important that the computer control system hardware was supported by a controller manufacturer. Leading manufacturers which include Fanuc (Japan), Siemens (Germany), OSAI A-B (Italy), Bosch (Germany), Philips (Holland), Heidenhain (Germany) and Fagor (Spain) supply equipment to suit a variety of machine tools including lathes, milling machines, boring machines and in some cases grinding machines. The manufacturer of the control system used in the project, OSAI A-B, had a proven industrial record and provided technical support from a team of application engineers.

Two alternative approaches for realisation of the control system design were considered. Both methods relied on a commercial C.N.C. system to provide elementary machine control functions such as axis control. The first approach was to integrate the control system design within a host computer system such as an IBM PC. The design of a communications interface between the PC and the C.N.C. together with data acquisition equipment would provide the host system with control over the process. An advantage of this method was that integration of the control system with C.N.C. systems from other manufacturers would be simplified. A schematic diagram of such a system is given in Figure 7. The second approach considered was to integrate the conceptual framework of the intelligent control system within the C.N.C. system itself. Advantages of integration included a reduction in the cost and the complexity of system hardware. It was therefore decided to integrate the intelligent control system within the C.N.C.

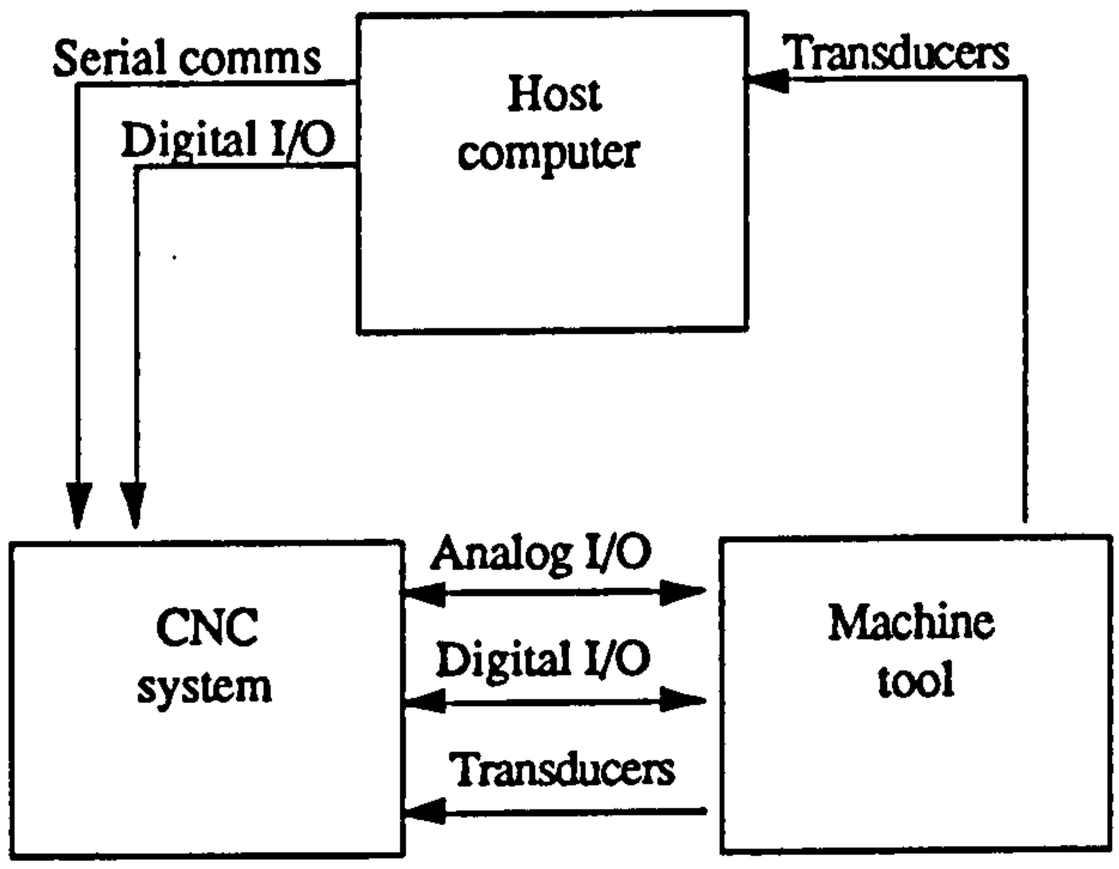



Figure 7. Schematic diagram of PC based control system.

Integration of the control system framework within the C.N.C. system placed certain requirements on the computer control system hardware and software functionality. Principally, the ability to generate custom software for real-time monitoring, calculation and analysis of the values of machine tool parameters was necessary. A multi-tasking operating system would allow execution of normal control system functions in parallel with such software. It was necessary to study the features of such modern computer control systems in order to evaluate the suitability of such systems for integration with the proposed control system. The capabilities and performance of the OSAI A-B 8600 C.N.C. used in the project were of particular interest [51-56].

A modern C.N.C. typically interfaces with a machine tool through a wide range of input and output (I/O) devices. Analog outputs provide speed reference voltages to the machine tool axis drives. Transducers provide axis position information from a range of devices that include resolvers, incremental encoders, linear scales and absolute encoders. Analogue inputs allow monitoring of such items as spindle power consumption and speed. Digital inputs indicate the condition of such items as selector switches, push buttons, limit switches, pressure switches and thermal over-loads. Machine functions controlled through devices including relays and solenoids are driven by digital outputs. Most C.N.C. systems provide communications with a variety of

protocols such as RS232 and RS485 in current use. Communications allow program and data transfer to and from external intelligent systems. In some cases it is possible to control the C.N.C. using a host computer with a suitable serial communications link. In order to overcome the noise and drift problems associated with analog electronics, the development of digitally controlled axis drives has allowed some manufacturers to replace analog outputs with high speed communications links. Fibre optics are also becoming common with uses including distributed I/O and axis control.

The 8600 C.N.C. system comprised an operator panel and controller chassis. The operator panel provided the operator interface to the C.N.C. through a 12" monochrome CRT display, console switches, QWERTY keyboard and function keyboard. Figure 8 illustrates the design of the 8600 console.

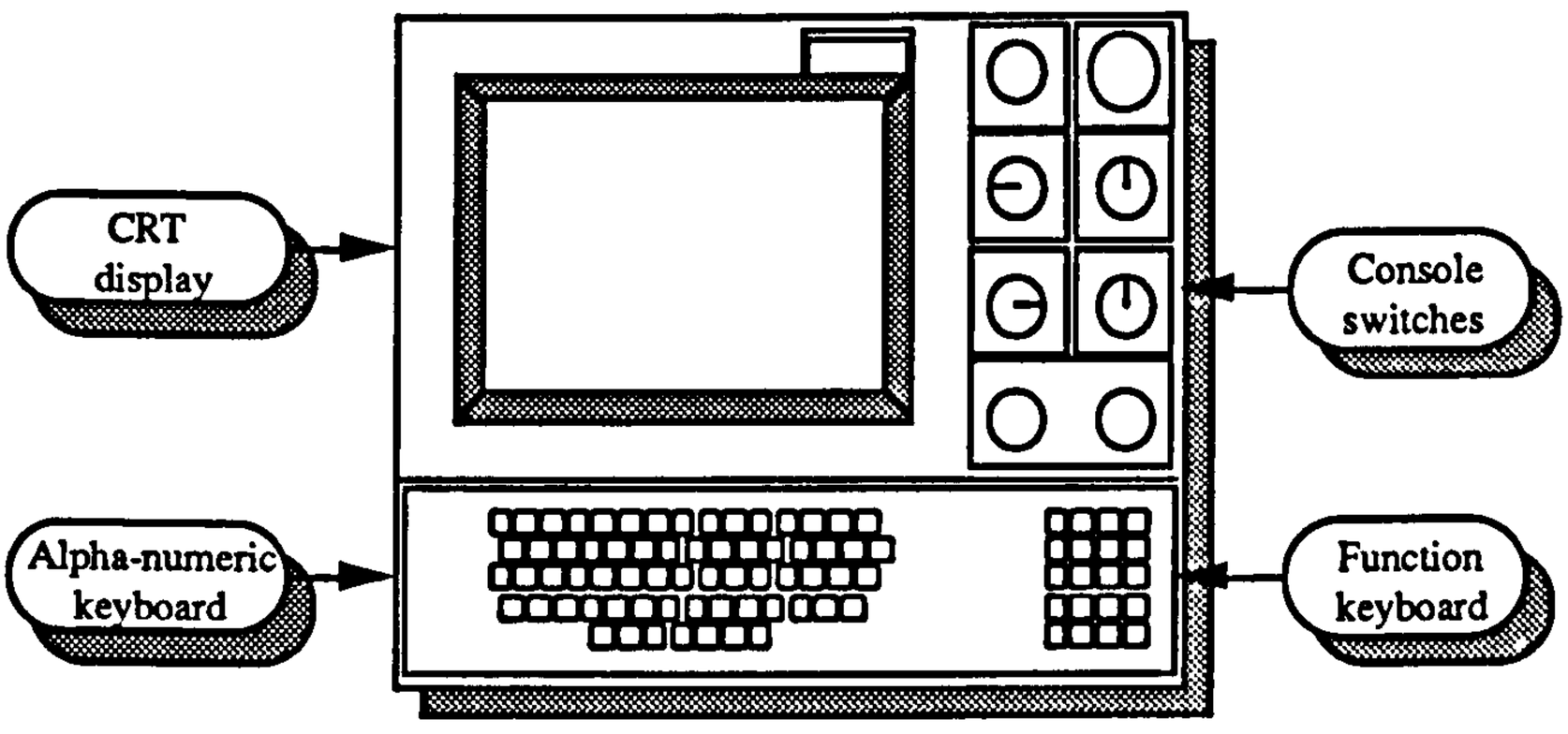



Figure 8. Control system console.

The controller chassis was a rack system with a 'multi-bus' based backplane into which a variety of cards were slotted. The cards that defined the system hardware comprised the two logical blocks of mini-computer and process oriented hardware. A schematic diagram of this arrangement is given in Figure 9.

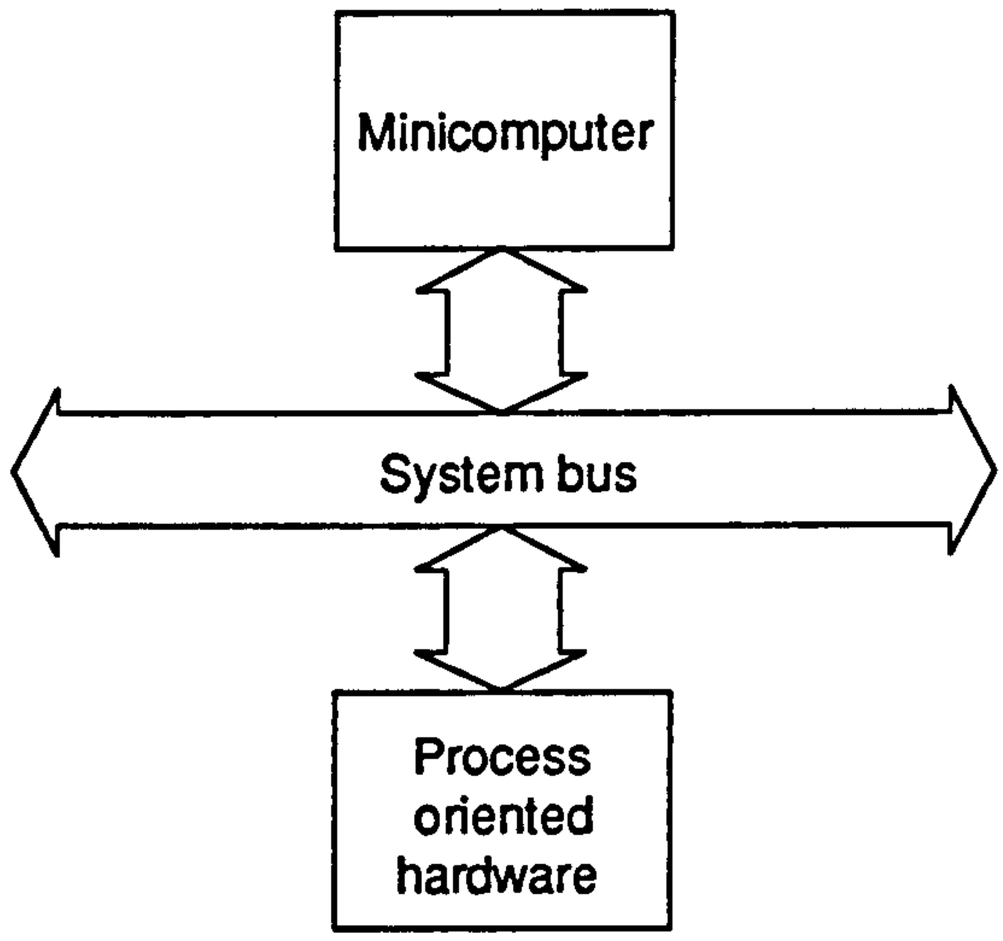



Figure 9. Hardware system configuration.

The 8600 series is a multi-processor system and the number of CPU cards that defined the mini-computer hardware depended on the performance required of a particular system. Each CPU card was based on the Intel 80286 processor working in tandem with an 80287 maths co-processor and included an amount of both RAM and EPROM memory storage. Specific tasks such as axis handling, part-program management, video display and file handling were allocated to the CPU cards in the system. Each CPU could autonomously access system resources when required. After discussions with OSAI A-B it was determined that the standard system specification of two CPU cards would provide adequate performance for the application. The first CPU was responsible for handling system housekeeping functions whilst the second CPU managed the machine tool axes servo-control.

Process oriented hardware handled the connection between the mini-computer and the machine tool. A number of cards catered for machine axis control, digital I/O management and communications. Two incremental encoder interface cards were specified to provide transducer inputs for the monitoring of infeed axis position, grinding wheel speed, control wheel speed and the dual sizing devices. Speed reference voltages to the infeed axis servo-amplifier, the grinding wheel thyristor drive and the control wheel invertor were to be provided with a D/A - A/D convertor card.

Also, an A/D input was to be used to measure grinding wheel power. Machine functions were catered for by a single digital input/output card that provided 32 input and 32 output points. Also, two serial ports were available for connection to a host computer. A 20 slot rack was used to house the cards and allow for future system expansion.

A wide range of computer control systems are applied to machine tool applications. Generally in such control systems a workpiece is produced by performing a series of moves and operations that are encoded in a 'part-program'. The codes and format of part-program instructions from different control system manufacturers typically adhere loosely to the ISO 'G code' standard. Although part-programs can often be created on-line with the control system, 'G code' programming is ideally suited for applications where a complex program is generated off-line with a computer using data generated by a CAD package. However, more user friendly methods of on-line program creation that employ graphics and menu-driven front end software are becoming popular. Early numerical control systems sequentially processed part-program commands encoded in punched tape. Although primitive and inflexible when compared with modern systems, such numerical control (N.C.) systems are still used in industry. As computer processing power increased, computer numerical control (C.N.C.) systems were employed. C.N.C. systems stored part-programs in memory with punched tape or other means used for off-line storage. Increasingly the C.N.C. market demands personal computer type features such as full colour graphics, simulation and dialogue programming environments. As it is common for one particular model of C.N.C. system to be applied to a range of machine tools with varying I/O requirements it is important that there is a facility for the machine tool builder to customise C.N.C. operation. Generally, a C.N.C. executes a machine logic or interface program in parallel with the part-program. The purpose of the machine logic program is to provide a flexible interface between machine tool I/O and the C.N.C. The machine logic program is usually written by a machine tool builder in a simple ladder or statement language.

The 8600 series software was characterised by a modular software organisation based on the ERTS-86 multi-tasking operating system. The system modules consisted of the operating system, the service software, the process software and the diagnostic software. Figure 10 illustrates the organisation of the system software modules. The operating system provided the interface between process software and machine hardware by scheduling hardware resources to different software tasks. The environment was a multi-tasking, real-time system where individual software tasks competed asynchronously for hardware resources. The operating system managed the competition and controlled the exchange of information between the tasks. The system provided two partitions that allowed process and service software to run simultaneously. Service software comprised three parts running in the background partition of the operating system. Service utilities allowed file handling and program editing operations and it was possible to execute service software whilst the machine tool was operating.

The SIPROM Boolean programming language and environment allowed development of the machine logic interface for the machine tool. The SIPROM program could be created either on-line on the C.N.C. or off line on a PC. Diagnostic facilities enabled debugging of the developed machine logic. The DEBUG utility allowed system hardware resources such as I/O and memory to be debugged. Process software performed functions required for numerical control of a machine tool whilst running from the operating system foreground partition. Application software and machine logic execution were encompassed by process software. The application software provided the normal working process of the machine tool such as axis control and part program execution. Input from an operator or peripheral was also handled by application software. The operation of the process software was determined by a number of characterisation files that allowed the machine tool builder to customise the C.N.C. The file named FCRSYS was used to define system characterisation parameters such as memory allocation and device driver installation. The axis characterisation file AXCFIL defined the C.N.C. in terms of the number and types of

controlled axes and provided parameters used for control of axes. The process configuration file PGCFIL calibrated process data areas that encompassed such elements as part program variables and graphic screen memory. Machine logic operation was configured by file IOCFIL. Further information is available from the OSAI A-B 8600 MC-TC multi-process software characterisation and interface manual.

Implementation of the conceptual framework of the intelligent control system structure required a detailed understanding of the 8600 software architecture. The control system framework was constructed principally through part-program, machine logic and CSI software elements interacting with each other and the C.N.C. hardware resources. The operating system managed communications between each of these elements. Sophisticated part-programming techniques were available through the use of the ASSET extension to the ISO standard 'G code' language. Amongst other functions, ASSET instructions coded into a part-program permitted a programmer to perform calculations, prompt the operator for keyboard input and display custom screens. Perhaps most importantly, the open architecture, multi-tasking operating system allowed integration of high-level custom software through the Custom Software Interface (CSI) environment. A feature of key importance for the implementation of the intelligent control system framework was the ability of a system developer to integrate CSI modules. The CSI modules allowed interaction with the C.N.C. operating system to an extent not possible with conventional part-programming languages or machine logic. This capability allowed control system resources to access data base structures through file handling, to provide custom screen displays and to perform complex mathematical operations whilst communicating with software at other levels in the system. CSI modules were created off-line on an IBM PC compatible computer using the Intel high level language PL/M-86 and assembler ASM-86. Once installed, a CSI module was allocated a priority and its execution scheduled by the operating system.

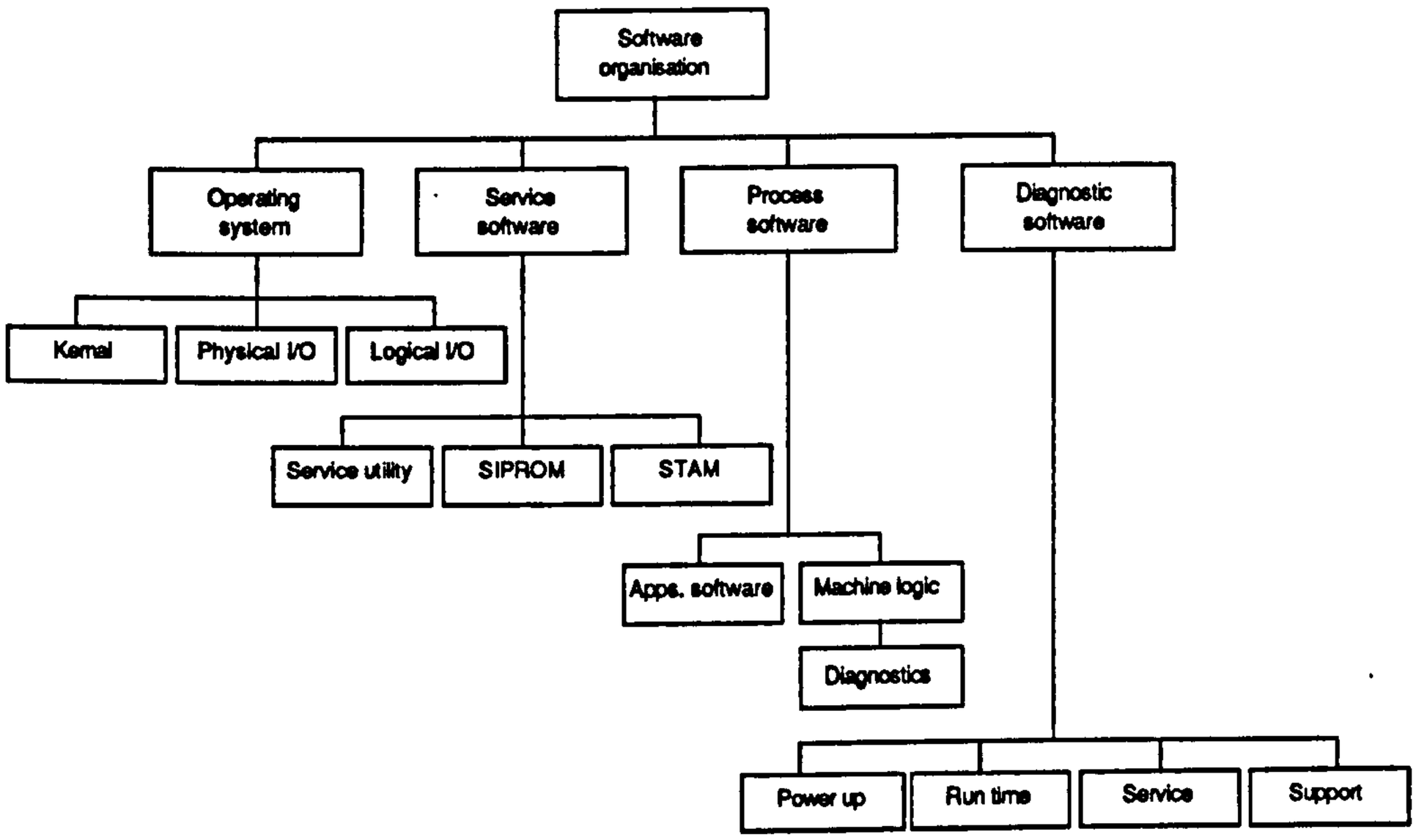



Figure 10. General organisation of software system.

## 4.3 Conclusions

Improvements in productivity can best be achieved with a combination of higher material removal rates and reduced workpiece handling times. In order to achieve the accuracies required of the process these objectives should be sought by improvements in the process and by incorporation of additional intelligence into the control system. Specifically, it is necessary to investigate techniques for improving the process so that separate rough and finish grinding operations are not required. Furthermore, it is important to develop suitable grinding cycles that, in conjunction with in-process measurement of workpiece size, are capable of accounting for variability in the compliance of workpieces. It was decided to aim for a 50% reduction in the time required for the grinding operations.

The decision to integrate the intelligent control system within the C.N.C. system rather than develop a host control system based on a PC was particularly important. The modularity of the 8600 series C.N.C. hardware was well suited to integration of the control system framework. Important features were the multi-tasking capabilities of the operating system, the available processing power and the range of

input/output options available. For the purposes of this project the advantages of integration over a host system were lower hardware cost and reduced system complexity for the end user. This approach also required that a detailed and in depth knowledge of the C.N.C. system software functionality was developed. However, it was clear that the ability of a host PC based system to be applied to a range of control systems from different manufacturers would be advantageous given different project objectives.

# Chapter 5. Development of system concepts

## 5.1 Control system structure

A generic control system structure for intelligent control of plunge grinding processes was to be devised and the developed structure applied to the centreless grinding process. The basic concept for such a control system was outlined at the outset of the project by Rowe [2]. The control system approach was described as a conceptual framework for intelligent control systems. A diagram of the initial conceptual framework is given in Figure 1.

Rowe proposed that design and creation of a general purpose executor routine allowed intelligent control systems for a range of grinding processes to be developed quickly and in an organised manner. The executor was to provide the interface between a machine tool and the control system modules required for a specific process. It was suggested that interaction between an operator and the control system would be through custom designed input and output modules. Also, system intelligence was to be achieved through incorporation of process models operating on data stored in a suitably constructed data base.

It was convenient to discriminate between the types of resource available from a particular control system and machine tool. Physical resources interface with the 'real world' and include operator input/output functions, in-process measurement features and machine tool control. However it was envisaged that physical resources are generally fixed for a particular machine tool. Logical resources perform operations on data and include elements such as data bases, file management and process modelling. The executor routine was intended to perform the following steps:

1. Request operator input
2. Search data base for relevant process constants
3. Reference process models for calculation of starting parameters from process constants

4. Wait for cycle start
5. Perform cycle whilst controlling process parameters dynamically
6. Check process conditions achieved during cycle -power, specific energy, etc.
7. Reference data base for calculation of improved process constants from process conditions and parameters
8. Display operator output

It was decided to refine the conceptual framework design and develop a definition for a structure compatible with the operation of modern micro-processor systems and high-level programming languages.

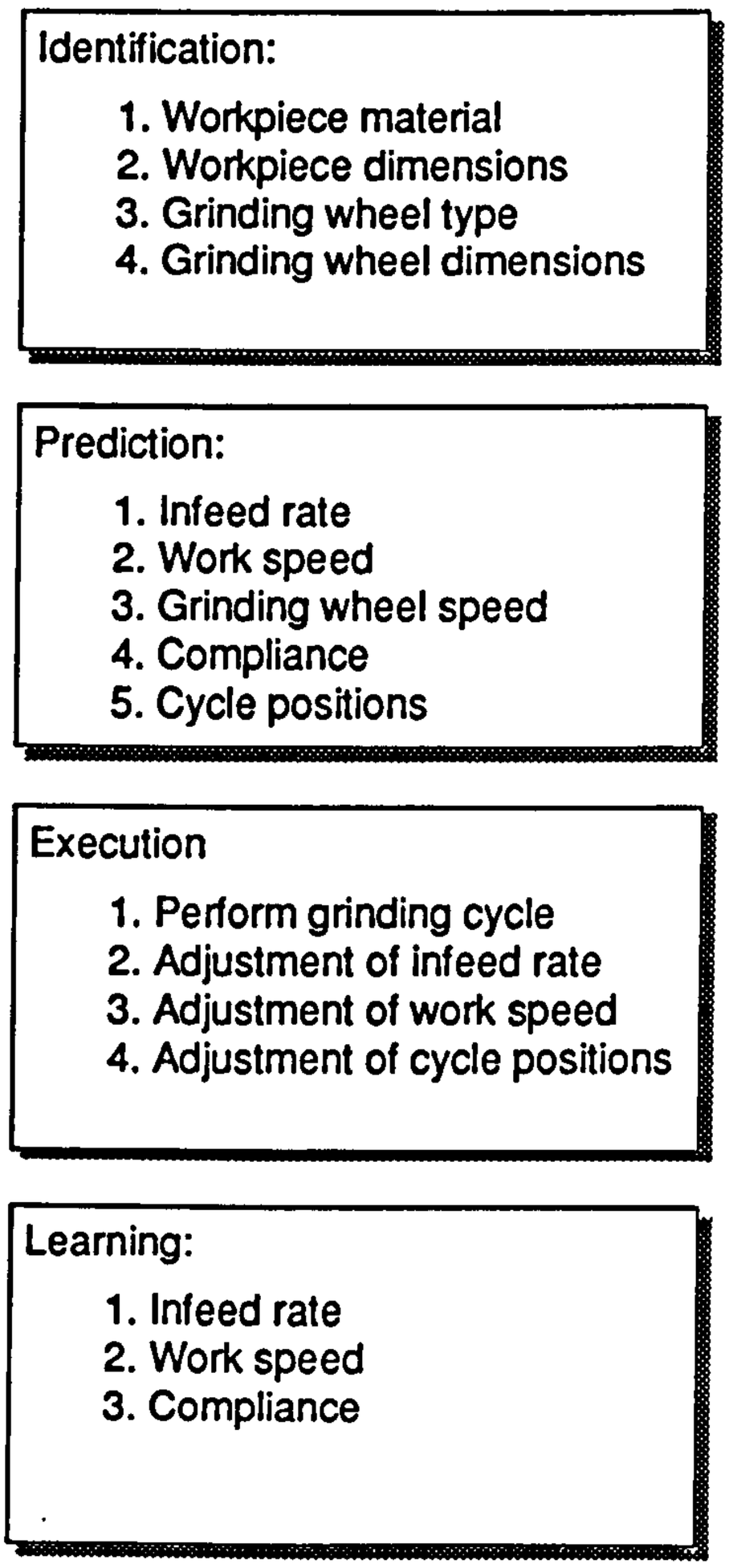



Figure 11. Example of cycle phases for centreless grinding

Consideration was given to the manner in which the requirements of an intelligent control system for the grinding process related to the conceptual framework described above. A conclusion reached was that a cycle performed by the control system comprised four basic phases of operation: identification, prediction, execution and learning. Although it can be argued that operation of cycle phases would be similar for all grinding processes it was apparent that the phases of such a cycle depended on the particular process in question. Figure 11 illustrates the cycle phases that might be considered appropriate for a centreless grinding process.

Defining control system operation through construction of appropriate cycle phases for the process required that the executor was no longer the main sequencing element of the control system. It became necessary to perceive the executor as a software engine managing data transfer between control system resources and providing machine control functions. The implication was that operation of a control system could be defined by simple allocation/mapping of system resources to each phase of the cycle. Furthermore, it was useful to define machine control cycles as system resources independent of both the executor and the execution phase of the cycle module. By creating a library of such machine control cycles it was possible to map the execution phase of the cycle module to an appropriate machining cycle.

In principle it was possible to add system resources to a library of control system modules and customise control system operation through development of suitable cycle phase maps and machine control cycles. This concept achieved the prime requirement of generality and flexibility for application to new processes. Therefore both cycle phase and machine control cycle module libraries were added to the modules previously defined in the control system framework. Figure 12 illustrates the revised conceptual framework.

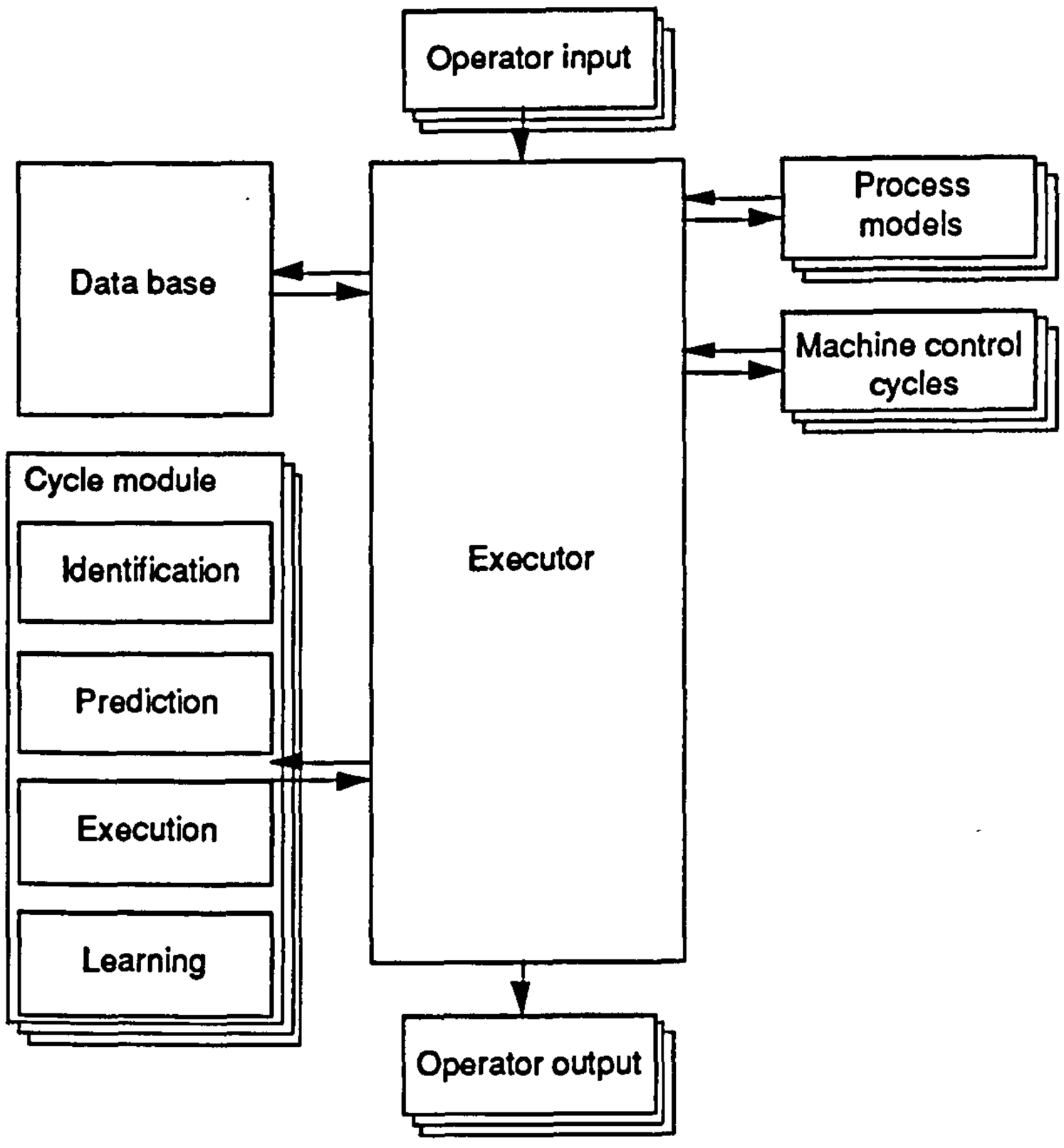



Figure 12. Revised outline of the conceptual framework

As the control system resources performed operations on data passed by the executor, data management was the key function of the executor. Each data item either stored in the data base or produced by a logical resource was allocated a unique identifier. Prior to performing a machining cycle the executor determined from each of the resources allocated in the cycle phase map the input data required and the output data generated by that resource. Input data to a resource was derived either from the data base, operator input or the output from a previous resource in the sequence. Similarly output data from a resource was directed to either the data base, operator output or as input data to another resource as required. Adoption of this approach ensured that modularity was maintained and development of new system resources was simplified.

The original conceptual framework definition provided learning capabilities as part of the executor routine operation. The principle was that process models acting on data base information predicted operating parameters that would produce the required operating conditions for the process. Subsequently the executor routine optimised the data base information using the information gathered through in-process measurement of operating conditions during a machining cycle. Central to the learning capability of the suggested control system structure was a requirement for in-process measurement and application of learning strategies.

The consideration given to related processes in earlier chapters led to the conclusion that available in-process measurement systems varied according to the process and that the objectives of learning strategies for such systems were not defined. It was therefore decided to remove the in-process measurement and learning elements from the executor routine and to include them as additional system modules. This approach simplified both development of new learning strategies and incorporation of varied in-process measurement devices. Also, the in-process measurement resources were then available for use by other system modules. The adoption of this strategy was therefore consistent with the developing conceptual framework.

The approach was extended to include the real-time control or adaptive elements of the system. Real-time control of operating parameters required the combination of an adaptive strategy employing in-process measurement of operating conditions with the ability to decide how to adjust the operating parameters in order to maintain the required operating conditions. It was concluded that adaptive strategies varied according to the specific process in the same manner as the learning strategies described earlier. To maintain flexibility it was therefore essential that the adaptive control elements of the system were constructed as logical system resources and included in an individual system module.

In any manufacturing system the safety element must be considered. In line with the development of the conceptual framework and to provide flexibility it was decided to incorporate system safety strategies within a further system module. Two types of safety strategy were identified. In the first case the executor passed values of parameters produced by other system modules, including operator input, to the safety module to ensure that the values were within an acceptable range. This approach ensured that calculated operating parameters such as infeed rate were limited to safe values for a particular process. The operation of the second type of safety strategy was defined as a result of a requirement for the system to modify operating parameters in real-time under conditions detected by in-process measurement. For example using this type of strategy it was possible to reduce infeed rate in response to overload of the grinding wheel drive. The system safety module was therefore included in the new conceptual framework illustrated in Figure 13.

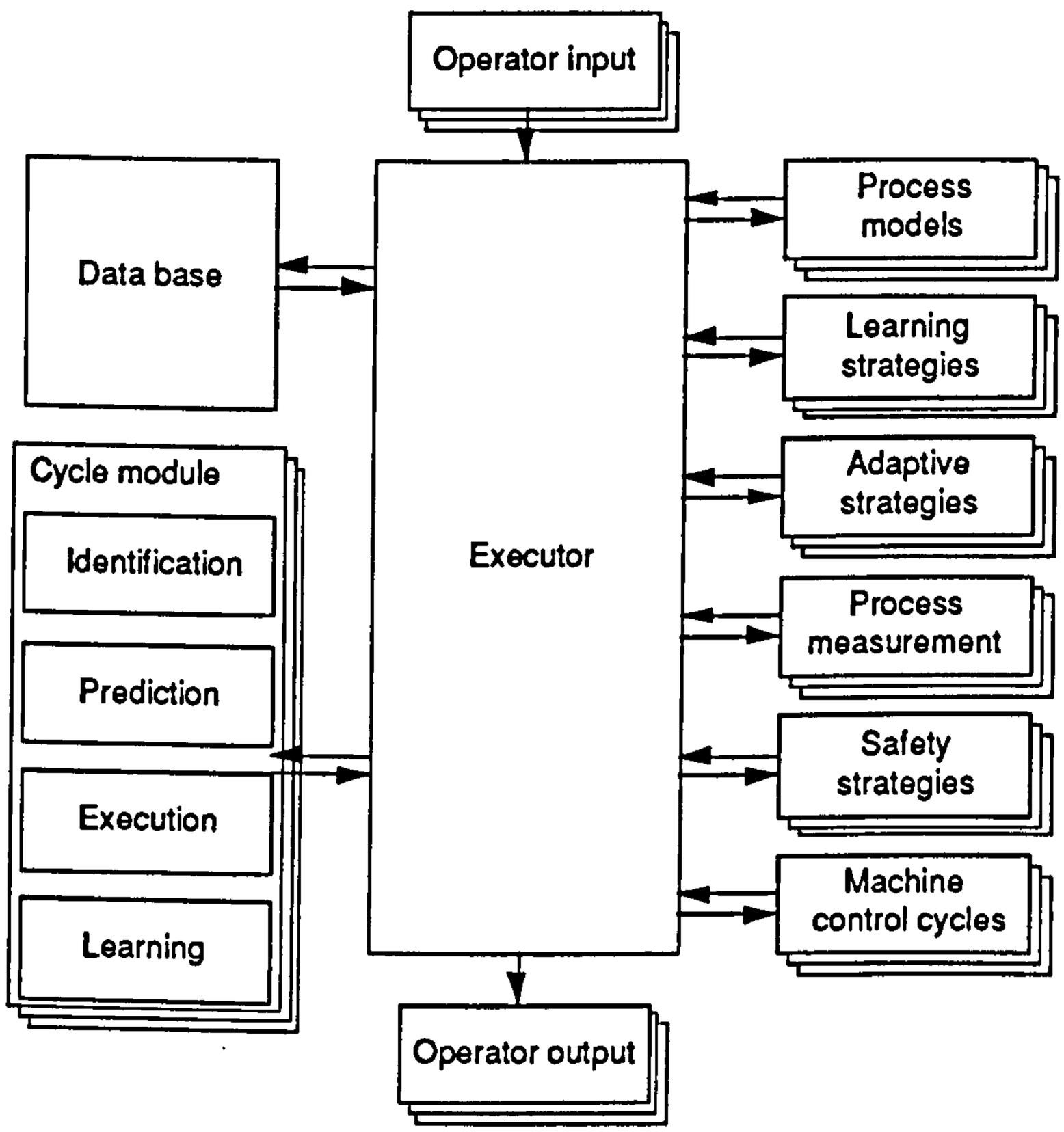



Figure 13. Developed outline of the conceptual framework

In order to preserve system modularity, each of the control system framework modules existed conceptually as an autonomous function. However, it was realised that in practice a module could comprise software operating at and communicating between a variety of levels within a system. Indeed the multi-layer, multi-tasking operating system employed on the OSAI A-B 8600 C.N.C. required that the framework modules were distributed between the part-program, SIPROM and CSI environments with the system providing communication between the environments.

Implementation of the structure described above on the 8600 C.N.C. required incorporation of the new conceptual framework design within appropriate levels of the 8600 system software. Figure 14 illustrates the interaction between the various system software levels.

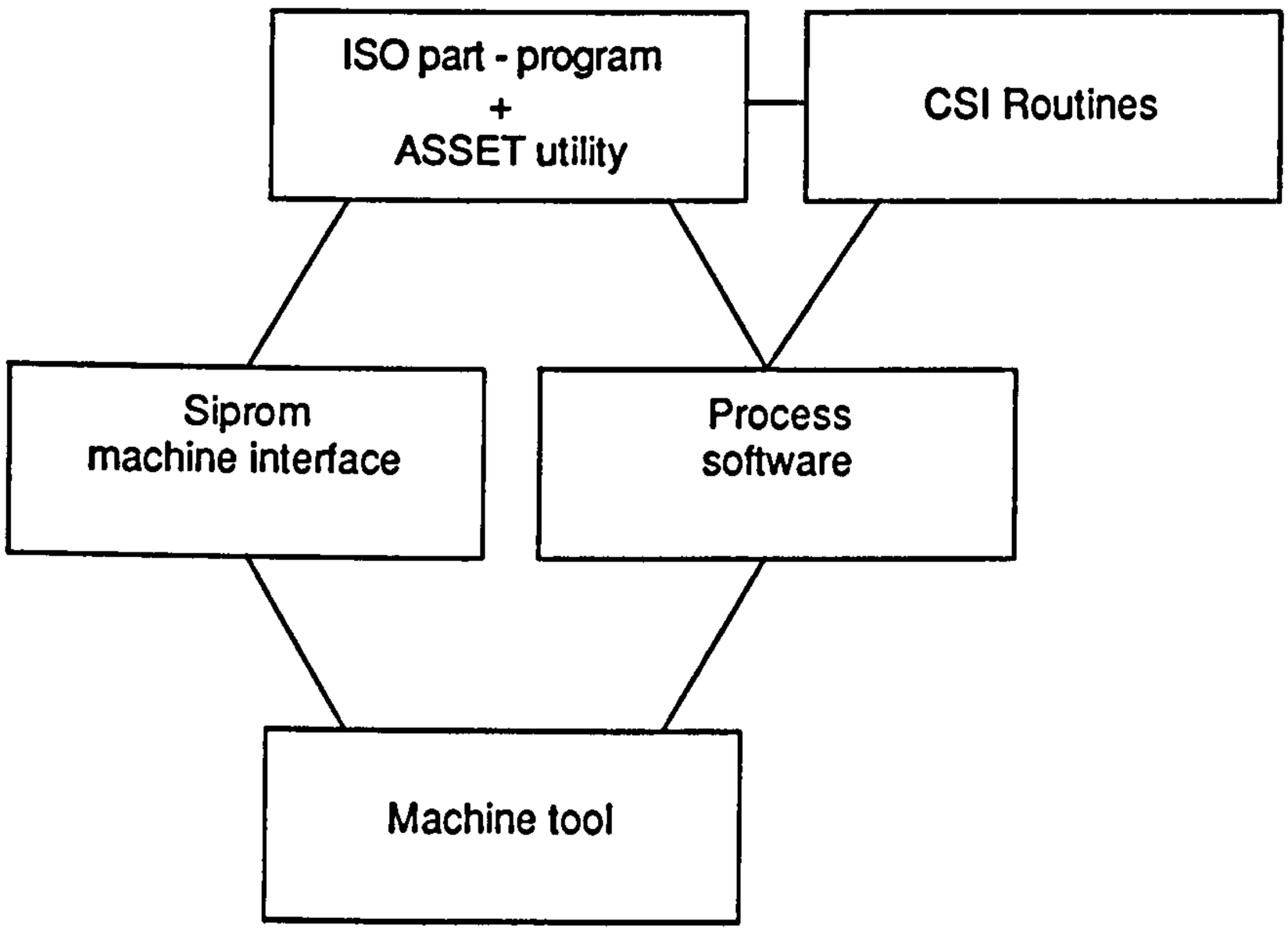



Figure 14. Interaction between the 8600 system software

The executor was written as a complex part-program that interacted with the other levels of the system software. For example, machine cycle control functions were designed as parameterised part-program sub-routines. Parameters defining axis positions, feed rates, wheel speeds, etc. were assigned values by the executor function both prior to and during execution as required. Additions to the ISO standard part-

program language instructions -provided by the ASSET utility, allowed incorporation within part-programs of operator input through the keyboard and output through the screen. Computationally intensive logical resources including real-time screen displays, process models, learning strategies and adaptive strategies were implemented in the CSI environment using modules written in PLM/86 and ASM-86 [57,58]. A data base was constructed for storage of system data using a number of cross-referenced formatted files stored in part-program memory. Physical resources requiring machine operations were accessed from part-program instructions via the process software for axis control and through SIPROM interface software for management of digital inputs and outputs.

Application of the control system structure to intelligent control of the centreless grinding process required the creation of particular system resources that were accessed by the identification, prediction, execution and learning phases of the cycle. The executor processed a typical cycle as follows. An operator selected the current workpiece description, material, coolant, grinding wheel and cycle type from the library of those available. The identification phase of the cycle required in-process measurement, machine tool control and operator input resources to identify the workpiece to be machined. The prediction phase utilised process model and safety resources to generate controlled grinding parameters that produced safe grinding conditions. The execution phase performed a grinding cycle whilst operating on adaptive, process model, process measurement and machine tool control resources. The learning phase required process model and process measurement resources in order to evaluate the previous grinding cycle. All resources operated on data passed by the executor and the outputs from each resource were verified by the safety module. Figure 15 illustrates resources accessed by the phases of a typical cycle module.

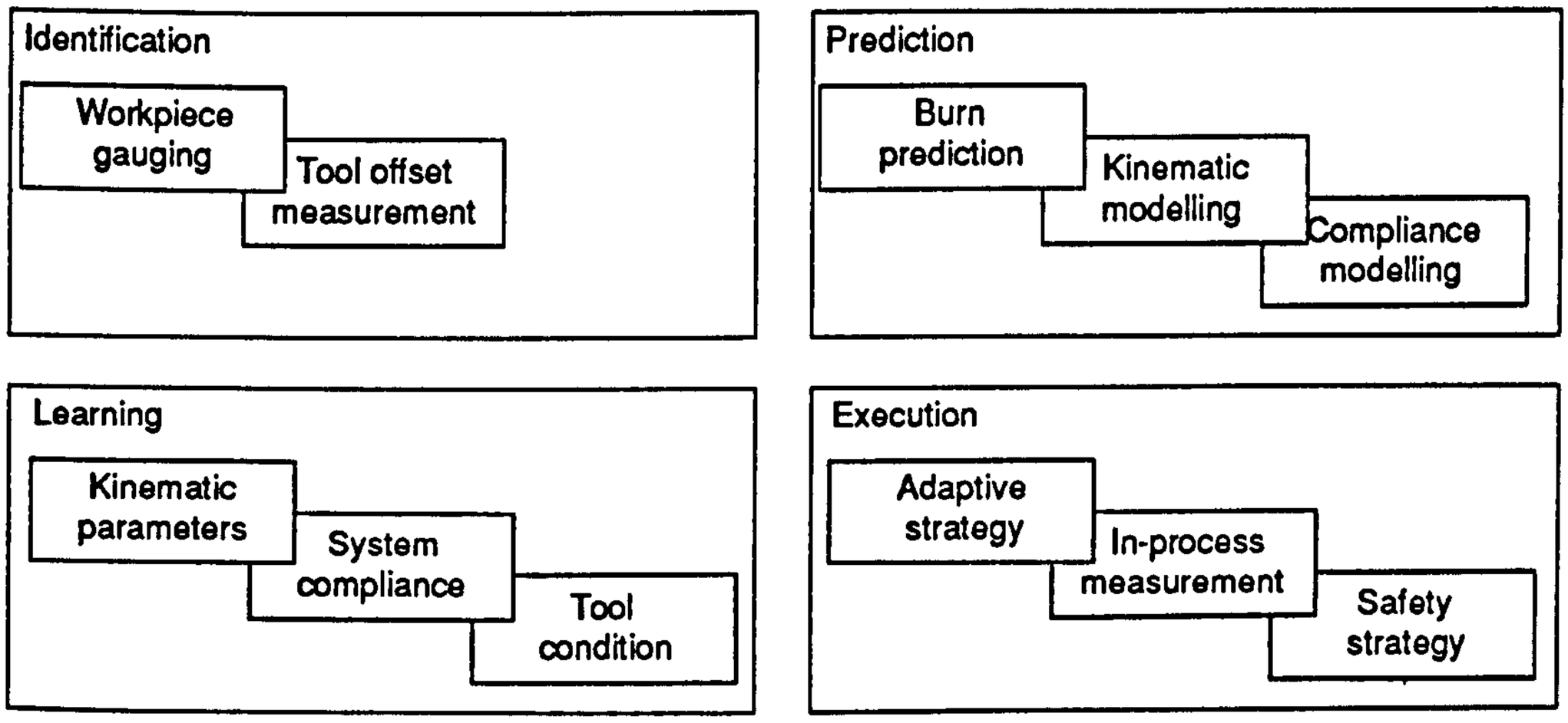



Figure 15. Phase resources for the centreless grinding process

## 5.2 Integration of process models

The system was designed to provide a generic approach to optimisation of the centreless plunge grinding process. Process optimisation was achieved through a combination of process modelling and optimisation strategies. The control system was required to optimise grinding conditions through selection of suitable values of the controlled grinding parameters of infeed rate, work speed and grinding wheel speed. Although most researchers considered maximum removal rate to represent grinding process optimisation it was also important to consider production rate and cost per workpiece.

In the particular case of the highly compliant cylinder liners produced by GKN, production rates were limited by system deflection and correspondingly low removal rates. Optimisation of grinding conditions produced only small improvements in production rates and consequently process optimisation relied mainly on incorporation of process models and strategies that allowed the system to compensate for system compliance and thus reduce the number of grinding operations.

| Parameter | Identifier |
|---|---|
| Workpiece | $w_i$ |
| Description | Text string |
| Internal diameter | $d_{internal}$ |
| Initial diameter | $d_{initial}$ |
| Target diameter | $d_{target}$ |
| Length | b |
| Index to material | $m_j$ |

Table 4. Workpiece data.

| Parameter | Identifier |
|---|---|
| Material | $m_j$ |
| Description | Text string |
| Chip shape ratio | $a_r$ |
| Mean chip volume | $v_m$ |
| Optimum grinding speed | $v_{s\ opt}$ |

Table 5. Material data.

| Parameter | Identifier |
|---|---|
| Grinding wheel | $G_k$ |
| Description | Text string |
| Wheel diameter | $d_s$ |
| Grit size ≈ grit spacing | $l_d$ |
| Maximum rated speed | $v_{s\ max}$ |

Table 6. Grinding wheel data.

A kinematic model [33] was incorporated within the intelligent control system structure. The purpose of this model was to predict values of infeed rate and work speed that reproduced known grinding conditions for varying workpiece and grinding wheel geometries. The expressions that defined the kinematic model are listed in Chapter 2. The model required empirically derived kinematic parameters '$a_r$' and '$v_m$'

for the specified material and information concerning the grinding wheel and workpiece geometry. Consequently, the information shown in Tables 4 to 6 was stored in the control system data base.

The relatively small amount of data required by kinematic modelling ensured an efficient, economic data base for the control system. The kinematic model resource comprised a CSI module written in a combination of PLM-86 and ASM-86 sub-routines. Values of infeed rate and work speed were determined according to the kinematic expressions 1 to 3 from Chapter 3.

A model of grinding zone heat dissipation was integrated within the control system structure. The purpose of this thermal model was to ensure that material removal rates produced by the optimisation strategies were not sufficient to cause thermal damage to the workpiece. The thermal model employed [44,45] is described by expressions 9 to 12 and required the information described in Tables 7 and 8 for each of the materials, grinding wheels and workpieces in the data base.

| Parameter | Identifier |
|---|---|
| Material | $m_j$ |
| Thermal conductivity | $\lambda_w$ |
| Thermal diffusivity | $\alpha_w$ |
| Critical temperature | $\theta_m^*$ |

Table 7. Thermal data for workpiece material.

| Parameter | Identifier |
|---|---|
| Grinding wheel | $G_k$ |
| Thermal conductivity | $\lambda_s$ |
| Thermal diffusivity | $\alpha_s$ |
| Wheel diameter | $d_s$ |

Table 8. Thermal data for grinding wheel.

A CSI module was created using a combination of PLM-86 and ASM-86 sub-routines that calculated the values of critical specific energy for a particular combination of workpiece, material and grinding wheel. The CSI module was implemented as a process model resource within the control system framework.

Figure 16 illustrates a conventional plunge grinding cycle and indicates the manner in which the workpiece diameter reduction lags behind the true infeed position due to the system deflections caused by grinding forces [9,16]. The workpiece diameter reduces towards the required target diameter '$X_T$' during the dwell period '$t_d$'.

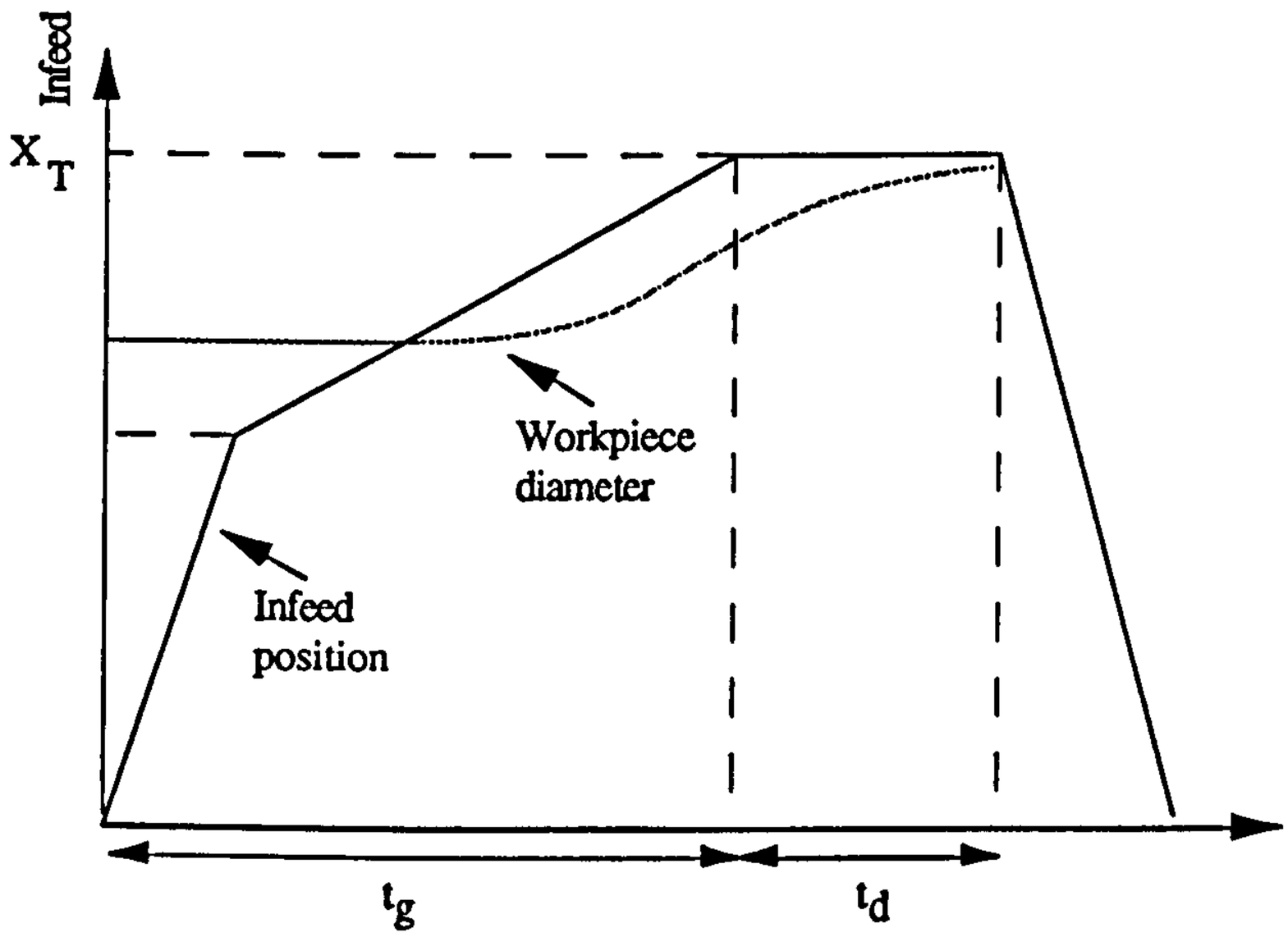



Figure 16. Illustration of conventional plunge grinding cycle with dwell.

In order to reduce the lengthy dwell times required for highly compliant workpieces to reach target size it was decided to replace the target position employed in the conventional grinding cycle with an overshoot position. Use of an infeed overshoot position '$X_{OS}$' leads to a build up of strains within the distorted workpiece allowing larger grinding forces and increased material removal rates during the dwell period [16,59]. Figure 17 illustrates a fixed overshoot grinding cycle.

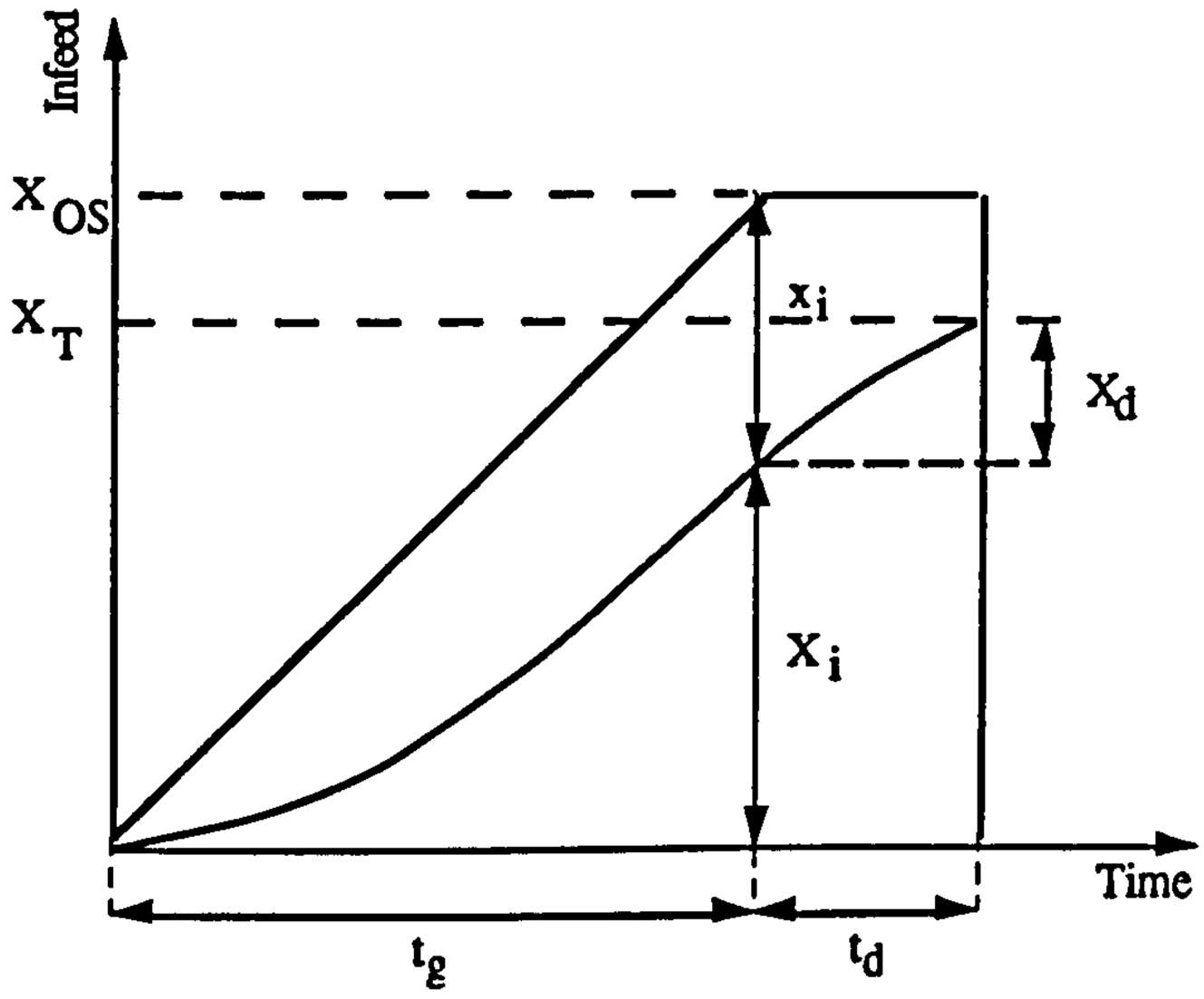



Figure 17. Relationship between programmed and actual infeed positions.

It was necessary to design a compliance model that predicted the overshoot position for a given value of system compliance. The overshoot position would be calculated so that the required amount of material would be removed from the workpiece during the infeed and dwell phases of the grinding cycle.

The combined effect of system compliance and grinding forces is characterised by the time constant '$\tau$' [60]. Over a number of grinding cycles a grinding wheel becomes dull and grinding forces increase. System deflection depends on grinding forces and a dull grinding wheel leads to an increase in deflection, an increase in system time constant and a deterioration in surface texture. Thus, for a given workpiece type, system time constant provides a measure of grinding wheel sharpness. To ensure the minimum time is spent dressing the grinding wheel it is important to ensure that the grinding wheel is dressed only when necessary to restore wheel form or workpiece surface texture values.

From Trmal [16], the grinding force ratio '$\frac{F_t}{F_n}$' is typically 0.5. Therefore an approximate value of normal force '$F_n$' may be obtained [31]:

$$F_n \cong \frac{2P}{v_s} \qquad 16.$$

The system stiffness '$K_t$' may then be derived:

$$K_t \cong \frac{2P}{v_s v_f \tau} \qquad 17.$$

where '$v_f \tau$' is the steady state deflection experienced at the force '$F_n$'. Given a value of time constant, derivation of the value of system stiffness permits calculation of system deflection.

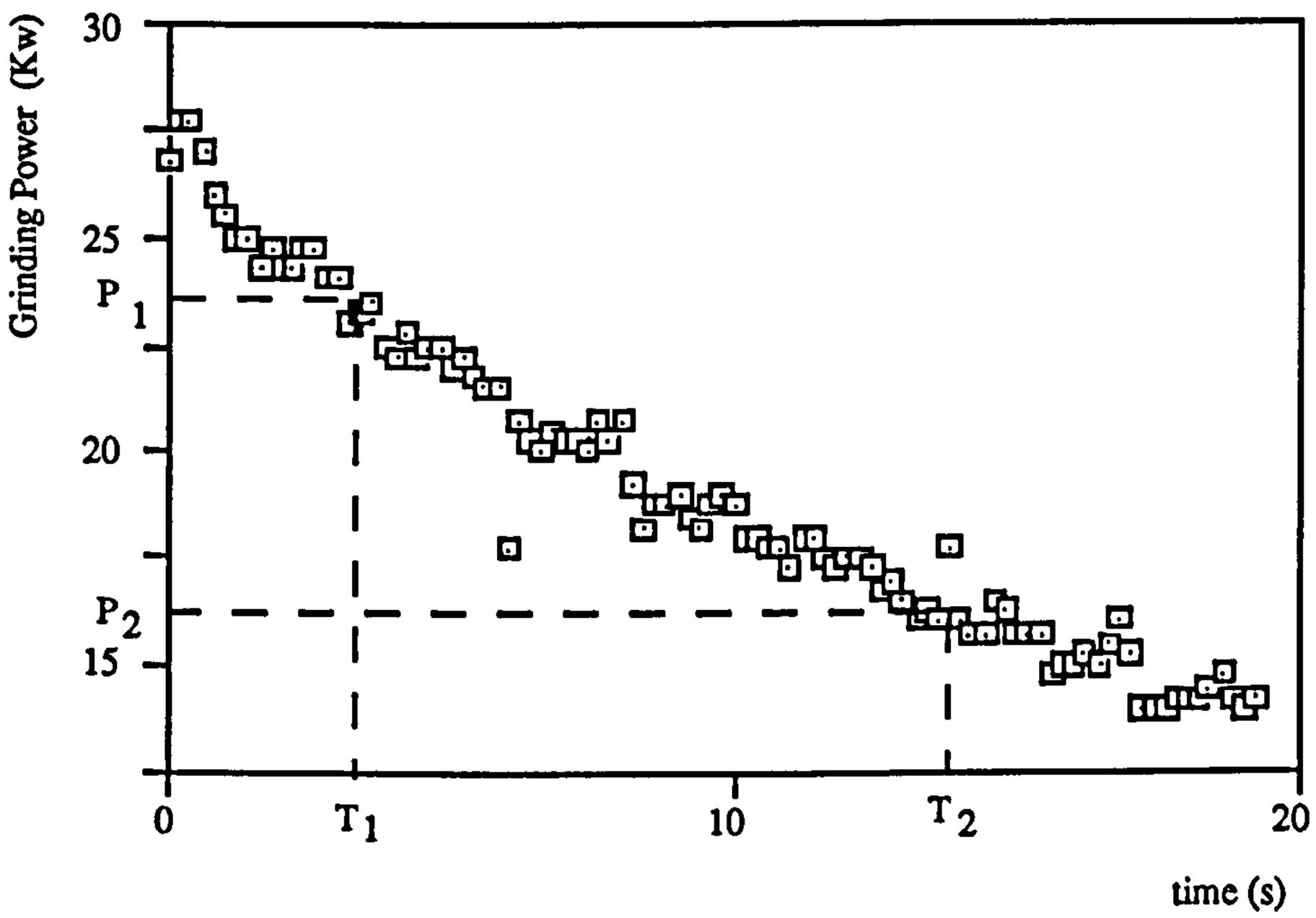



Figure 18. Grinding power during the dwell period.

The time constant may be determined by monitoring the exponential power decay during the dwell period. It is assumed for this purpose that the normal grinding force coefficient is constant during the dwell period, so that power is proportional to the depth of cut. The system time constant may be defined as [16] :

$$\frac{1}{\tau} = \frac{\ln\left(\frac{P_1}{P_2}\right)}{T_1 - T_2} \qquad 18.$$

where $P_1$, $P_2$, $T_1$ and $T_2$ are as shown in Figure 18.

Assuming a programmed infeed rate '$v_{fp}$' the system deflection '$x_i$' at time 't' is obtained from equation 19 [28]:

$$x_i = v_{fp}\,\tau\left(1 - e^{\left(\frac{-t}{\tau}\right)}\right) \qquad 19.$$

Steady state deflection occurs after an infeed time of approximately three time constants. Therefore, for infeed times less than three time constants the depth of cut and hence material removal rate is limited by system time constant. The actual material removed from the workpiece after grinding time '$t_g$' may be derived from the expression for true infeed position '$X_i$':

$$X_i = v_{fp}\,t_g - x_i = v_{fp}\,t_g - v_{fp}\,\tau\left(1 - e^{\left(\frac{-t_g}{\tau}\right)}\right) \qquad 20.$$

At the commencement of the dwell period the system deflection and hence the demanded depth of cut is equivalent to a step input to the programmed position. Thus, workpiece diameter reduction '$X_d$' during a dwell period is given by the equation:

$$X_d = x_i\left(1 - e^{\left(\frac{-t}{\tau}\right)}\right) = v_{fp}\,\tau\left(1 - e^{\left(\frac{-t_g}{\tau}\right)}\right)\left(1 - e^{\left(\frac{-t_d}{\tau}\right)}\right) \qquad 21.$$

where '$t_d$' is the dwell time. It was necessary to devise a technique for calculating the grinding time that ensured the sum of '$X_i$' and '$X_d$' was equal to the required material removal '$X_T$':

$$X_T = X_i + X_d = v_{fp}\,t_g - v_{fp}\,\tau\left(1 - e^{\left(\frac{-t_g}{\tau}\right)}\right) + v_{fp}\,\tau\left(1 - e^{\left(\frac{-t_g}{\tau}\right)}\right)\left(1 - e^{\left(\frac{-t_d}{\tau}\right)}\right) \qquad 22.$$

Solving the above expression for grinding time '$t_g$' was complex and so a bisection algorithm was designed and employed within the process model modules. A flow chart describing this algorithm is given in Figure 19. The calculated grinding time was

found to converge to sufficient accuracy within four or five iterations. From grinding time the overshoot position '$X_{OS}$' was derived as a function of the infeed rate.

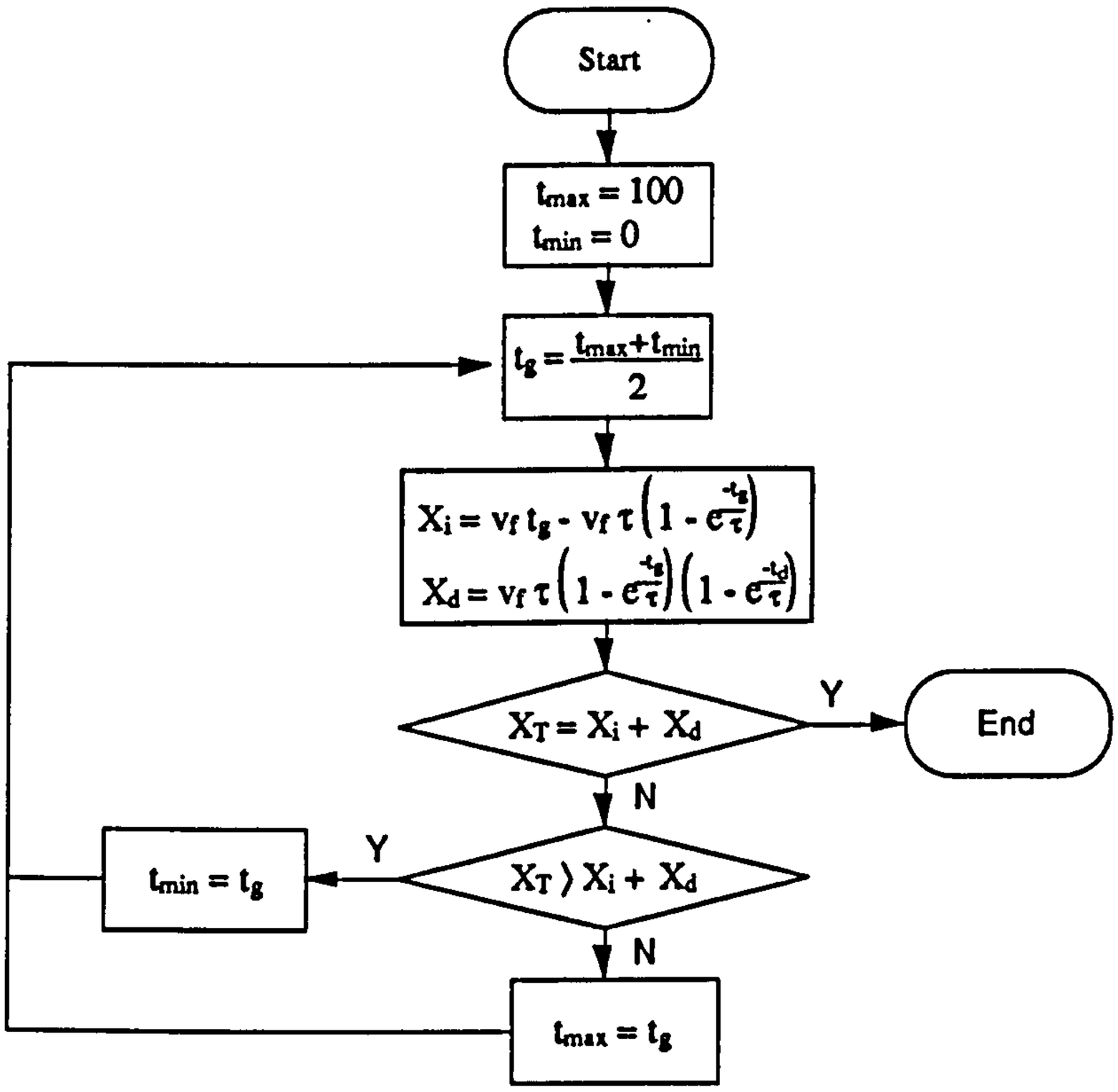



Figure 19. Flowchart for the calculation of grinding time.

The modelling of system compliance may be of little benefit if the compliance of individual workpieces varies significantly or system time constant varies rapidly due to changes in grinding conditions. In such a case the predicted overshoot may produce an out of tolerance workpiece. It was decided to analyse the effect of an error between actual and assumed values of system time constant. Figure 20 illustrates the overshoot target positions that are predicted by the compliance model for a typical grinding cycle with a dwell time of 10 s and an infeed rate of 0.1 mm/s. Predicted overshoot position is displayed against a range of values of system time constant for material removals of 0.3 mm, 0.15 mm and 0.05 mm. The graph indicates that a 5% error in the value of time constant results in an overshoot position error of approximately 25 μm when removing 0.3 mm of material. However, the error in overshoot position when removing 50 μm of material is approximately 5 μm for the same error in time constant.

Therefore, it can be inferred that errors between assumed and actual values of system time constant affect overshoot position less with lower material removals.

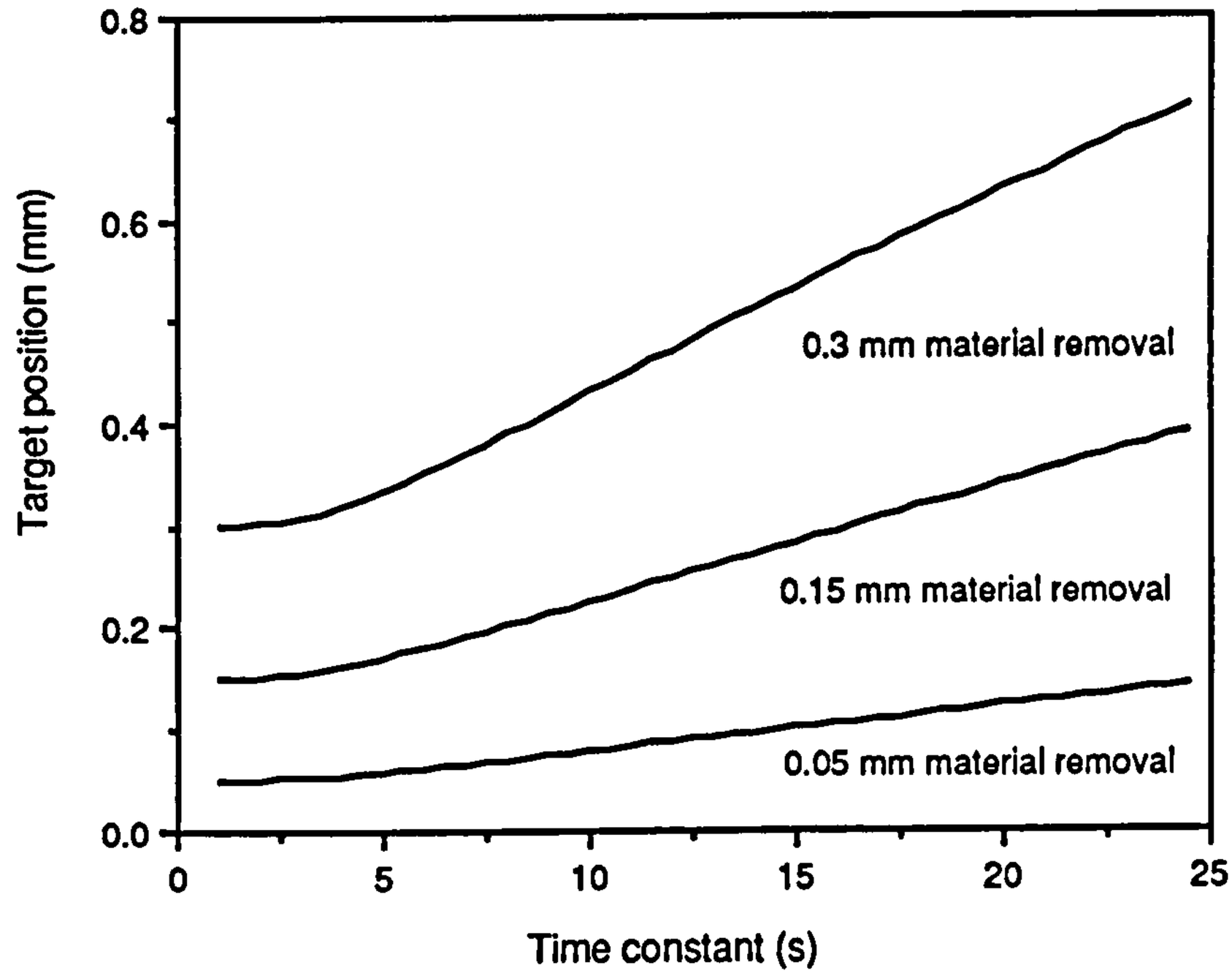



Figure 20. Variations in target position for varying time constant and material removal.

It was necessary to consider the effect of an error in overshoot position on material removal. Figure 21 illustrates the material removal predicted by the compliance model for a typical grinding cycle with a dwell time of 10 s and an infeed rate of 0.1 mm/s. Predicted material removal is displayed against target overshoot position for a range of values of system time constant. The graph indicates that the error in material removal that arises from an error in overshoot position is less than the overshoot position error. Also, errors in overshoot position have proportionally less effect on material removal for higher values of time constant.

Therefore, for highly compliant workpieces with values of time constant that vary from the average value by 5% it would be expected to experience an error in workpiece size of somewhat less than 5 μm when removing 50 μm of material. As the control system positioned the infeed axis to 4 μm it was apparent that variations in time

constant of 5% would have little effect on workpiece size. In order to produce workpieces of the required diameter it was preferable to determine the most accurate value possible of time constant before performing a finishing cycle. The material removed during a finishing cycle should be chosen to be small enough that the estimated workpiece error arising from the expected variations in time constant is within the required tolerance.

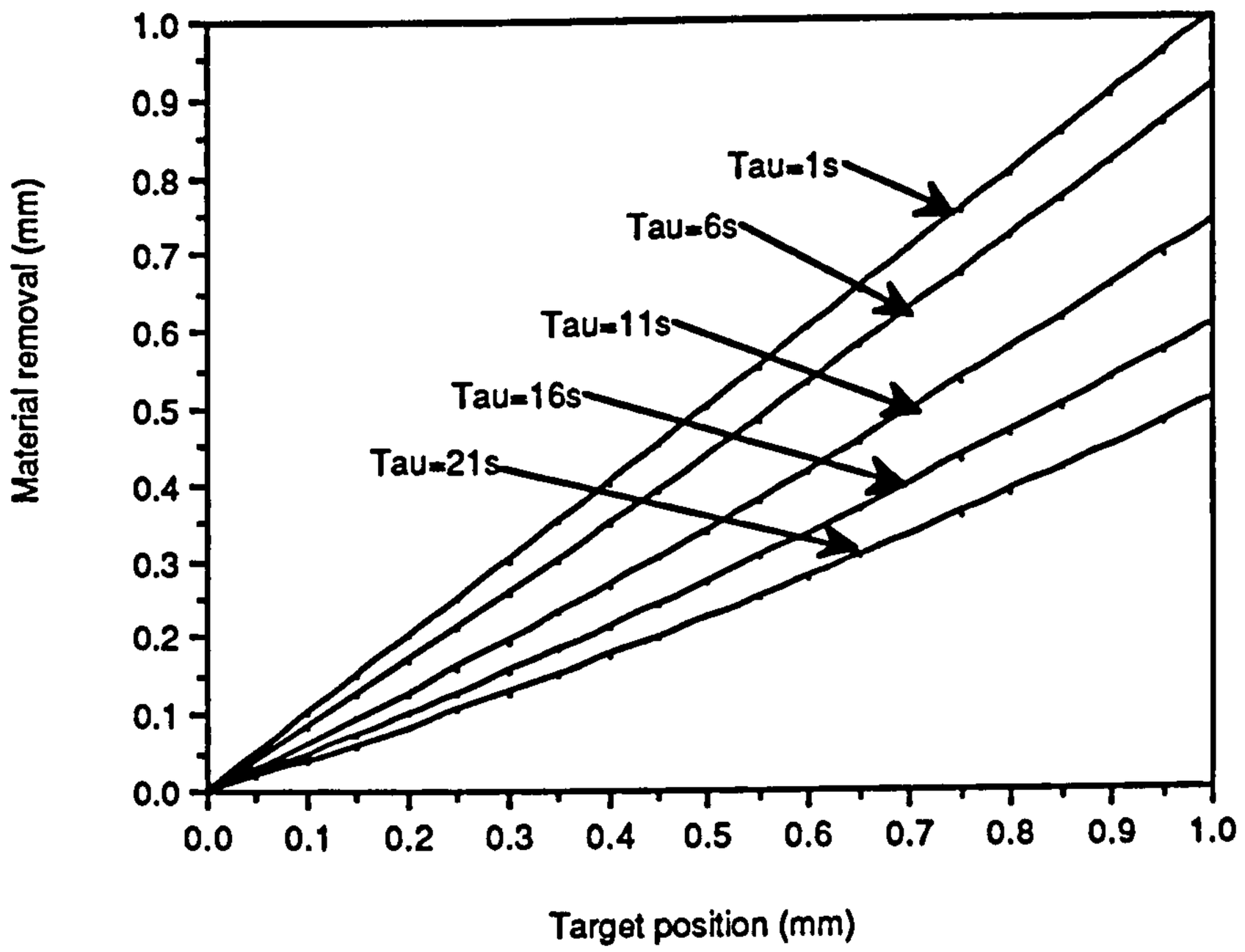



Figure 21. Variations in material removal for varying target position and time constant.

Workpiece roundness is controlled principally by dwell period. The infeed overshoot predicted by the compliance model varies inversely with changes in dwell time. Despite longer cycle times, increasing dwell time allows more time for workpiece rounding up and reduces workpiece distortion caused by overshoot. The workpiece rounding mechanism is well documented [8,14,20,21,23,60]. Figure 22 illustrates that target overshoot position and hence final workpiece size is less sensitive to changes in system time constant when longer dwell times are used. The value of dwell should be chosen to minimise cycle times whilst providing the demanded degree of workpiece roundness.

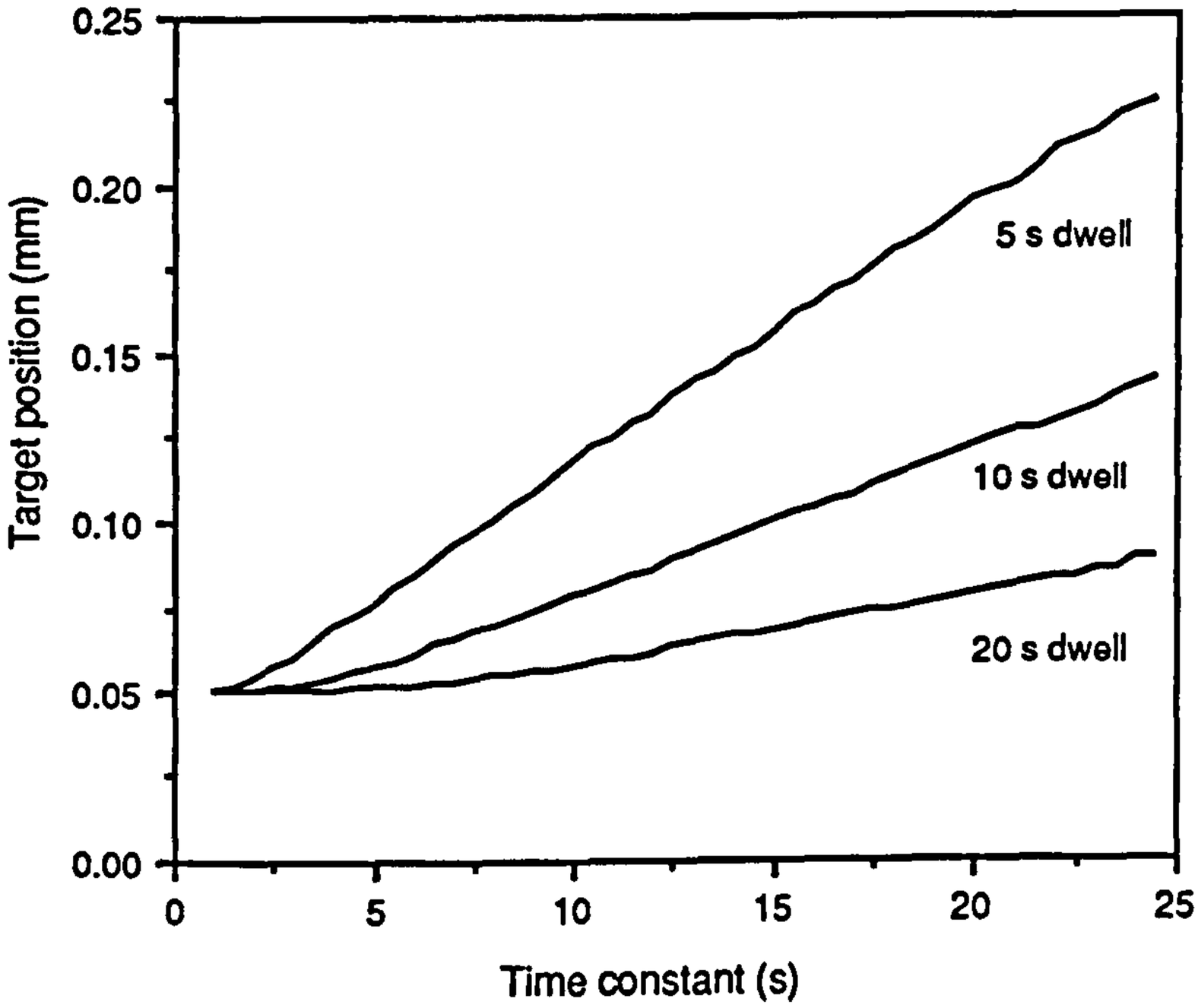



Figure 22. Effects of time constant on target overshoot position for varying dwell times.

The compliance model was incorporated within the intelligent control system structure to allow prediction of infeed overshoot position for given cycle parameters. Table 9 lists the parameters required by the process model for calculation of overshoot position. The compliance model resource comprised a CSI module written in a combination of PLM-86 and ASM-86 sub-routines.

| Parameter | Identifier |
|---|---|
| System time constant | $\tau$ |
| Programmed infeed rate | $v_{fp}$ |
| Programmed dwell time | $t_d$ |
| Workpiece material removal | $X_T$ |

Table 9. Input data for the compliance model.

## 5.3 Strategies for optimisation of grinding conditions

The controlled grinding parameters for both the Cincinnati No. 3 and Wickman 2K centreless grinding machines were infeed rate, work speed and grinding wheel speed. However, many commercial machines either have fixed grinding wheel speeds or operate at the maximum rated speed of the grinding wheel for safety reasons. Also, it was shown [33] that optimum grinding wheel speeds existed in centreless grinding for cast iron and easy to grind steels. For these reasons and to avoid over complication of the developed control system it was decided that a simple rule should be employed to select grinding wheel speed. The rule dictated that grinding wheel speed was set to the optimum speed for the material unless that value exceeded the maximum allowable speed of the grinding wheel specified by the manufacturer.

Limit chart theory, suggested by Rowe [33], was employed in the developed control system to provide a method for optimising grinding conditions. Limit charts describe the boundaries of the centreless grinding process in terms of the controlled grinding parameters of infeed rate and work speed. Figure 5 shows a limit chart for cast iron with the process boundaries of machine power, thermal damage and chatter. A study of the optimisation process showed that for maximum removal rate and avoidance of thermal damage, specific energy was minimised.

For the centreless grinding process maximum removal rate occurred at the junction of the machine power and thermal damage boundaries. The maximum power available from the machine tool was determined by the type of grinding wheel drive motor fitted. Measurement of machine power consumption by the control system was therefore used to provide an indication of the distance to the machine power boundary. The values of the controlled grinding parameters of infeed rate and work speed were fixed during a grinding cycle. This ensured that changes in grinding conditions were minimised for an individual workpiece. After each grinding cycle the control system adjusted the controlled grinding parameters according to the power and specific energy achieved.

Equation 4 indicates that increased removal rate requires a proportional increase in power consumption. Therefore, assuming a rigid machine tool and constant grinding conditions, infeed rate is proportional to power consumption. Based on this assumption a rule was developed that used the value of infeed rate '$v_f$' to approach the machine power boundary '$P_{max}$' using the following expression:

$$v_f = v_{f\,old}\left(1 + k\left(\frac{P_{max}}{P} - 1\right)\right) \qquad 23.$$

where 'P' is the power achieved during the grinding cycle that was performed using infeed rate '$v_{f\,old}$'. A value of constant 'k' between 0 and 1 smoothed the adjustments made to infeed rate to account for process and workpiece variations. Due to the wide variation in the size and compliance of cylinder liners supplied by GKN it was decided to use a 'k' value of 0.2. Higher values of 'k' provide a more dynamic system suitable for workpieces with less variable compliance.

However, the position of the thermal damage boundary was not available from a machine input and its position was required to be inferred from the thermal model. Approach towards the thermal damage boundary was controlled by adjustments to the controlled grinding parameter of work speed '$v_w$'. The change in specific energy that was achieved after a change in work speed was dependent on grinding forces achieved during the grinding cycle. Therefore, for a given infeed rate it was not practical to calculate a value of work speed that placed the grinding conditions at the boundary of thermal damage.

To approach the thermal damage boundary predicted by the thermal model, the developed control system varied work speed in fixed increments. If the value of specific energy achieved during a grinding cycle was less than the predicted critical specific energy, work speed was reduced by an increment. The work speed increment '$\Delta v_w$' was chosen to provide a satisfactory optimisation rate whilst providing an element of smoothing to account for process and workpiece variations. From experience it was possible to program a rule for minimum safe work speed below

which roundness problems were encountered. Similarly, a value of maximum work speed was used to prevent chatter problems. These additional boundaries were incorporated into the process model in the form of 'rules'. After reducing '$v_w$' a check was performed to ensure that the value of critical specific energy predicted by the thermal model was not increased. However, as process efficiency is not highly dependent on '$v_w$' it was decided to rely on a safe distance from the burn boundary.

The strategy for optimisation of grinding conditions produced improved values of the grinding parameters of infeed rate and work speed after each grinding cycle. Figure 22 illustrates the optimisation process.

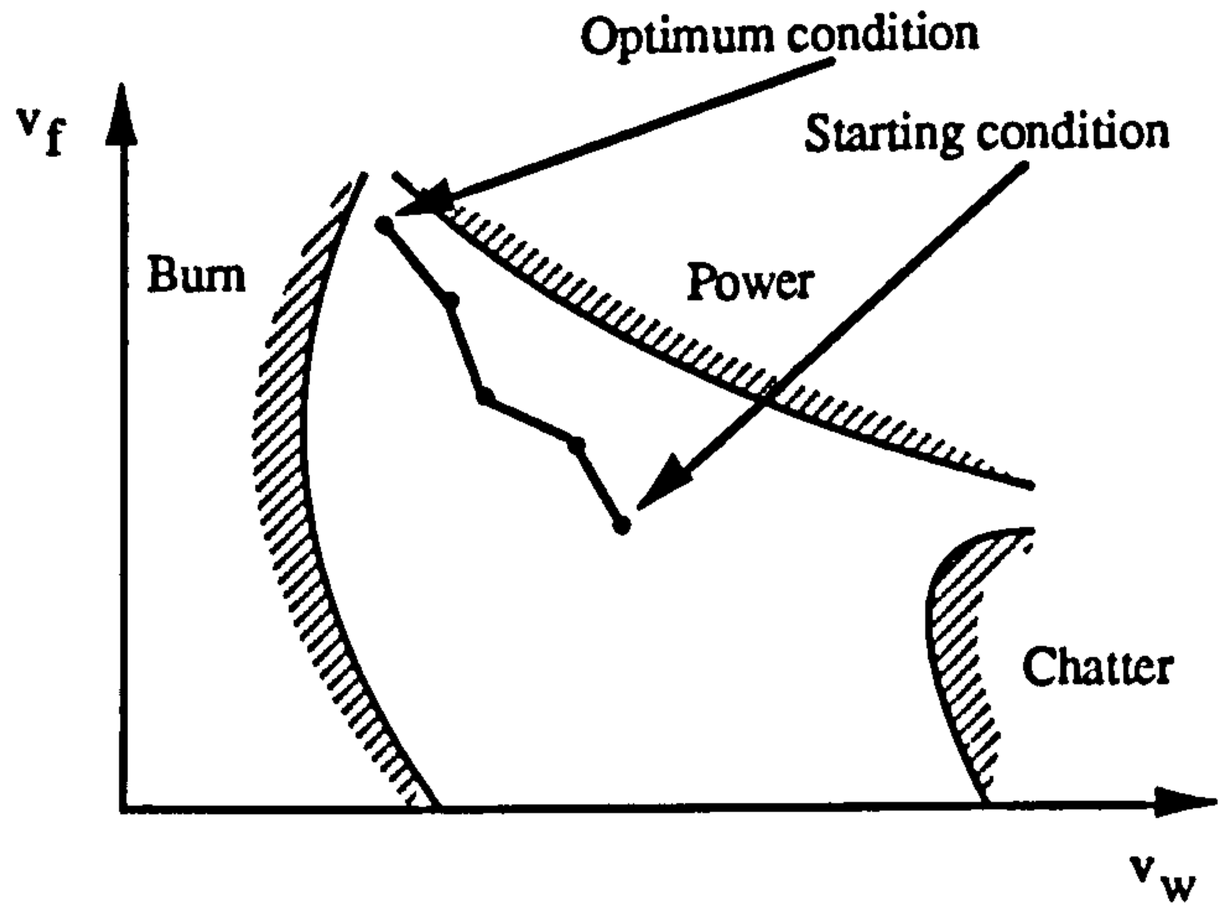



Figure 22. Optimisation of grinding conditions through adjustment of infeed rate and workpiece speed.

A learning strategy was employed to ensure that the improved grinding conditions were duplicated for other workpieces of the same material. This was achieved using the kinematic model to calculate new values of the kinematic parameters chip shape ratio '$a_r$' and mean chip volume '$v_m$'. These kinematic parameters were determined from information in the data base concerning grinding wheel and workpiece geometry together with the improved grinding parameters of infeed rate and work speed. The materials file of the data base was updated with the new values so that improved grinding conditions were stored. Using this technique it was possible to

learn grinding conditions for a new material given suitable starting values of kinematic parameters. A flow chart of the optimisation process is given in Figure 23.

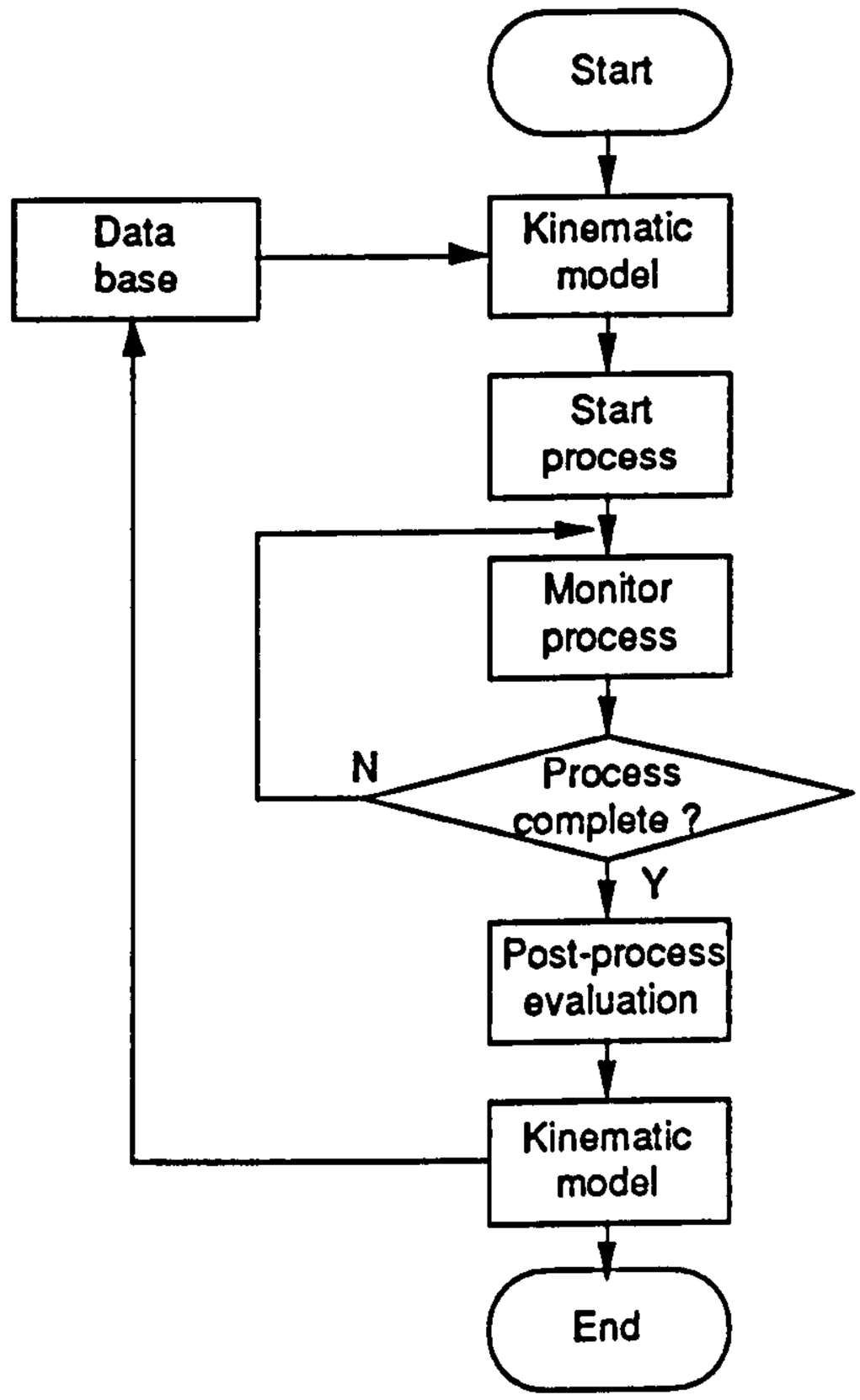



Figure 23. A flow chart of the grinding condition optimisation process

## 5.4 Machine cycle strategy design

In parallel with the incorporation of process modelling and learning strategies within the new control system conceptual framework it was necessary to design a machine control cycle for production of the highly compliant, cast iron cylinder liners produced by GKN. In a conventional grinding cycle the infeed mechanism starts at a 'home' position and approaches a workpiece at rapid traverse rate before reducing to the grinding infeed rate at a preset safety position termed the high speed break point. In some cases the high speed break point may be determined in-process by a gap elimination device as discussed in Chapter 3. The infeed is then driven to a target position corresponding to the required workpiece diameter at the programmed grinding

infeed rate. Generally the infeed remains at the target position for a dwell time in order to allow system deflections to reduce and correspondingly workpiece diameter to approach the required value. The dwell time may be either fixed or subject to a signal indicating that the workpiece size has been achieved from an in-process workpiece sizing device. After the dwell period the infeed retracts to the home position for unloading of the workpiece. Figure 16 illustrates a conventional plunge grinding cycle.

In order to allow a comparison between conventional plunge grinding of rigid and compliant workpieces a conventional grinding cycle was defined in the machine cycle module of the new control system structure. The executor routine passed from the data base to the cycle module values of the following parameters:

1. Initial workpiece diameter
2. Target workpiece diameter
3. Safety distance
4. Home distance
5. Infeed rate
6. Grinding wheel speed
7. Work speed
8. Dwell time

The machine cycle routine calculated the home position as the sum of the initial workpiece diameter and home distance and the high speed break point as the sum of the initial workpiece diameter and safety distance. For the purposes of grinding the thin walled cylinder liners produced by GKN it was clear that conventional grinding cycle design was not satisfactory. The high level of compliance and low grinding forces ensured that dwell times required to achieve workpiece size were excessive and resulted in unacceptable production rates. For this reason it was necessary to design an alternative machine cycle strategy that minimised dwell times by using an infeed overshoot position.

A machine cycle was added to the machine cycle module of the control system that operated by calculating the overshoot position for a workpiece from values stored in the data base. The overshoot position was calculated by adding a fixed value of overshoot distance to the distance to the target position for the required target diameter. Although use of a fixed overshoot position results in reduced dwell times and increased removal rates, problems were experienced in achieving accurate workpiece size. Principally, in order to produce a workpiece to a specified size accuracy it is necessary to fine tune the overshoot distance to an accuracy with a similar order of magnitude. Once this has been achieved any small change in grinding conditions arising from grinding wheel wear or from variations in the workpiece requires a further adjustment of the overshoot distance. It was experienced that variations in the internal and external diameters of workpieces produced by GKN led to varying compliance. Also, it was required that the performance of the system in terms of the size accuracy of workpieces was as far as possible independent of grinding conditions. It therefore became necessary to consider methods for calculating an overshoot distance for each workpiece individually and to compensate for changes in grinding conditions over time.

In order to allow the grinding cycle to be customised for individual workpieces there was a requirement for an in-process measurement system. Adoption of suitable sizing devices allowed comparative measurements of workpiece diameter to be made before, during and after the grinding cycle. An early approach was to simply vary the overshoot distance in proportion to the difference between the measured initial workpiece diameter and the diameter of a nominal master workpiece. This approach was considered unsatisfactory as it failed to account for variations in internal diameter and also changes in grinding conditions. However, automatic measurement of workpiece size before a grinding cycle allowed the safety distance to be determined for each individual workpiece. This strategy termed gap elimination ensured that non-productive time spent at a low infeed rate was minimised and provided a reduction in overall cycle times.

A strategy was devised that combined the compliance model [28] with workpiece size measurement to compensate for variations in the size of cylinder liners prior to grinding. In place of a nominal overshoot position an overshoot distance was derived for each cycle from the system compliance model. On the basis of a current value of system time constant '$\tau$' stored in the data base and a fixed dwell period '$t_d$', the compliance model was used to predict an overshoot position that would remove the required amount of material from a workpiece. The required workpiece diameter reduction was determined by subtracting the target workpiece diameter from the workpiece diameter measured by the sizing device.

The compliance model relied on the value of system time constant to predict overshoot positions. As stated earlier, there were significant variations in compliance between cylinder liners supplied by GKN. Also, system time constant varies due to changes in grinding conditions. Consequently errors in the diameter of the workpieces were experienced following a grinding cycle where a fixed value of time constant was assumed. A requirement for learning the current value of system time constant was therefore identified.

Whilst it was possible to calculate time constant using the exponential power decay method described in equation 18, a simpler method was employed. This technique relied on measurement of workpiece diameter by the sizing device following a grinding cycle. An algorithm based on the compliance model equations was designed and incorporated within the learning phase of the developed control system. This algorithm re-calculated system time constant by searching for the value of time constant that predicted the measured workpiece diameter error.

Once the learning phase had utilised the compliance model to learn the new value of time constant this value was filtered and stored by the executor routine in the data base. A simple filter was employed of the type described in expression 24. The value of filter constant 'k' was chosen to provide a stable response. A nominal value of 0.2 was found to provide acceptable performance. Koenig [5] stated that a system

employing a process model to learn the controlling parameters for that model was self-aligning.

$$\tau_{new} = \tau_{old} + k\,(\tau_{new} - \tau_{old}) \qquad 24.$$

As infeed rate and grinding time were determined by the control system dynamically, only the value of system time constant was stored in the control system data base. Experience indicated that the machine compliance was small in comparison with that of the cylinder liners produced by GKN. However, compliance varied between different types of cylinder liners. Therefore, a value of time constant was stored for each of the workpieces defined in the control system data base.

In order to account for varying workpiece compliance a grinding cycle known as the 'pecking' cycle was developed and incorporated within the control system. The principle behind the pecking cycle was to allow the control system to evaluate the compliance of an individual workpiece. This was achieved by dividing the grinding cycle into a number of defined stages. Each stage or peck comprised the following sequence: measure workpiece diameter, rapid traverse to safe distance from workpiece, decelerate to grinding feed rate, feed to overshoot position, dwell, rapid retract and measure the new workpiece diameter. Measurement of workpiece size was performed after a short delay to allow the workpiece to cool and thermal expansion to reduce.

The requirement for a cycle with at least three pecks was identified. The purpose of the first peck was to correct any workpiece roundness errors so that an accurate value of diameter was achieved prior to the second peck. The second peck was effectively a roughing cycle that permitted an accurate calculation of system time constant from the workpiece diameter error following the cycle. Thus, the third peck could be used to produce a finishing cycle based on the measured value of system time constant for that workpiece. However, workpiece compliance increases as material is removed during a grinding cycle. Also, workpiece compliance depends on the wall thickness of the cylinder liner which varied by up to 8% in the case of the workpieces

supplied by GKN. The rate at which workpiece compliance increases during a grinding cycle depends on both these variables. Consequently, although a three peck cycle provides an accurate measurement of system time constant after the second peck, it is not clear how the cylinder liner wall thickness will affect time constant during a subsequent peck. It is possible to account for this effect by considering the difference in values of time constant experienced during the second and third pecks. Thus, a fourth peck could be used to account for the expected rate of change of workpiece compliance. However, from the graphs of Figures 20 and 21 it is clear that the effects of changes in time constant are small when removing small amounts of material. Therefore, the final effect on workpiece size of an error between the measured value of time constant from the second peck and the actual value of time constant for the third peck is reduced with a small final cut. Furthermore, grinding wheel wear during a small cut is minimised and the low grinding forces ensure that thermal effects on workpiece size are reduced. Other advantages of a small final cut include that workpiece shape is improved as the depth of cut is low at the point of retraction. Due to the constraints of production rate it was decided to account for workpiece compliance using a three peck cycle.

The three peck cycle was constructed so that a learned and filtered value of system time constant was stored in the workpiece data file for each of the programmed pecks. Following each peck the measured value of system time constant was used to modify the time constant used for the subsequent peck. For example, if the workpiece diameter error following a peck showed the system time constant to be 1.1 times that of the stored value for that peck, the time constant for the next peck was modified by a factor of 1.1. The system was designed so that the control system user programmed the required material removal for each peck of the grinding cycle. For simplicity, it was arranged that the control system user programmed a value of dwell to be applied to all of the pecks. A diagram illustrating a pecking cycle is given in Figure 24.

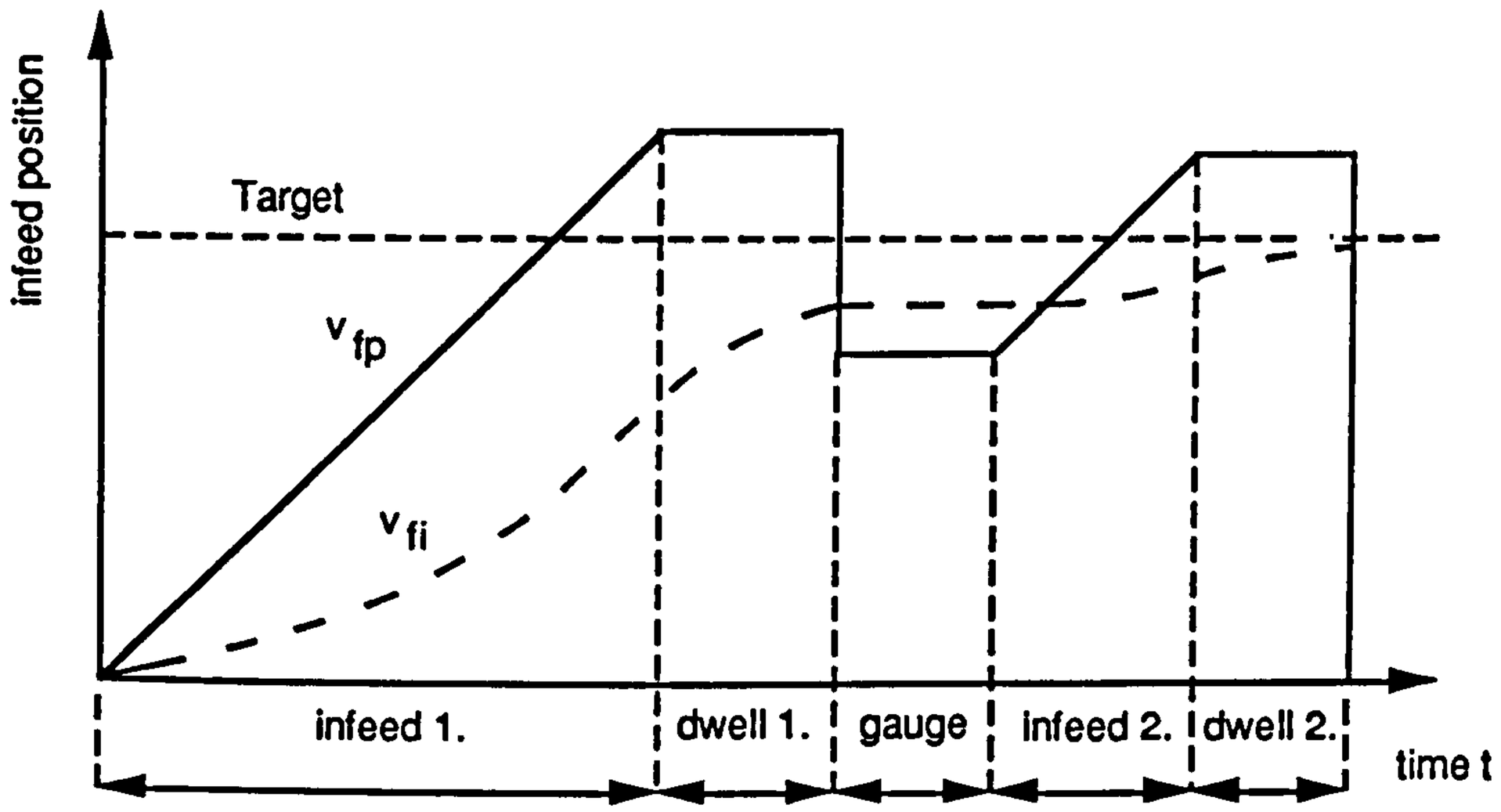



Figure 24. A schematic diagram of a two peck 'pecking' cycle

# Chapter 6. Development of equipment

A substantial modification of the manually controlled centreless grinding machine was necessary before it was suitable for automated operation. This chapter describes the machine development and control system integration.

## 6.1 Machine development

A schematic diagram of the main elements of the manually operated Cincinnati No. 3 centreless grinding machine is given in Figure 3. The machine featured two slide ways. The upper slide supported the control wheel whilst the lower slide supported the workplate. It was possible to clamp either of the slides and so positioning of the control wheel was performed with either the workplate fixed or with the workplate moving. Control of the angle in the horizontal plane between the control wheel and the grinding wheel was made by adjusting the skew angle between the slide ways and the machine tool bed.

Originally the grinding wheel was driven by a 16 kW fixed speed a.c. motor through belt driven pulleys with a 1:1 ratio between the motor spindle and the grinding wheel. A surface speed of 33 m/s was achieved with a 609 mm diameter grinding wheel. Variations in grinding conditions were caused by wear of the grinding wheel. The grinding wheel diameter reduced with wear leading to a decrease in wheel surface speed. A wheel was often worn to half its original diameter before replacement thus halving the effective cutting speed. A variable speed motor was therefore required to allow the control system to maintain surface speed by increasing wheel speed as the diameter decreased with wear.

According to Rowe [33] an improvement in grinding conditions could be achieved by increasing grinding wheel speed to 45 m/s. A spindle that was capable of speeds in excess of 1400 rev/min was required to achieve a target surface speed of 45 m/s with a 609 mm diameter grinding wheel.

A limiting factor in the realisation of high material removal rates was the low level of power available from the grinding wheel drive motor. An increase in power from the drive system would allow increased removal rates before the grinding wheel was in danger of stalling. For the centreless grinding process, maximum removal rate occurs at or near the burn boundary [33]. Therefore, to determine the power required from the drive system it was necessary to consider the specific energy achieved during a grinding cycle. For a workpiece of material EN9 it was shown [44] that burn occurred at a specific energy level of 46 J/mm$^3$. The workpiece diameter '$d_w$' was 38 mm, the length 'b' was 70 mm and the infeed rate '$v_f$' was 0.055 mm/s. Specific energy '$e_c$' was determined from the following equation:

$$e_c = \frac{2P}{\pi d_w b v_f} \qquad 25.$$

Rearranging equation 25 allows the power required to achieve burn to be predicted:

$$P = \frac{\pi \times 46 \times 38 \times 0.055 \times 70}{2} = 10571W \qquad 26.$$

It was possible to maximise specific removal rate for a workpiece of similar dimensions and material to the example workpiece as 16 kW was available from the original grinding wheel motor. However, the machine tool was capable of producing workpieces with a length of up to 254 mm with a standard grinding wheel. Production of a workpiece 3.6 times longer than that of the example workpiece required a proportional increase in power (38 kW) to achieve similar specific removal rates. Therefore, a more powerful grinding wheel drive was required for high removal rate grinding on workpieces using the full capability of the machine.

In order to meet the requirement for a variable speed, high power grinding wheel drive system the following modifications were made. The grinding wheel drive motor was replaced with a 75 kW d.c. motor powered by a KTK 6PS thyristor drive amplifier. The amplifier provided infinitely variable speed control of the motor up to

2000 rev/min. Speed of the motor spindle was set by a 0-10 V reference voltage whilst a d.c. tachometer provided closed loop speed control of the motor.

A direct drive from the motor to the grinding wheel was developed to replace the standard pulleys and thus reduce transmitted vibrations from the original drive belts. The direct drive also reduced loadings on the grinding wheel head bearings when grinding with high surface speeds and power levels. A schematic diagram of the arrangement is given in Figure 25. The motor was mounted on a large concrete plinth and shimmed to provide accurate alignment with the grinding wheel spindle. The grinding wheel spindle was connected to the motor with a shock/vibration absorbing flexible coupling in order to protect the spindle bearings.

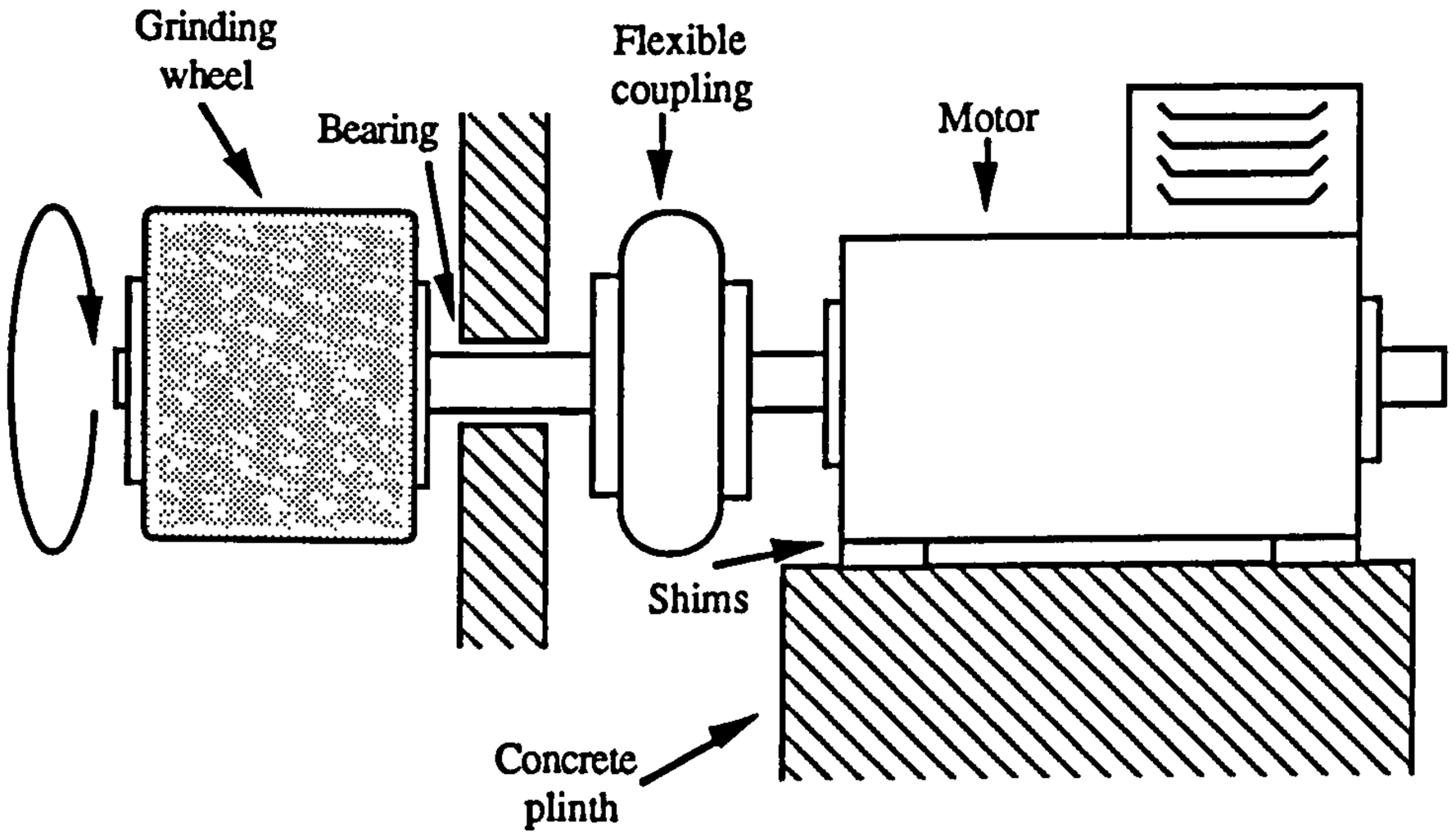



Figure 25. A schematic diagram of the grinding wheel drive arrangement.

The developed grinding wheel drive arrangement allowed surface speeds of up to 64 m/s with a 610 mm diameter grinding wheel. However, the standard Universal grinding wheels that were used by GKN were rated to 45 m/s. Therefore, to prevent damage to the grinding wheel it was necessary to ensure that the control system limited maximum spindle speed. In order to minimise vibrations the rotating elements of the drive system were statically balanced.

Whilst running the grinding wheel for extended periods at 1430 rev/min it was found that the wheelhead bearings overheated. The wheelhead bearing temperature was measured with thermo-couples and found to rise to 67°C after 90 minutes. As the maximum specified working range of both the bearings and the lubricating oil was from 35°C to 60°C a cooling system was required. It was determined that the machine was producing 2-2.5 kW of heat and that an oil cooler with a capacity of 10 kW/100°C was therefore necessary. After a cooler was fitted the wheelhead bearing temperature remained within the specified boundaries. The grinding wheel drive mechanism was thus capable of satisfying the requirements proposed above. Infinitely variable speed allowed the control system to maintain constant surface speed regardless of wheel diameter. The extra speed allowed by the direct drive and motor allowed grinding at more efficient grinding wheel surface speeds. Finally, the increase in power provided by the motor and amplifier allowed higher removal rates than was possible with the original machine.

As with any cutting tool a grinding wheel becomes dull after a period of use. A dull wheel typically requires more power to achieve the required removal rates and workpiece surface texture may be adversely affected. A technique known as dressing was used to 'sharpen' the grinding wheel surface. Grinding wheel dressing was performed by traversing a diamond mounted on a quill across the face of the wheel using a hydraulic ram. Depth of cut of the quill mounted diamond dressing tip was adjusted via a threaded quill assembly. Dressing lead, which is measured in mm/rev of the grinding wheel, was varied by altering oil flow to the hydraulic ram. The effect of the dressing process was to cut a very fine screw thread onto the grinding wheel thus presenting new cutting edges on the surface. Dressing of the grinding wheel required a skilled operator in order to maintain consistency due to the lack of positional and speed feedback from the dressing mechanism.

Investigation of grinding wheel dressing was not an aim of the project as the effects of wheel dressing conditions are well documented [6,36,61,62]. Automation of the grinding wheel dressing mechanism was desirable in order to maintain consistency. However, the type of grinding performed by GKN meant that dressing was only required relatively infrequently. Typically more than 50 grinding cycles were performed before grinding conditions changed enough to warrant dressing. Project objectives included that consistent results were to be achieved throughout the period between dresses and to consider extending the dressing interval as dressing time is non-productive. As a result of these factors and financial restraints time was not spent modifying this area of the machine and the original dressing mechanism was retained.

It was decided that optimisation of grinding conditions would require in-process measurement of grinding wheel power. Although force measurement provides valuable process information, it was considered to be more expensive and less reliable than power measurement. A power signal was not available from the KTK 6PS d.c. thyristor drive electronics and it was therefore necessary to devise a technique for measuring power consumption. In a d.c thyristor drive a d.c voltage is generated by rectification of the 415V three phase mains voltage. Amplifier control circuits use phase angle triggering of thyristors to vary the average voltage level applied to the motor armature. Phase angle adjustment is performed by drive control circuits according to motor speed feedback from the d.c. tacho-generator mounted on the motor and motor load feedback provided by small current transformers. The amplifier varies the average d.c voltage to the armature and maintains the demanded motor speed. The motor power was therefore a function of the complex voltage and current waveforms.

Measurements were performed that showed the r.m.s. armature voltage '$V_{rms}$' was related to motor speed '$\omega(t)$' by a constant '$k_1$':

$$V_{rms} = k_1 \omega(t) \qquad 27.$$

Therefore, r.m.s. power '$P_{rms}$' was determined by measuring r.m.s. current '$I_{rms}$' and motor speed:

$$P_{rms} = k_1 \, \omega(t) \, I_{rms} \qquad 28.$$

Given measurements of motor current and grinding wheel speed it would be possible for the control system to calculate the grinding power. It was decided to use a Heme Hall-Effect transducer to measure the motor armature current. The transducer provided an analogue voltage that was proportional to the current passing through the armature cable. However, use of an oscilloscope indicated that noise on the analogue signal was caused by switching of the thyristors. In order to generate a clean power measurement signal a simple electronic filter was designed and implemented. Consequently, a stable analog voltage proportional to r.m.s. armature current was available. It was decided to measure grinding wheel speed by fitting a rotary encoder to the drive motor as derivation of power required that the current signal was scaled in proportion to the motor speed. The analog current signal and grinding wheel encoder were interfaced to the C.N.C. control system. A CSI routine was created that calculated grinding wheel power from the monitored values of armature current and grinding wheel speed. This method provided power measurement with an accuracy of approximately 5% when compared with readings achieved by a digital Siemens Function-Meter. Relating power to current and motor speed provided a cheap and efficient way of measuring power as both measurements were readily achieved. Commercial power meters capable of handling signals generated by high speed thyristor triggering were prohibitively expensive.

A CSI routine was created that used the value of power measurement to ensure that grinding power did not exceed the capability of the grinding wheel drive and cause the grinding wheel to stall. The control system software automatically reduced the infeed rate if the power level exceeded a preset limit. If, after a short delay, the power had not reached a safe level the infeed was retracted and the grinding cycle aborted.

The control wheel on a centreless grinding machine regulates the surface speed of the workpiece. An incorrect value of control wheel speed can cause the workpiece to burn or chatter whilst a more suitable value improves removal rates. A change of workpiece diameter required a proportional change in control wheel speed in order to maintain surface speed.

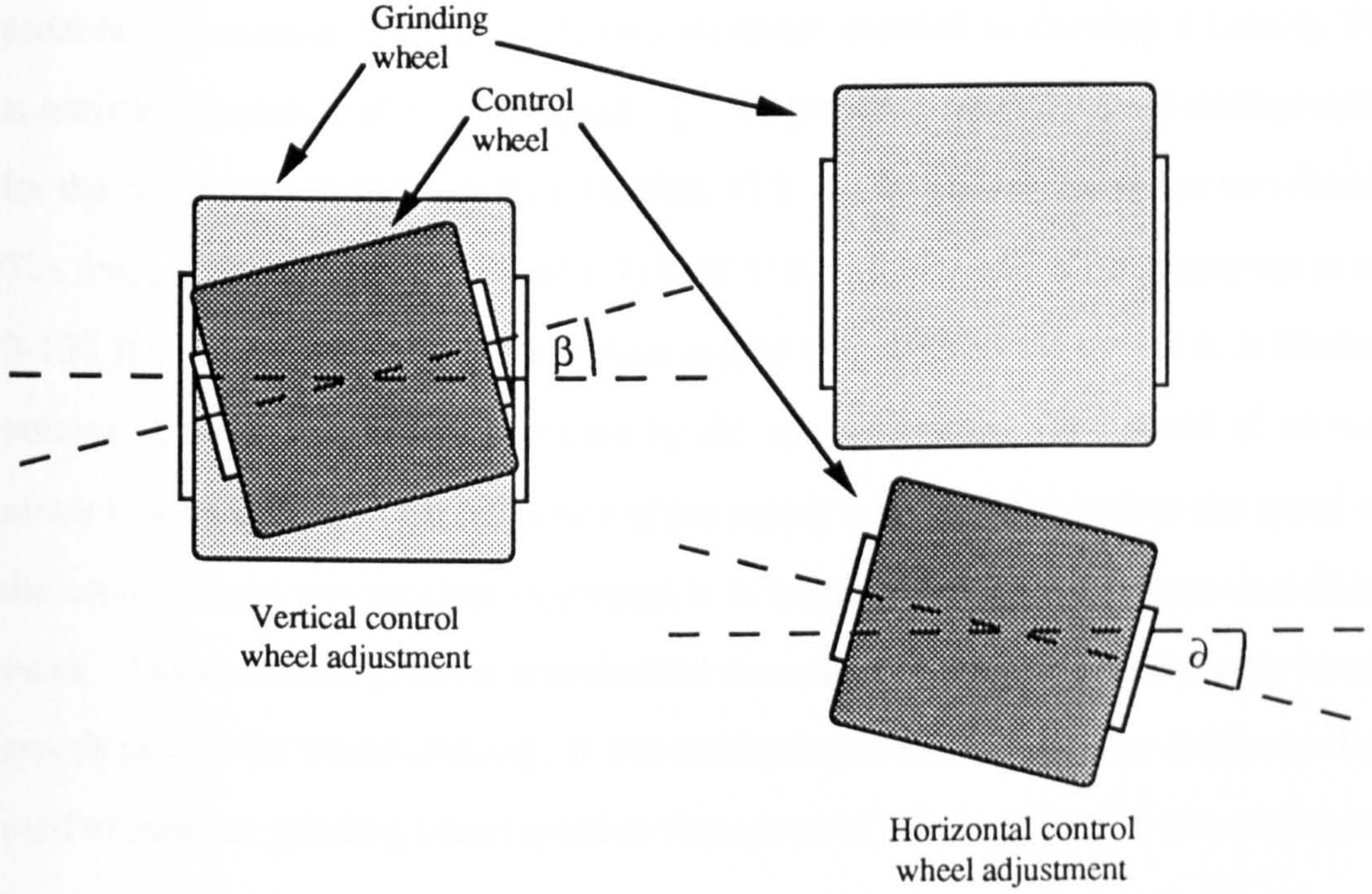



Figure 26. Planes of control wheel adjustment.

Originally a small 50 Hz 3 phase a.c. motor was used to drive the control wheel through a two speed gearbox. The speed of the control wheel and hence workpiece surface speed was varied by changing gears in the gearbox. Dressing of the control wheel was by a similar method to that of the grinding wheel. In order to achieve effective dressing the control wheel speed was increased using the high-ratio selector on the gearbox. Control wheel dressing was normally only performed after the control wheel surface was damaged. Therefore, automation of the dressing mechanism was not considered necessary. The control wheel head was adjustable for angle in both the horizontal and vertical planes. Figure 26 shows the plane of these adjustments with 'ß' in the vertical plane and '∂' in the horizontal plane. Horizontal adjustment

compensated for workpiece taper. Vertical adjustment provided a thrust on the workpiece in the axial direction to maintain workpiece contact with the end stop during the grinding cycle. The angular adjustments were performed manually. It was decided that automation of these adjustments was beyond the scope of this project.

Selection of appropriate values of workspeed was required as part of the process optimisation strategies. It was therefore decided to develop a facility for automatic adjustment of the workspeed '$v_w$'. To provide a variable speed arrangement for the control wheel mechanism a Danfoss VLT 101 frequency convertor was fitted. The frequency convertor generated a 3 phase 415 V supply with a frequency range of 0-100 Hz. The frequency of the output supply was determined by a d.c. reference voltage in the range 0-10 V provided by the control system. The speed of an a.c. motor is proportional to the frequency of the supply voltage and therefore the speed of the control wheel was continuously variable in the range 0-200% of the nominal fixed value. As the original gearbox was retained the system was capable of the high wheel speeds needed for wheel dressing. It was decided to fit a rotary encoder similar to that used to measure grinding wheel speed so that speed of the control wheel was measured by the control system.

Although the dynamic response of the frequency convertor was not as good as is available from more modern type of drives, it was considered satisfactory for the application. Control wheel speed was constant during a grinding cycle and hence good dynamic performance was not needed. The relatively low power consumption of the control wheel motor coupled with the gearing between the wheel and motor meant that control wheel speed did not vary significantly when loaded.

To remove material from a workpiece on a centreless grinding machine the workpiece is forced against the grinding wheel. This is achieved by positioning one of the two slideways with respect to the machine tool bed. Positioning of the upper slide with the lower slide clamped to the bed causes the workpiece to roll up the angled workplate and against the grinding wheel. Similarly, positioning with the lower slide

clamped to the upper slide also forces the workpiece against the grinding wheel. There was no consensus as to which approach was most satisfactory and the choice of technique was largely dependent on individual operational requirements. However, in the latter case the relative positions of the workplate and control wheel remain constant and changes in the work support geometry during a grinding cycle are minimised. In this latter condition the change in geometry between the workpiece, grinding wheel and control wheel during the cycle is due principally to workpiece size reduction. Infeed of the control wheelhead on the Cincinnati No. 3 centreless grinding machine was originally performed by an operator with a handwheel. Automation of the infeed mechanism was considered essential to allow accurate and repeatable control of position and infeed rate.

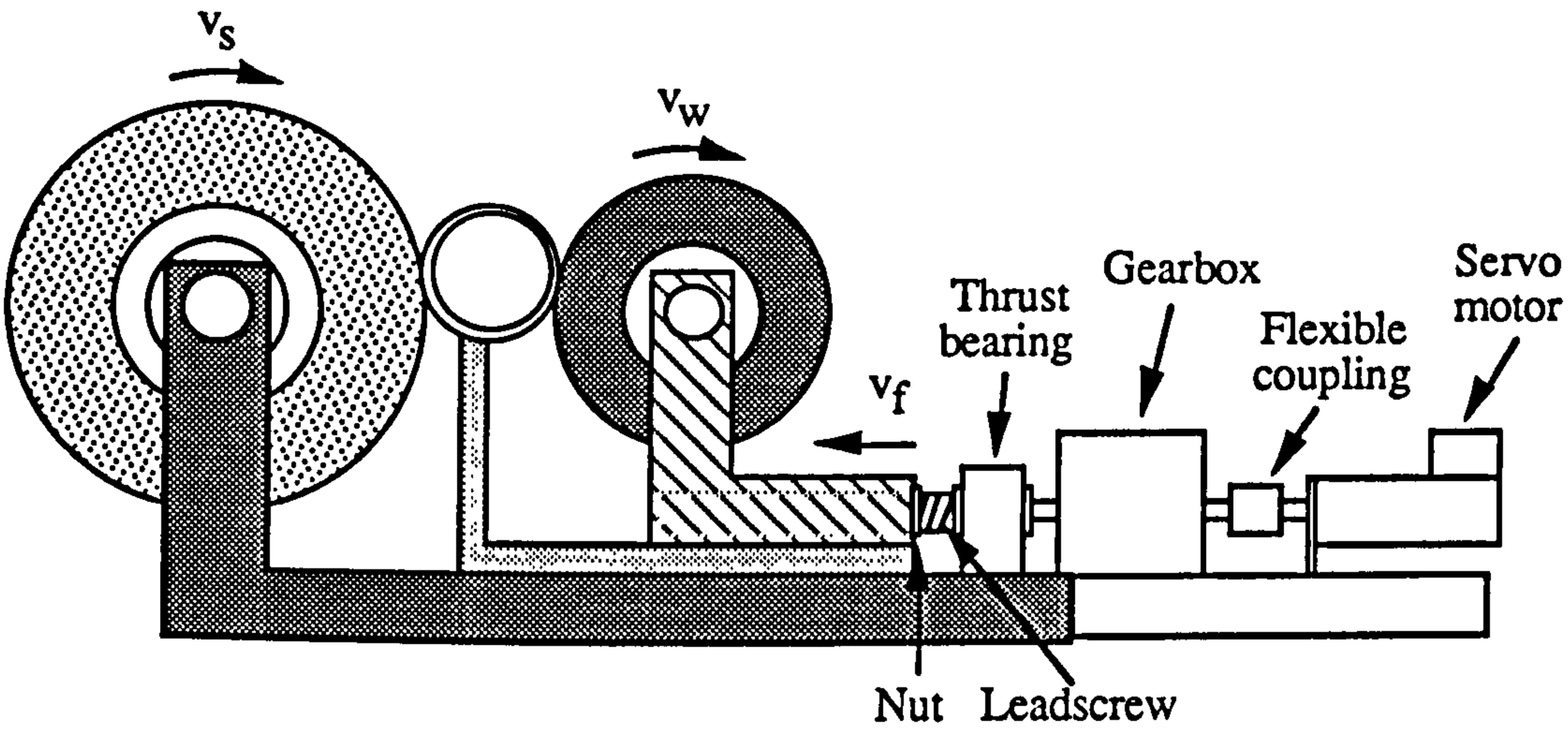



Figure 27. A schematic diagram of the automated infeed mechanism.

Originally the control wheelhead and slides were positioned by a handwheel. A pinion attached to the handwheel revolved a worm nut around a fixed lead screw. The wheelhead was positioned as the revolving worm nut traversed along the lead screw. The handwheel was removed and the worm nut clamped to the control wheelhead so that the wheelhead could be positioned by rotating the leadscrew. It was determined that a 3000 rev/min, 3.1 Nm servo motor connected to the 3.1 mm pitch lead screw through a 16.1:1 ratio gearbox would provide the required torque/speed characteristics

for positioning the infeed mechanism. A diagram of the mechanical arrangement is given in Figure 27. Consequently, a d.c. servo motor meeting these criteria was selected from the range produced by S.E.M. A 2.5 kW AM Power II servo amplifier was employed to provide infinitely variable speed control of the servo motor up to 3000 rev/min. The amplifier utilised d.c. tachometer feedback from the motor to achieve closed loop control of motor speed. The velocity reference to the servo amplifier was an analog voltage in the range ±10 V.

The infeed system design provided closed loop control of infeed rate through adjustment of a reference voltage. However, workpiece size accuracy ultimately depends on the accuracy of the infeed position. Therefore, it was necessary for the control system to position the infeed axis with the required accuracy. The C.N.C. system utilised an algorithm based on following error to provide closed loop control of the position and velocity of axes. The performance of the closed loop positioning system was dependent on the dynamic response of the infeed axis mechanism. Elements affecting the dynamic response included the motor performance, torsional compliance of the gearbox , control wheelhead inertia and friction. As the effect of these parameters was difficult to assess it was decided to analyse the dynamic response of the developed infeed mechanism.

With a 10 V d.c. reference signal to the servo amplifier, the maximum motor speed achieved whilst driving the wheelhead was recorded:

$$\text{maximum motor speed} = 84.2\pi \text{ rad/s} = 42.1 \text{ rev/s} \qquad 29.$$

The maximum lead screw speed was calculated by taking into account the 16.1:1 ratio gearbox:

$$\text{maximum lead screw speed} = 42.1 / 16.1 = 2.6 \text{ rev/s} \qquad 30.$$

Finally, the maximum infeed rate in mm/s was determined on the basis of a 3.1 mm lead screw pitch:

$$\text{maximum infeed rate} = 2.6 \times 3.1 = 8.1 \text{ mm/s} \qquad 31.$$

The response of the infeed mechanism to a 10 V step input was measured using an oscilloscope connected to the servo motor tacho-generator. Experimentation revealed that the average time taken to reach maximum speed was 0.28 s. As the maximum infeed rate was 8.1 mm/s it was possible to calculate the average infeed acceleration:

$$\text{acceleration} = 8.1 / 0.28 = 28.9 \text{ mm/s}^2 \qquad 32.$$

The response of infeed motor velocity '$\omega$' to a 10 V step speed demand to the servo amplifier was recorded using a storage oscilloscope connected to the servo motor tacho-generator. Mechanical systems comprise inertia, stiffness and damping elements which lead to potentially oscillatory second order responses. However, the characteristics of this particular system suggested a non-oscillatory response with a damping factor greater than unity. As infeed axis speed approached the steady state value exponentially, the infeed response was predominantly first order. The first order response of such a system is:

$$\omega(t) = \omega_{peak}\left(1 - e^{(-a\,t)}\right) \qquad 33.$$

where '$\omega(t)$' is the motor velocity in rad/s at time 't' and '$\omega_{peak}$' is the maximum motor velocity in rad/s. The time constant of the servo drive '$\tau$' is defined as the inverse of constant 'a'.

A value of time constant that provided a match between predicted and actual motor response was estimated to be $\frac{1}{16.41}$. The infeed drive system response was therefore approximated by the equation:

$$w(t) = 84.2\pi\left(1 - e^{(-16.41\,t)}\right) \qquad 34$$

Figure 28 illustrates the correlation between predicted infeed axis response from equation 34 and the actual values determined experimentally. It was apparent that a first order approximation was adequate for the purposes of this analysis.

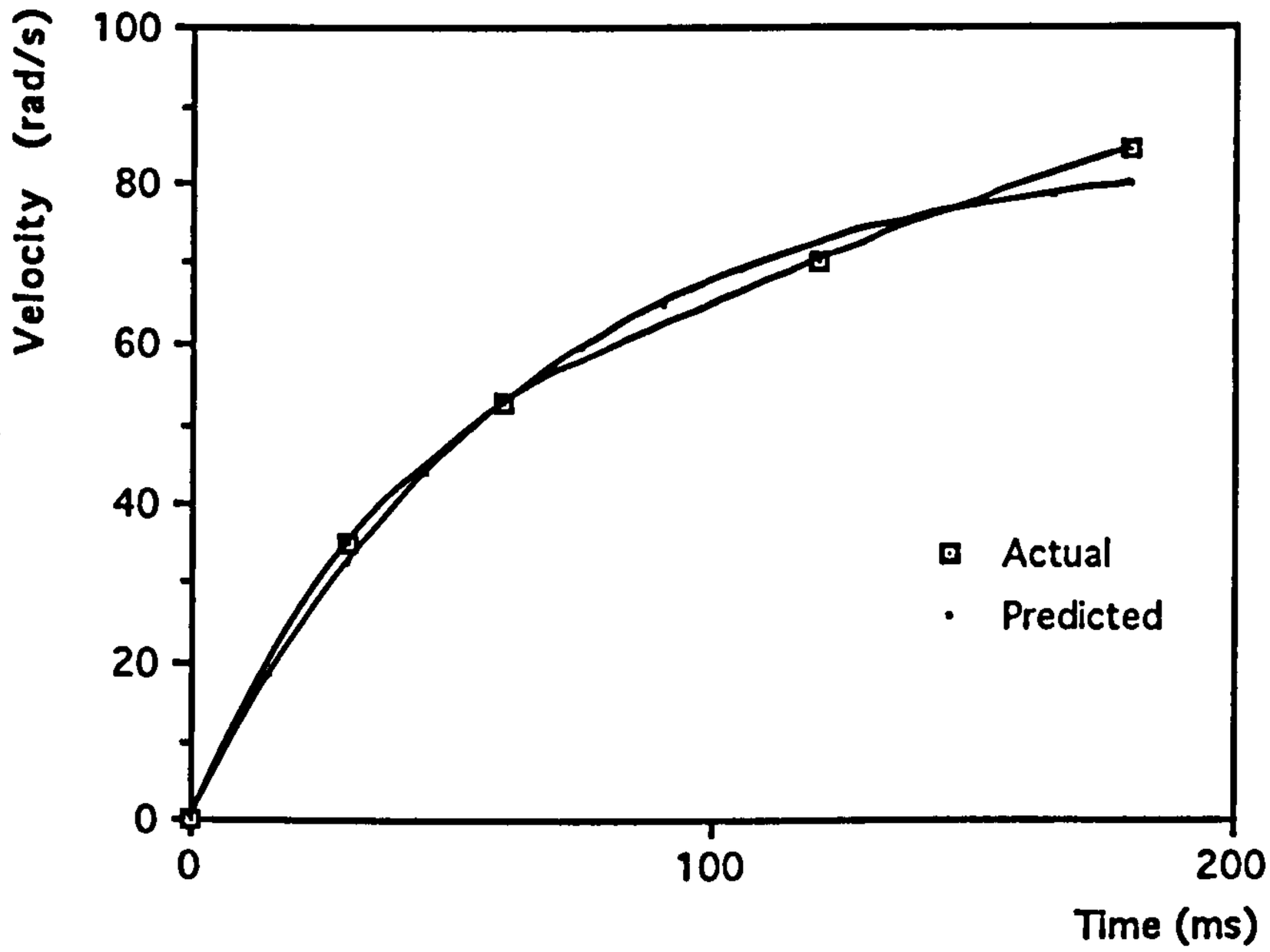



Figure 28. A graph showing actual and predicted motor response.

The Laplace transform of $\omega(t)$ from equation 34 is of the form:

$$L(w(t)) = w(s) = \frac{k\,a}{s\,(s + a)} \qquad 35.$$

Hence, by substituting the values determined experimentally:

$$w(s) = \frac{84.2\pi \times 16.41}{s\,(s + 16.41)} \qquad 36.$$

The Laplace transform of a step input of 10 V which corresponds to a demanded velocity of $84.2\pi$ rad/s is:

$$V(s) = \frac{84.2\pi}{s} \qquad 37.$$

The transfer function of a system is defined as the system output divided by its input with zero initial conditions. Hence the transfer function of the servo drive system was defined by the expression:

$$\frac{w(s)}{V(s)} = \frac{16.41}{s+16.41} \qquad 38.$$

The analysis of the performance of the infeed axis configuration was used to determine the parameters used by the control system to achieve a satisfactory closed loop positioning system.

The control system required positional feedback from a transducer connected to the infeed axis mechanism. Backlash between the servo motor and the control wheelhead of between 0.1 mm and 0.2 mm contributed a transportation lag or dead-time to the closed position loop system. It was considered impracticable to employ a rotary encoder mounted on the back of the infeed motor to remove the dead-time. With such an approach the positioning accuracy that may be achieved is determined by the total backlash in the system. In closed loop systems with a significant dead-time it is usually necessary to avoid instability by reducing the gain of the system.

Therefore, it was decided to use a linear scale for position measurement of the infeed axis. The scale was mounted with the body of the scale attached to the main machine casting. The reader head was connected to the machine tool upper slide so that the reader head moved whether the upper or lower slide was used for grinding. The linear scale chosen provided a resolution of 0.002 mm which allowed the control system to position with an accuracy of ±0.002 mm. The linear scale ensured that infeed position was not affected by backlash in the infeed mechanism.

The machine tool development provided the key elements required for control of the grinding machine by a computer control system. Principally, control of the infeed axis, control wheel and grinding wheel drives was achieved. However, there

remained the requirement for development of an in-process workpiece measurement system.

There are difficulties in measuring workpiece size on centreless grinding machines due to the lack of workpiece centre. In a centreless grinding machine the workpiece is supported by the workplate and control wheel. Consequently, the position of the workpiece centre changes as material is removed during a grinding cycle. The complex geometry of the control wheel, workplate and measuring device ensures that the relationship between workpiece size and probe reading is non-linear. However, two identical workpieces produce the same size reading from the device as the geometries are the same. Therefore, it is necessary to either datum the probing system with a target sized workpiece or to model the geometric non-linearity. Modelling of geometric non-linearities would have required knowledge of workplate height, control wheel diameter and the relative positions of upper and lower slides. This information was not readily available from the machine and the mathematics proved extensive.

As the project aims involved the production of large numbers of similar workpieces it was decided that in-process workpiece sizing should rely on relative size measurement. Contactless measuring systems were impracticable due to the large quantity of coolant in the grinding zone and it was necessary to employ a touch probe. GKN used analogue inductive probes on a number of manually operated centreless grinding machines for in-process measurement of workpiece size. A probe was positioned at each end of the workpiece to measure across the diameter. Workpiece taper was indicated by the difference in readings between the two probes. The operator used a trial and error approach to calibrating the probes. Over a number of workpieces the operator adjusted the probes so the 'at size' signal was triggered with a workpiece distorted and the infeed at a fixed position. This technique did not account for the errors in size readings that occurred due to variations in workpiece compliance and grinding forces.

The probes used by GKN were not suitable for use by the developed control system. Problems arising from the use of the inductive devices included achieving adequate resolution when interfacing the analog output to the C.N.C. and thermal drift of the size signal. The sizing devices were interfaced to a control system with a signal in the range 0-10 V corresponding to a reading of 0-0.25 mm. Assuming no electrical noise, a computer control system resolved size to ± 2.0 μm with an 8-bit analog to digital convertor. As the size signal clipped at 10 V there was a loss of information during the grinding cycle for workpieces with more than 0.25 mm of stock. For these reasons it was necessary to consider an alternative approach.

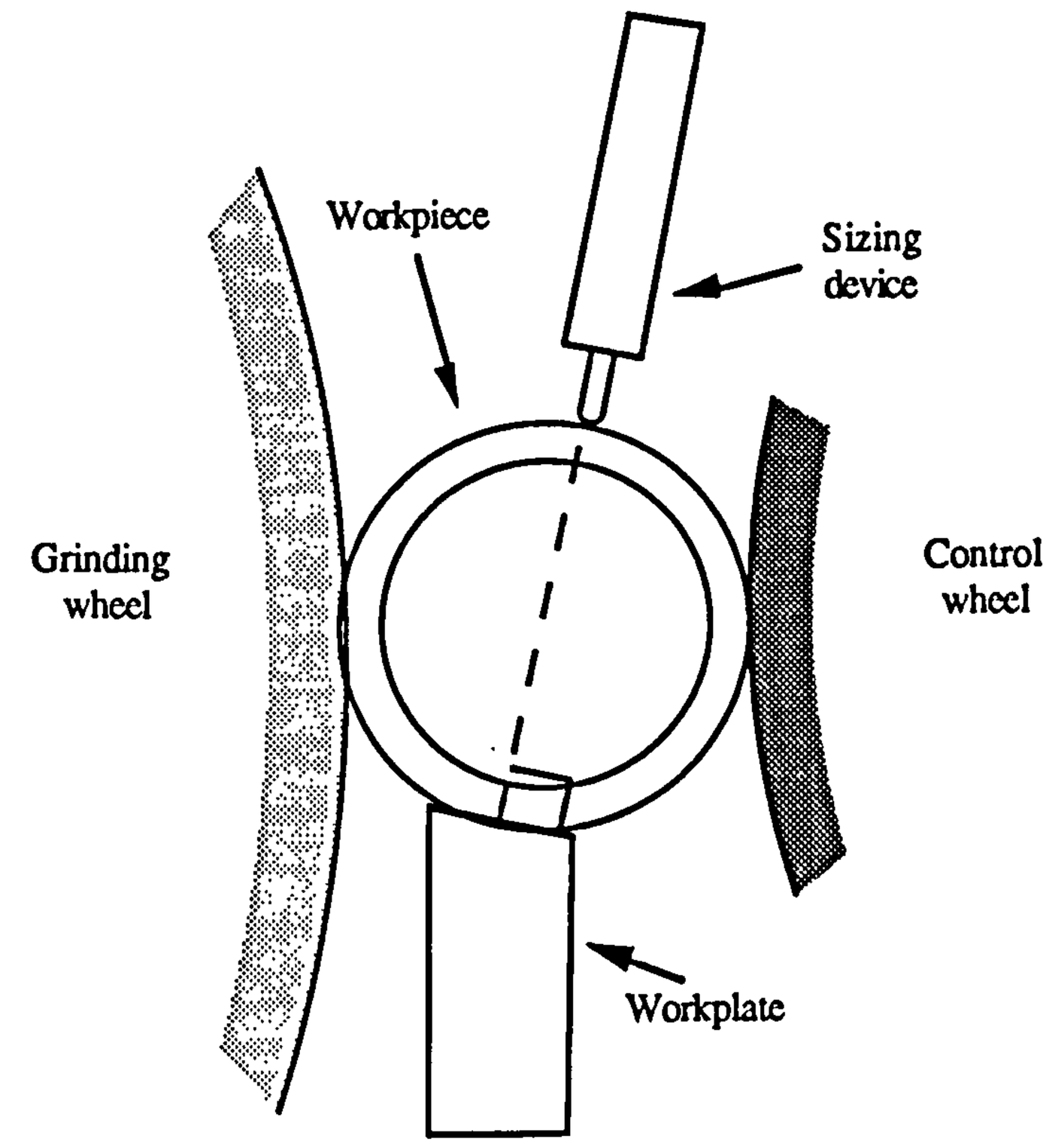



Figure 29. Arrangement of sizing devices for the centreless grinding process.

Digital probes were developed for the application in conjunction with Goodwin Electronic Measuring Systems Ltd. The sizing device comprised a pneumatically actuated probe with a hardened tip attached to a 50 mm x 1 μm resolution linear scale. The quadrature position signal from the linear scale interfaced directly to the C.N.C.

as if it was a conventional machine axis feedback device. A diagram of the probe arrangement is given in Figure 29.

The measurement technique devised was to first datum a sizing device on a ground workpiece of the required diameter. As workpiece size converged on the datum value during a grinding cycle the measurement errors due to geometric non-linearity tended to zero. In order to reduce geometric non-linearity careful machine setting was employed so that the sizing device measured across the centre-line of a finished workpiece. It was decided that two probes should be employed along the length of the workpiece to provide an indication of workpiece taper.

## 6.2 Electrical interface

A control cabinet was built to house the C.N.C. controller chassis, the axis drive electronics and various contactors, relays, transformers and power supplies. As there was a lot of air borne dust in the machining of cast iron, the cabinet was sealed and a heat exchanger installed for cooling the electronics. Connection of wires from the panel to the machine tool was through two large diameter 'anaconda' conduits with wires terminating on a terminal rail at the bottom of the cabinet. The design of the electrical control panel is included in the appendix.

The C.N.C. operator panel was suspended from a pendant at the front of the machine tool. Included in the pendant was an operator push button station that permitted control of a number of machine functions. The push button station included an electronic handwheel that allowed the operator to request infeed axis motion from the C.N.C. by simply rotating the ratcheted handwheel. This feature was useful during machine setting operations. The operator panel provided the following functions:

1. An over-travel over-ride keyswitch enabled an operator to restart the machine tool after the infeed axis had struck a hardware over-travel limit switch.

2. To prevent wear and tear of the C.N.C. cycle start push button, a secondary cycle start push button was fitted.
3. The electronic handwheel was enabled with the key switch provided.
4. A selector switch allowed the operator to choose between three coolant options. If 'AUTO' was selected the coolant was controlled by the workpiece part program via the M functions M08/M09 -coolant on and off respectively. If 'ON' was selected then coolant was on permanently. If 'OFF' was selected the coolant was turned off.

A number of relays were connected to the digital output points of the C.N.C. The relays allowed the C.N.C. control of a number of machine functions that included:

1. The grinding wheel drive amplifier enable.
2. The infeed axis drive amplifier enable.
3. Sizing device advance/retract.
4. Software generated emergency infeed retract.

Machine tool safety was of paramount importance. Grinding machines require guarding to reduce the possibility of operator injury in the event of a grinding wheel exploding at speed. Prevention of this type of problem was beyond the power of the control system. However, it was vital that the operator could stop the machine tool as safely as possible in an emergency. Also, diagnostics were required to inform the operator of any machine problems. The first level of electrical safety was provided by hardwired safety features:

1. 'Emergency stop' push buttons were provided so that power was removed from the machine tool in an emergency.
2. A 'jog retract' system was devised. The 'jog retract' push button created an emergency stop condition and caused the infeed axis to retract. This ensured that the machine tool was stopped safely with the workpiece away from the grinding wheel.

A number of signals were fed back to the C.N.C. to provide a second level of safety:

1. Machine electronics enabled
2. Machine tool not in emergency stop condition
3. Electrical control cabinet doors closed
4. Spindle amplifier electronics enabled
5. Infeed axis drive amplifier electronics enabled
6. Jog retract in progress
7. Heat exchanger over-temperature
8. Grinding wheel drive cooling fan contactor over-load
9. Machine tool hydraulics contactor over-load
10. Coolant pump contactor over-load
11. Swarf removal contactor over-load
12. Infeed axis hardware over-travel limit switches
13. Hydraulics at working pressure
14. Air supply at working pressure
15. Hydraulics at working temperature

## 6.3 Computer control system integration

The machine logic program executed by the 8600 C.N.C. process software was developed using the SIPROM (Programmable Interface System) utility. The purpose of machine logic was to interface the C.N.C. functions to control a particular machine tool and act as a communications interface between the machine tool and the numerical control. The machine logic program is referred to in this thesis as the SIPROM interface.

Machine logic was developed on the 8600 C.N.C. with the programming tools provided by SIPROM. The SIPROM language was a Boolean type language based on simple logical expressions. SIPROM allowed a series of elements to be combined according to certain rules to produce a statement. Combinations of such statements

formed the machine logic program. Statements used logical expressions to define and control both physical and logical input and output (I/O) signals.

The machine tool was interfaced to the C.N.C. so that machine power up was achieved in the following manner. Pressing the C.N.C. CONTROL ON push button applied power to the control. After the C.N.C. had powered up and performed its diagnostic tests pressing the CONTROL ON push button a second time latched a relay that supplied power to the machine electronics. The CONTROL ON push button then illuminated to indicate that power was supplied to the machine tool. At this point the machine interface was arranged to perform the following power up sequence:

1. Check emergency stop push buttons, jog retract push buttons and axis overtravel limit switches.
2. Check the overloads for the spindle motor blower, hydraulics pump, hydraulics cooler and swarf drag link.
3. Wait for hydraulic pump to achieve pressure for a maximum of 25 seconds.
4. Perform a control reset, initialise CSI software and force process display no. 1.
5. Load the main sequencing part-program ready for execution.

The interface was designed so that an emergency stop was generated if any of these checks failed and an appropriate message was displayed on line 5 of the process display message area. Once the machine had powered up the interface enabled closed loop position control of the infeed axis by the C.N.C.

The interface also decoded the following machine 'M' functions issued by the part program:

1. M03 Grinding wheel drive start.
2. M05 Grinding wheel drive stop.
3. M08 Coolant on.
4. M09 Coolant off.
5. M10 Infeed servo amplifier disable.

6. M11 Infeed servo amplifier enable.

7. M30 Program end, control reset.

Also the following machine elements were controlled by the interface:

1. Sizing device activation.

2. Sizing device calibration.

3. Electronic handwheel enable.

4. Message and I/O diagnostics.

The SIPROM compiler was a program resident on the 8600 that translated the SIPROM program into a language (i8086) that was scheduled by the operating system under process software. A debugger allowed a user to prove out a SIPROM program by monitoring the state of SIPROM variables whilst the machine logic was being executed. The SIPROM program developed for the centreless grinding machine is included in the appendix.

As described earlier, a following error system provided control of axis position and speed. The process software maintained a 'model' axis position for each controlled axis. After an axis move was requested, the C.N.C. updated the position of the model axis every 'tick' of 10 ms according to the programmed feed rate and the axis acceleration and deceleration ramps. An increase in model axis speed was generated by adding larger increments to the model position every tick. The model axis position and speed at a point in time was therefore the idealised position and speed of the real axis. As the C.N.C. started to move the model axis a positional following error was created between model and real axis positions. The analog reference voltage to the axis drive varied in proportion to the following error and therefore real axis motion followed that of the model axis. An increase in following error produced a corresponding increase in analog reference voltage. It can be shown by analysing the transfer function of the control loop that for constant model axis velocity, there is steady state value of following error and so a steady state reference to the axis drive.

The velocity error '$K_v$' defined in AXCFIL/MP0 related the analog reference voltage to the following error:

$$V_{analog} = \frac{K_v (x_{model} - x_{real})}{K_s} \qquad 39.$$

where '$X_{model}$' and '$X_{real}$' are the model and real axis positions respectively. The scaling factor '$K_s$' is a constant determined by the relationship between maximum axis speed '$v_{max}$' and reference voltage '$V_{max}$' defined in AXCFIL/MP0:

$$K_s = \frac{v_{max}}{V_{max}} \qquad 40.$$

For example, an axis that travelled at maximum speed of 10 m/min (166.7 mm/s) with an analog reference of 10 V required 1 V to traverse at 1 m/min. The value of velocity error controlled the gain of the control loop. With a velocity error of 16.7/s a following error of 1 mm generated the required 1 V to the axis drive. Decreasing the value of velocity error reduced the gain of the system and hence response became sluggish. A larger following error was therefore required to produce the 1 V required by the axis drive. When real axis position matched the model axis position there was no following error and hence no reference voltage.

In order for the C.N.C. to accurately control the position and speed of the modified infeed axis of the centreless grinding machine it was necessary to define a number of the parameters described above. Maximum axis speed '$v_{max}$' and reference voltage '$V_{max}$' were defined in AXCFIL/MP0. There were linearity problems with the analogue to digital convertors as the output voltage approached the full scale value. Therefore maximum axis speed was defined as the speed achieved with a reference of 8.5 V. The axis speed achieved with a 10 V signal was determined earlier and hence rapid traverse rate was calculated:

$$\text{Rapid traverse rate} = 8.1 \times \frac{8.5}{10.0} = 6.885 \text{ mm/s} \qquad 41.$$

If rapid traverse was set at the maximum available reference voltage it would not be possible to compensate for deceleration of the axis due to an increased load at the rapid traverse rate.

A value of velocity error that was too high caused the system to exhibit oscillatory behaviour whilst too low a value lead to a sluggish response. An oscilloscope was used to measure the response of the infeed motor subject to various velocity commands from the C.N.C. and a suitable value of velocity error was determined for the infeed axis. The value of velocity error '$K_v$' that provided a stable response was 10 /s. This value was lower than that expected on a new machine tool as there were significant lags introduced into the control loop by the backlash between the motor and the headstock.

# Chapter 7. Evaluation and results

Trials were performed using the intelligent control system integrated within the OSAI A-B 8600 C.N.C. and interfaced to the Cincinnati No. 3 centreless grinding machine. The purpose of the trials was to evaluate the success of the control system structure when applied to the centreless grinding process. In particular it was necessary to test the effectiveness of the proposed control strategies in producing the highly compliant cylinder liners produced by GKN. In order to evaluate the performance of the control system independently of the C.N.C. based real-time analysis, a BBC computer based data logging system was used for external in-process measurement of process parameters. The data logger was interfaced to the machine tool and provided measurement of workpiece diameter and grinding wheel power consumption. Workpiece quality was evaluated through measurement of roundness, surface finish and diameter using suitable equipment from the metrology laboratory of the Liverpool John Moores University.

The developed control system provided a means of optimising grinding conditions for a range of workpieces, materials and grinding wheels. As part of the project aims and objectives it was necessary to consider optimising the process as opposed to optimising grinding conditions.

## 7.1 Thermal effects on in-process measurement of workpiece size

Initial grinding experiments revealed the effects of temperature on the in-process measurement of workpiece size. As grinding imparts energy to a workpiece there is a rise in temperature and consequently expansion of the workpiece occurs. Bryan [63] suggested that such thermal effects were the largest source of dimensional errors in machining processes. The aim of the experiment was to evaluate the amount of workpiece expansion experienced and to determine an appropriate technique for workpiece diameter measurement. Therefore, it was necessary to determine the time

taken for thermal expansion to reduce to a point at which satisfactory diameter measurements could be obtained. Also, the effects of coolant on the workpiece measurement cycle were considered.

A thin walled cast-iron cylinder liner of the type produced by GKN was heated in boiling water to approximately 100°C. The workpiece was placed in the grinding machine and the sizing device activated. Readings of the change in workpiece diameter over time were recorded by the external data logging system connected to the sizing device. The experiment was performed both with machine flood coolant off and on.

Data logging of workpiece size produced the graphs shown in Figure 30 without coolant and Figure 31 with coolant.

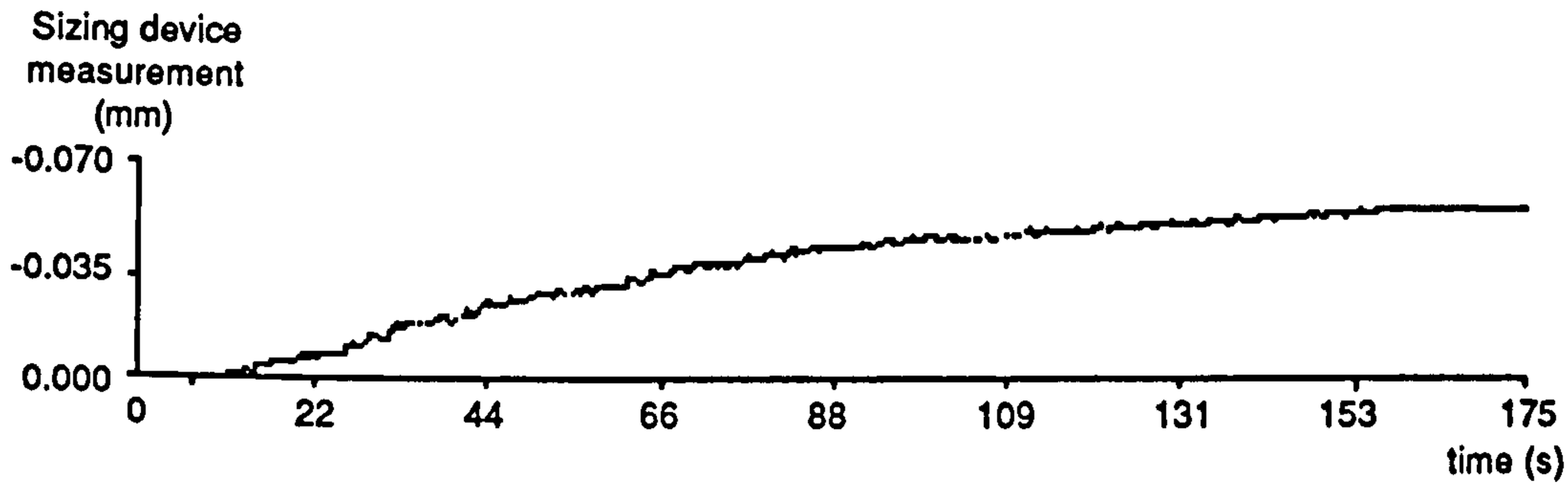



Figure 30. Reduction in workpiece size without coolant

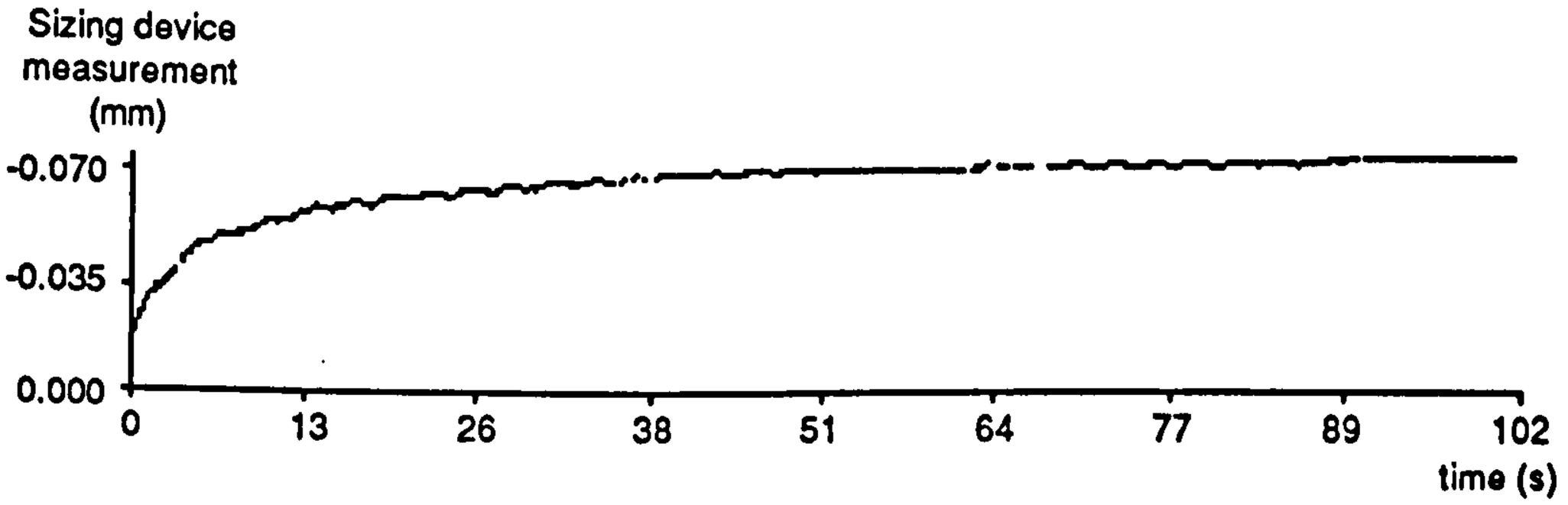



Figure 31. Reduction in workpiece size with coolant.

The results indicate that the thin walled cylinder liners produced by GKN expanded by up to 50 μm with an increase in temperature from ambient to 100°C. The time constant of the reduction in size was approximately 40 seconds without coolant and 6 seconds with coolant. Therefore, machine coolant significantly increased the rate at which size of the cylinder liner reduced. Peters [13] indicated that an approximate value of steady state workpiece temperature rise 'θ' could be derived from the following equation:

$$\theta = \frac{K P}{\alpha \Omega} \qquad 42.$$

where 'K' is the percentage grinding energy entering the workpiece, 'P' steady state grinding power, 'α' the average heat transfer coefficient and 'Ω' the cooled workpiece area. Using approximate values of these parameters from Peters provides an estimated temperature rise for a cylinder liner:

$$\theta = \frac{0.1 \times 38000}{3500 \times 0.073} = 14.9°C \qquad 43.$$

Thermal expansion is proportional to temperature and a temperature rise of 15°C would therefore result in workpiece expansion in the order of 7.5 μm.

The intelligent grinding control system framework included a resource for in-process measurement of workpiece diameter. With the measuring strategies employed it was possible to measure diameter either during a grinding cycle or by interrupting a cycle and retracting the infeed to a gauging position. The benefit of interrupting the cycle to measure the workpiece was to avoid the deflections experienced with compliant workpieces. It was not possible to eliminate the effects of thermal expansion from the sizing device readings when measuring during a grinding cycle. However, when using the interrupted cycle strategy for workpiece measurement it was ensured that machine coolant remained on and that a suitable delay was allowed for the workpiece temperature to stabilise. It was decided that a delay of 5 s would be suitable

for the purposes of this work. Using this technique thermal effects on workpiece measurement accuracy and cycle times were minimised.

## 7.2 Conventional grinding of rigid workpieces

The developed control system software was initially tested by performing a conventional plunge grinding cycle of the type shown in Figure 16 on rigid workpieces. The workpiece descriptions are given in Table 10. This trial was important as the effects of workpiece compliance on results are minimised with rigid workpieces and the dimensional accuracies achieved indicate the size holding capability of the machine tool. Also, measurement of grinding wheel power consumption allowed machine compliance to be estimated. Machine performance was evaluated through measurement by the data logging system of workpiece diameter and of grinding wheel power consumption. Post-process measurement of workpiece size was performed with an S.I.P. three axis co-ordinate measuring machine. In-process measurement, process models, adaptive strategies and learning strategies were not implemented by the control system at this stage.

| Parameter | Rigid workpiece |
|---|---|
| Initial diameter | 50.800 mm |
| Target diameter | 50.000 mm |
| Internal diameter | None |
| Width | 75.000 mm |
| Material | EN9 |

Table 10. Workpiece description.

The controlled grinding parameters entered into the data base by the operator were values of grinding wheel speed, work speed, infeed rate and dwell time. The grinding parameters are given in Table 11.

| Parameter | Rigid workpiece |
|---|---|
| Wheel speed '$v_s$' | 45 m/s |
| Wheel diameter '$d_s$' | 609 mm |
| Work speed '$v_w$' | 0.45 m/s |
| Infeed rate '$v_f$' | 0.1 mm/s |
| Dwell time '$t_d$' | 10 s |
| Average specific removal rate 'Z" | 7.92 $mm^2/s$ |

Table 11. Controlled grinding parameters.

Effects of grinding wheel wear on the results were minimised by grinding small batches of workpieces between grinding wheel dressing. After allowing two hours for the machine to reach a steady operating temperature grinding trials were performed in batches of 10 on the rigid workpieces.

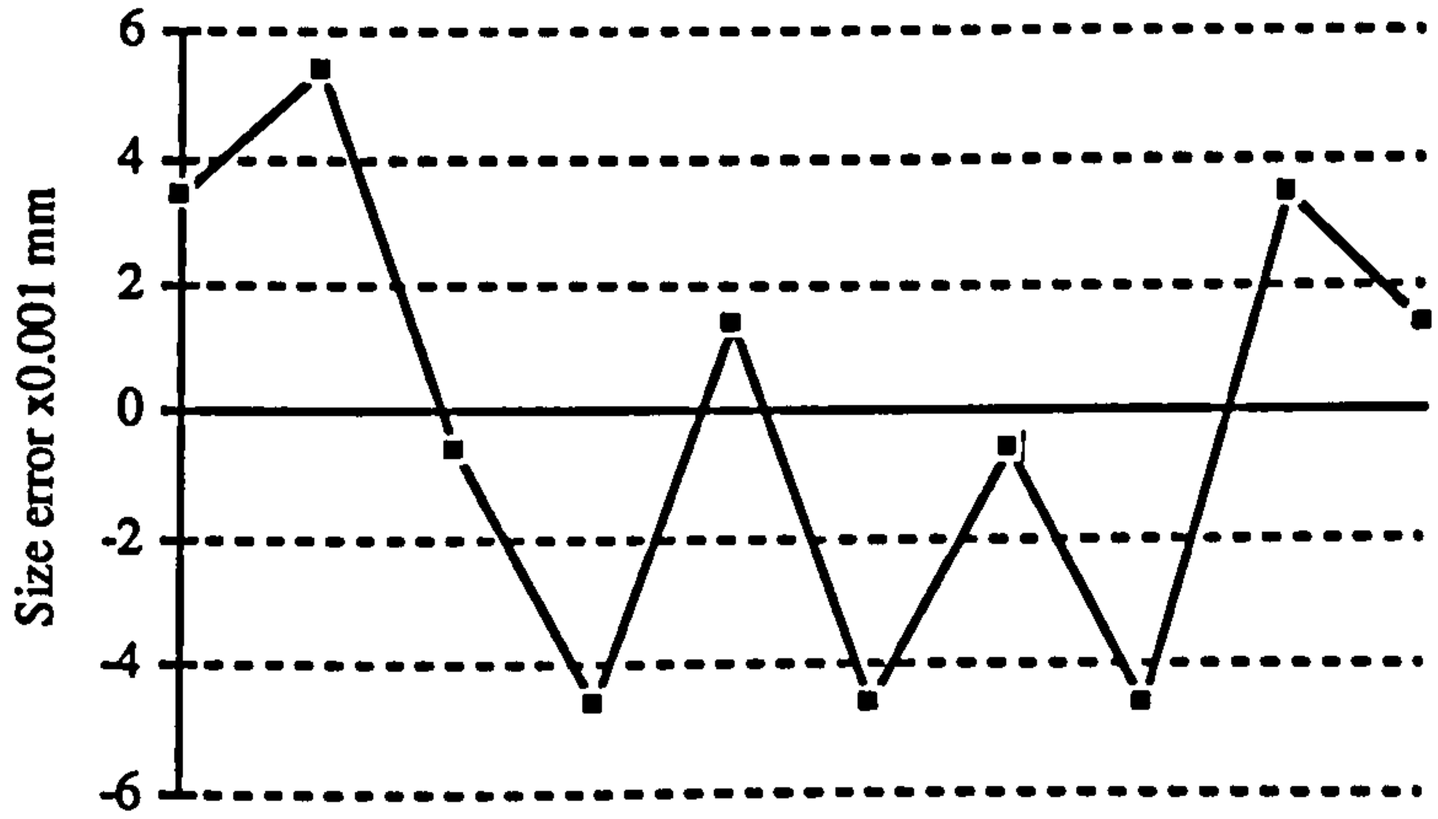



Figure 32. A graph of deviation of workpiece size from the average value.

A sample graph of deviation of workpiece size from the average batch diameter obtained from post-process measurements is given in Figure 32. Table 12 summarises the measurements of workpiece diameter after grinding. Assuming an exponential power decay during the spark out period, the approximate value of system time constant shown in Table 13 was determined.

| Parameter | Rigid workpiece |
|---|---|
| Average diameter | 50.003 mm |
| Maximum diameter | 50.008 mm |
| Minimum diameter | 49.998 mm |
| Standard deviation | 3.666 μm |

Table 12. Summary of grinding trial results.

| Parameter | Rigid workpiece |
|---|---|
| Approx. power | 26 kW |
| Approx. spark out | 3.5 seconds |
| Approx. time constant | 1.4 seconds |

Table 13. Time constant measurements.

It was noted that the grinding wheel speed temporarily fluctuated by up to 10% of the steady state value when load was suddenly applied. An effect of the fluctuation in grinding wheel speed was to cause a corresponding instability in power measurement. However, after a period of around 2 seconds, the grinding wheel drive regained stable operating speed.

Grinding trials on rigid workpieces indicated that the Cincinnati No. 3 centreless grinding machine was capable of grinding diameters with a standard deviation of 3.67 μm. Assuming a normal distribution of results and a rejection rate of 0.5% the machine size holding capability on workpiece diameter was ±10.31 μm. The computer control system relied on a quadrature linear scale mounted on the headstock to provide 2 μm resolution position feedback. Therefore, as positional accuracy was maintained by the control system to within ±2 μm, the remaining error of ±8.31 μm was attributed to the machine tool and grinding wheel wear. Causes of workpiece size error in machine tools include worn spindle bearings [22], incorrectly adjusted or worn slideways [36] and the effects of temperature on the machine geometry [13,63].

The measured value of system time constant '$\tau$' was approximately 1.4 seconds for the rigid workpieces. The machine stiffness parameter '$K_m$' was derived from equation 17:

$$K_m \cong \frac{2P}{v_s v_f \tau} = \frac{2 \times 26000}{45 \times 100 \times 1.4} = 8.25 \text{ N/}\mu\text{m} \qquad 44.$$

where 'P' was the grinding power before spark out. From the approximate value of system stiffness it was shown that machine deflections in the region of 140 µm were experienced during the grinding of rigid workpieces. The large degree of deflection illustrated the reason for spark out times during conventional grinding cycles. However, derivation of machine stiffness from grinding wheel power consumption is approximate. This is because grinding force ratio varies with grinding conditions over time due to grinding wheel wear.

## 7.3 Conventional grinding of compliant workpieces

The effects of workpiece compliance on quality and production rates were assessed by application of the conventional plunge grinding cycle of the type shown in Figure 16 to the production of thin walled cylinder liners. The workpiece descriptions are given in Table 14. The cast-iron cylinder liners provided by GKN for grinding trials were representative of those commonly used in modern internal combustion engines and a diagram of a cylinder liner is given in Figure 6. Measurement of grinding wheel power consumption allowed workpiece compliance to be estimated. System performance was evaluated through measurement by the data logging system of workpiece diameter and of grinding wheel power consumption. Post-process measurement of workpiece size was performed with an S.I.P. three axis co-ordinate measuring machine. In-process measurement, process models, adaptive strategies and learning strategies were not implemented by the control system at this stage.

| Parameter | Compliant workpiece |
|---|---|
| Initial diameter | 103.600 ± 0.125 mm |
| Target diameter | 103.000 mm |
| Internal diameter | 98.000 mm |
| Width | 225.000 mm |
| Material | Cast-iron |

Table 14. Workpiece description.

The controlled grinding parameters entered into the data base by the operator were values of grinding wheel speed, work speed, infeed rate and dwell time. The grinding parameters are given in Table 15. Although controlled grinding parameters were chosen that provided satisfactory grinding conditions for similar workpieces, due to the highly compliant nature of the cylinder liners an extended dwell time of 120 seconds was programmed to allow spark out.

| Parameter | Compliant workpiece |
|---|---|
| Wheel speed '$v_s$' | 45 m/s |
| Wheel diameter '$d_s$' | 609 mm |
| Work speed '$v_w$' | 0.45 m/s |
| Infeed rate '$v_f$' | 0.1 mm/s |
| Dwell time '$t_d$' | 120 s |
| Average specific removal rate 'Z" | 16.22 $mm^2/s$ |

Table 15. Controlled grinding parameters.

Effects of grinding wheel wear on the results were minimised by grinding small batches of workpieces between grinding wheel dressing. After allowing two hours for the machine to reach a steady operating temperature grinding trials were performed in batches of 10 on the rigid workpieces.

A sample graph of deviation of workpiece size from the average batch diameter obtained from post-process measurements is given in Figure 33. Table 16 summarises the measurements of workpiece diameter after grinding. Graphs of the type shown in Figure 34 were produced from measurement of grinding wheel power consumption and workpiece size with external data logging equipment. Figure 34 describes the effect of the dwell time on machine power and hence grinding forces. Also shown is the distortion of the workpiece caused by grinding forces during the grinding cycle. The distortion prevents accurate in-process measurement of workpiece size. Cylinder liner compliance was so extreme that manual machine tool operators used an 'overshoot' target infeed position to squeeze and distort the liner in an attempt to increase cutting forces and minimise dwell times. Assuming an exponential power decay during the spark out period, the approximate value of system time constant shown in Table 17 was determined.

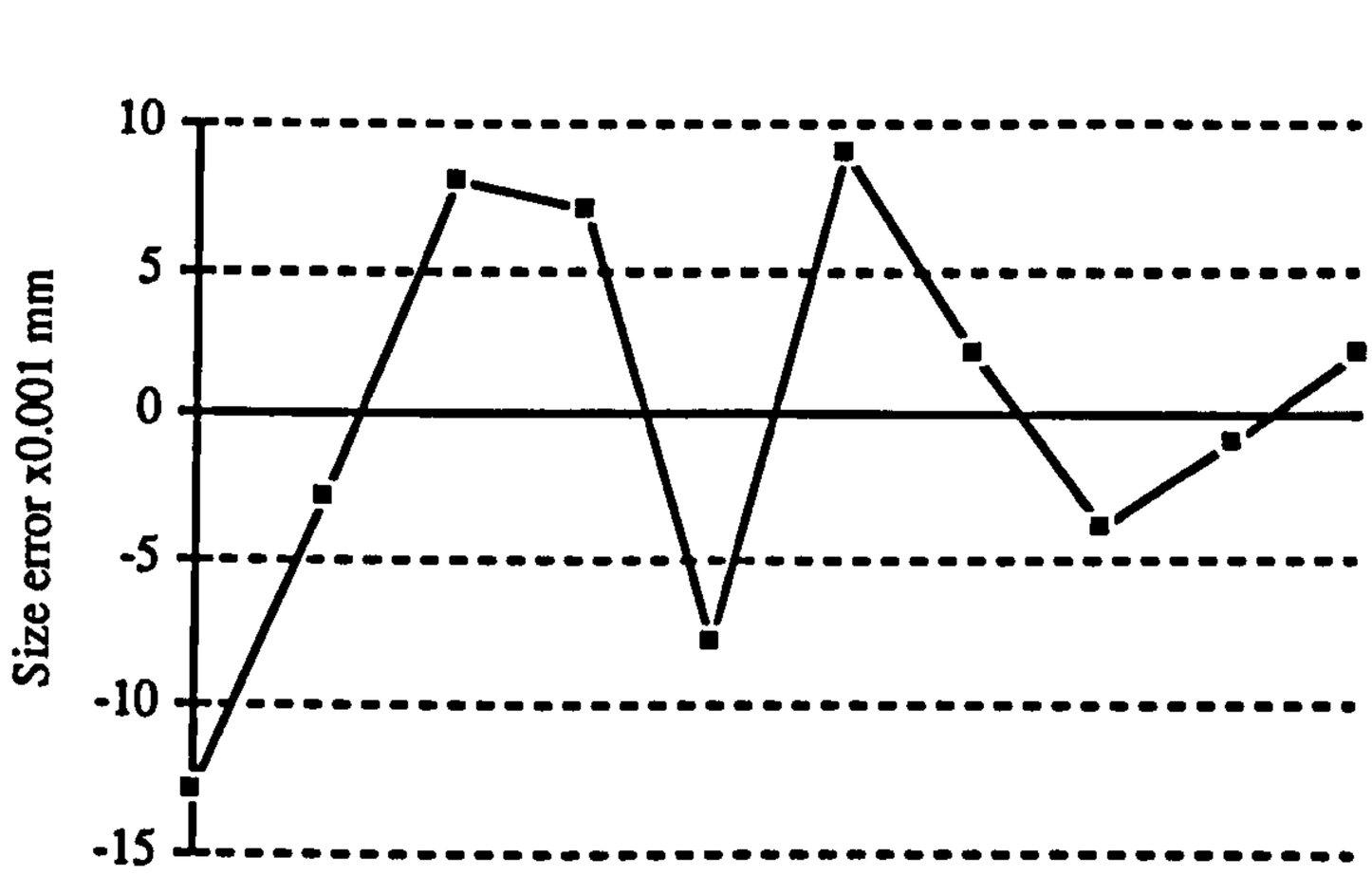



Figure 33. A graph of deviation of workpiece size from the average value.

| Parameter | Compliant workpiece |
|---|---|
| Average diameter | 103.001 mm |
| Maximum diameter | 103.010 mm |
| Minimum diameter | 102.988 mm |
| Standard deviation | 6.782 μm |

Table 16. Summary of grinding trial results.

| Parameter | Compliant workpiece |
|---|---|
| Approx. power | 25 kW |
| Approx. spark out | <120 seconds |
| Approx. time constant | >20 seconds |

Table 17. Time constant measurements.

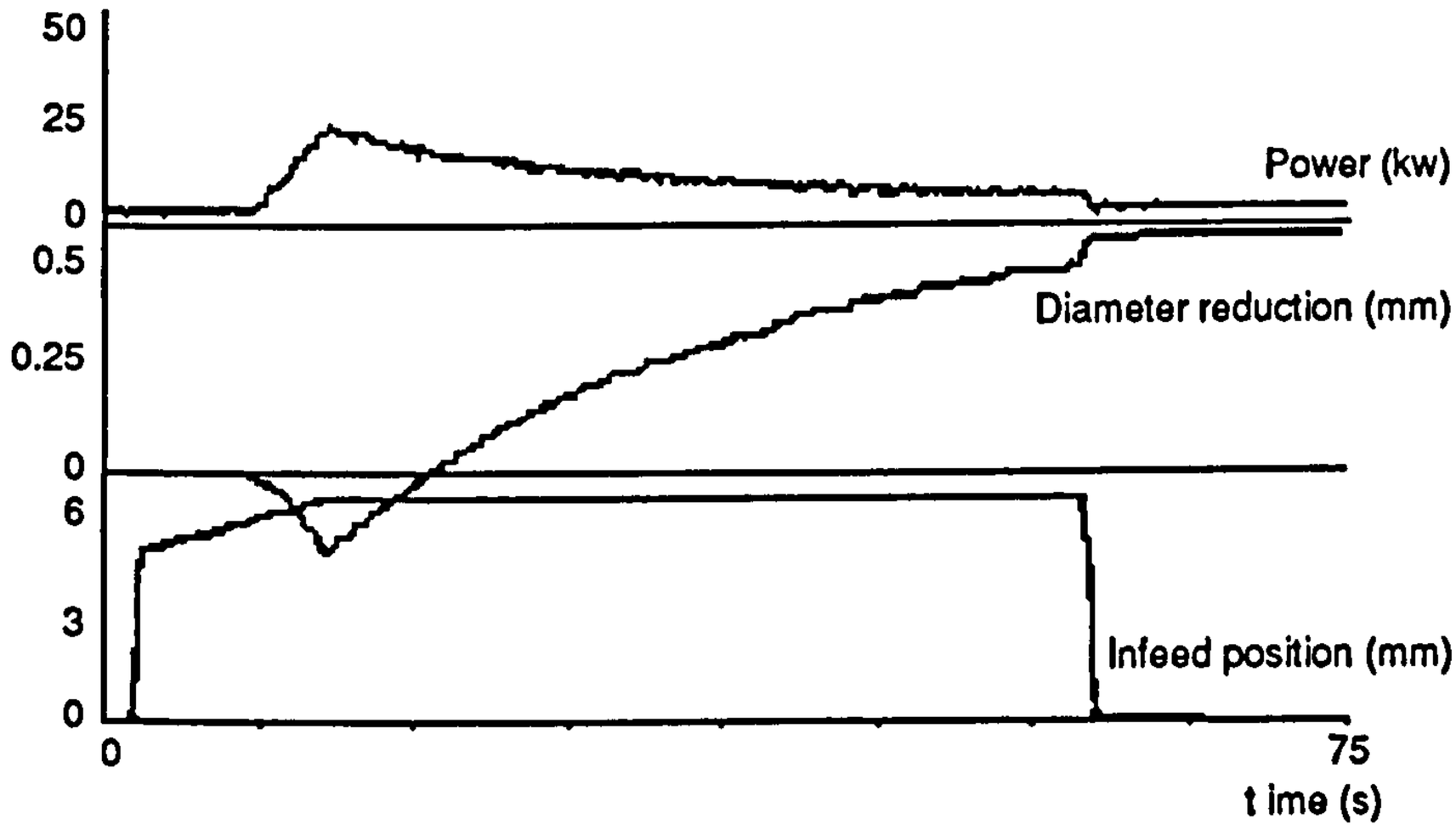



Figure 34. Data log of a 'feed to size' cycle for a compliant workpiece.

The grinding trials performed on cylinder liners indicated that, with a dwell time of 120 seconds, the Cincinnati No. 3 centreless grinding machine was capable of grinding the diameter of a small batch of compliant workpieces with a standard deviation of 6.78 μm. Assuming a normal distribution of results and a rejection rate of 0.5% the machine size holding capability for compliant workpiece size was ±19.05

μm. As the basic process accuracy was ±10.31 μm, variations in workpiece compliance and initial diameter accounted for a finished diameter error of approximately ±8.74 μm. However, after 120 seconds of dwell period the value of workpiece surface texture was too low and the cycle time was excessive.

Spark out of the cylinder liners was completed within the 120 second dwell period and a system time constant '$\tau$' of approximately 20 seconds was derived from the power decay curve. Steady state grinding conditions were not achieved during the grinding time '$t_g$' of 6 seconds and depth of cut lagged infeed axis position. After feeding for 0.6 mm at an infeed rate '$v_f$' of 0.1 mm/s the true infeed rate '$v_{fi}$' relative to the workpiece was derived from the compliance model [28]:

$$v_{fi} = v_f\left(1 - e^{\frac{-t_g}{\tau}}\right) = 0.1\left(1 - e^{\frac{-6}{20}}\right) = 0.026 \text{ mm/s} \qquad 45.$$

The total system stiffness parameter '$K_t$' for workpiece and machine was therefore:

$$K_t \cong \frac{2P}{v_s v_{fi} \tau} = \frac{2 \times 25000}{45 \times 20 \times 26} = 2.15 \text{ N/}\mu\text{m} \qquad 46.$$

where 'P' was the grinding power before spark out. From Hahn [1] total system stiffness is related to machine stiffness and workpiece stiffness '$K_w$' in the following manner:

$$\frac{1}{K_t} = \frac{1}{K_m} + \frac{1}{K_w} \qquad 47.$$

As machine stiffness was found to be approximately 8.25 N/μm from the previous trials, the workpiece stiffness was calculated to be 2.91 N/μm. From the approximate value of system stiffness it was calculated that workpiece deflections of up to 380 μm were expected during the grinding of compliant cylinder liners. The magnitude of predicted workpiece deflections was verified by in-process measurements of workpiece size. The graph of workpiece size in Figure 34 illustrates the workpiece distorting and appearing to 'grow' by approximately 250 μm as the infeed axis advances.

Variations in compliance and initial size between individual workpieces compounded problems in achieving accurate workpiece size. Figure 35 shows variations in cylinder liner wall thickness that contributed to the variable compliance of workpieces. From this data it was determined that wall thickness varied by up to 9% of the average value.

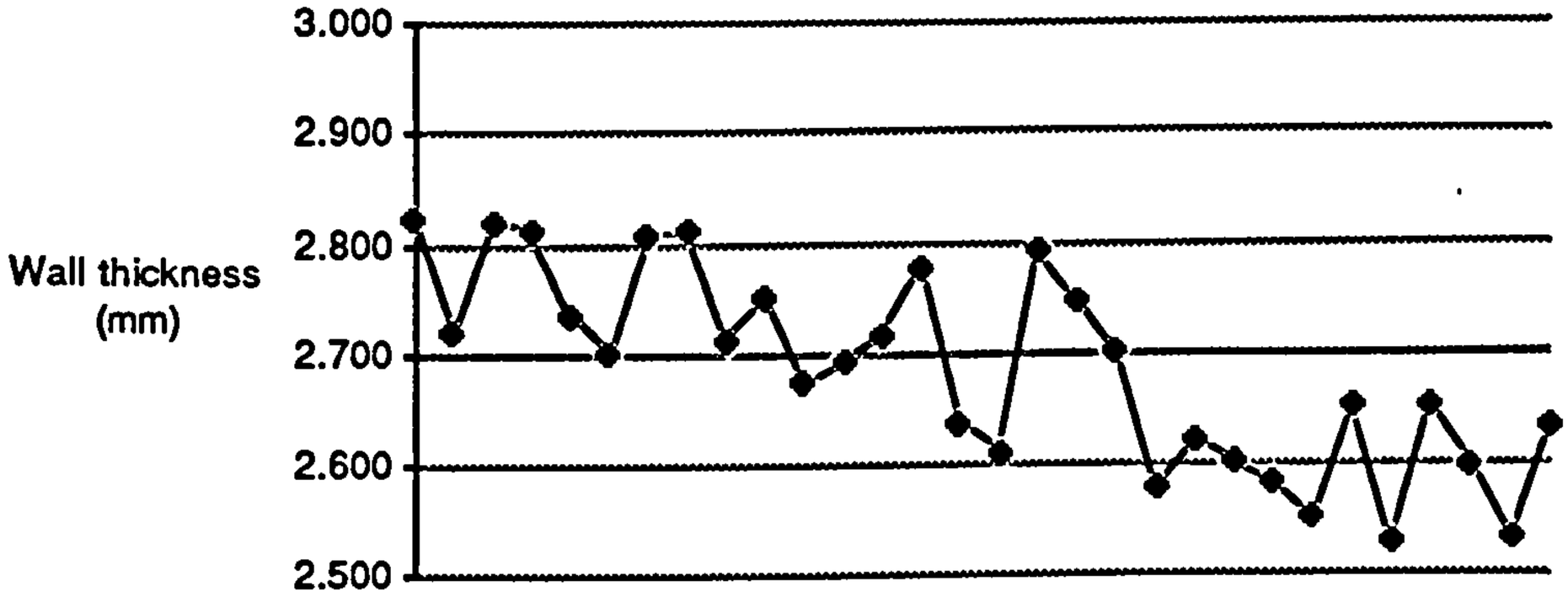



Figure 35. Variations in cylinder liner wall thickness

The large degree of workpiece deflection and the correspondingly low level of grinding force illustrated the need for lengthy spark out times when grinding thin walled cylinder liners with conventional grinding cycles. In order to reduce cycle times by reducing the dwell period required that it was necessary to consider methods of increasing grinding forces.

## 7.4 Pecking cycle evaluation

Early trials indicated problems arising from use of a conventional cycle when grinding compliant workpieces. Principally, large system deflections led to low grinding forces and hence low removal rates. It was therefore decided to test the pecking cycle illustrated in Figure 24 as a means of increasing grinding forces through use of a feed cycle employing an overshoot.

The aim of the experiment was to test the effectiveness of the intelligent control system in implementing a three peck cycle and of the success or otherwise of the pecking cycle itself. System performance was evaluated through measurement by the data logging system of workpiece diameter and of grinding wheel power consumption. Post-process measurement of workpiece size was performed with a 'match size' gauging system and roundness with a Talyrond. A workpiece description of the cylinder liners used in the trials is given in Table 18.

| Parameter | Compliant workpiece |
|---|---|
| Initial diameter | 96.168 ± 0.125 mm |
| Target diameter for peck 3 | 95.660 mm |
| Internal diameter | 91.000 mm |
| Width | 163.000 mm |
| Material | Cast-iron |

Table 18. Workpiece description.

The size of the first peck was chosen to be sufficient to provide rounding of the workpiece and thus an accurate measurement of workpiece size prior to the second peck. This was necessary as workpieces were initially out of round by over 0.050 mm and measurement of size was therefore unreliable. The value of overshoot used in the rounding cycle was calculated from the initial measurement of workpiece size, the required material removal and the stored value of time constant for that peck. Variations between the time constant used to predict an overshoot position and the time constant achieved whilst grinding, resulted in workpiece diameter errors that were detected by the sizing devices. Following the roughing cycle the workpiece diameter achieved was measured and the compliance model calculated a new value of time constant for the rounding peck. The new value of time constant was filtered and stored in the data base. Thus, the pecking cycle provided the developed control system with

the ability to compensate for varying workpiece compliance whilst also accounting for changes in system time constant that occurred due to changes in grinding conditions.

The pecking cycle was defined by entering a stock allowance for each peck of the cycle. The pecking allowances that were used for the trials are given in Table 19.

| | |
|---|---|
| Peck allowance 1 | 0.150 mm |
| Peck allowance 2 | 0.050 mm |
| Peck allowance 3 | 0.000 mm |

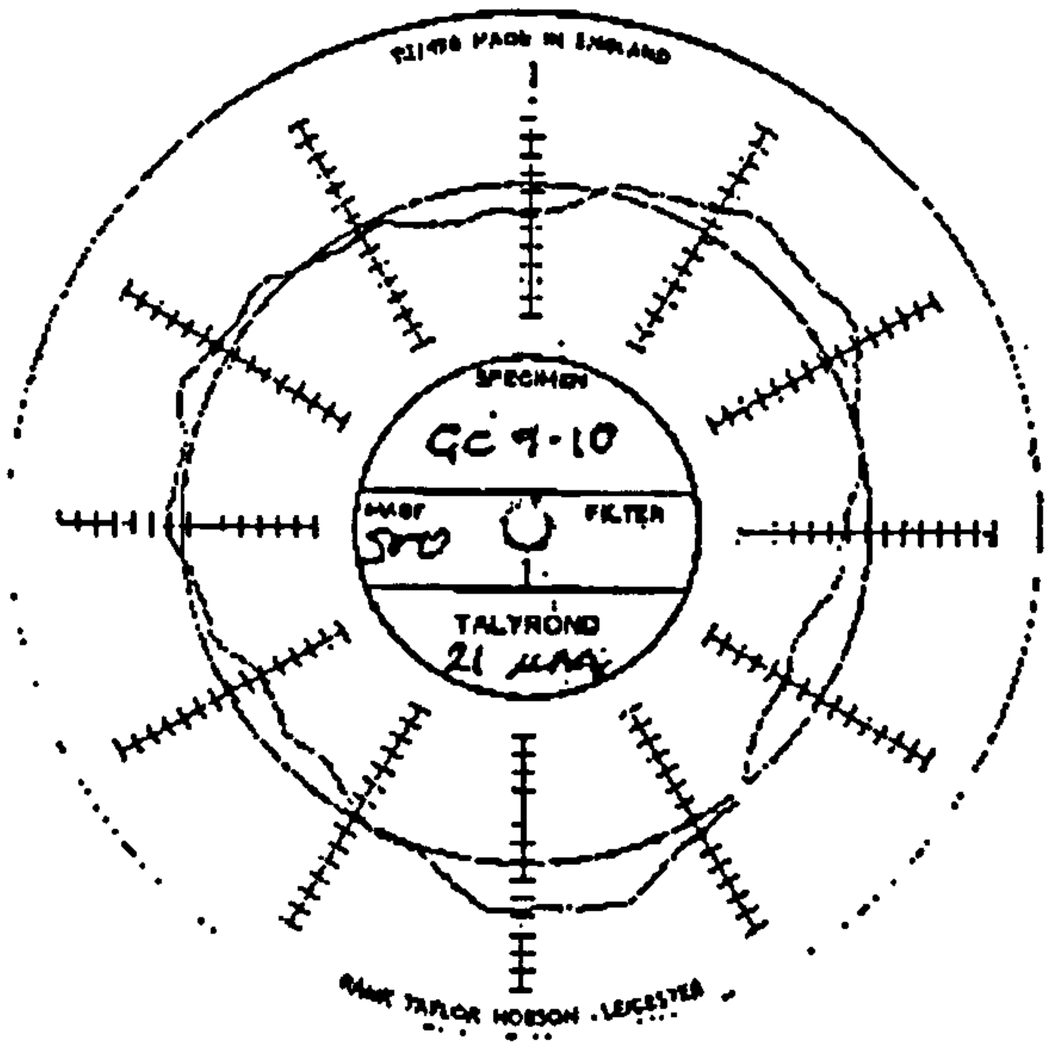
Table 19. Pecking cycle allowances.

The grinding trials were performed with the pecking cycle using values of grinding parameters for the process that were considered suitable for cast-iron [33]. The controlled grinding parameters that were chosen are given in Table 20.

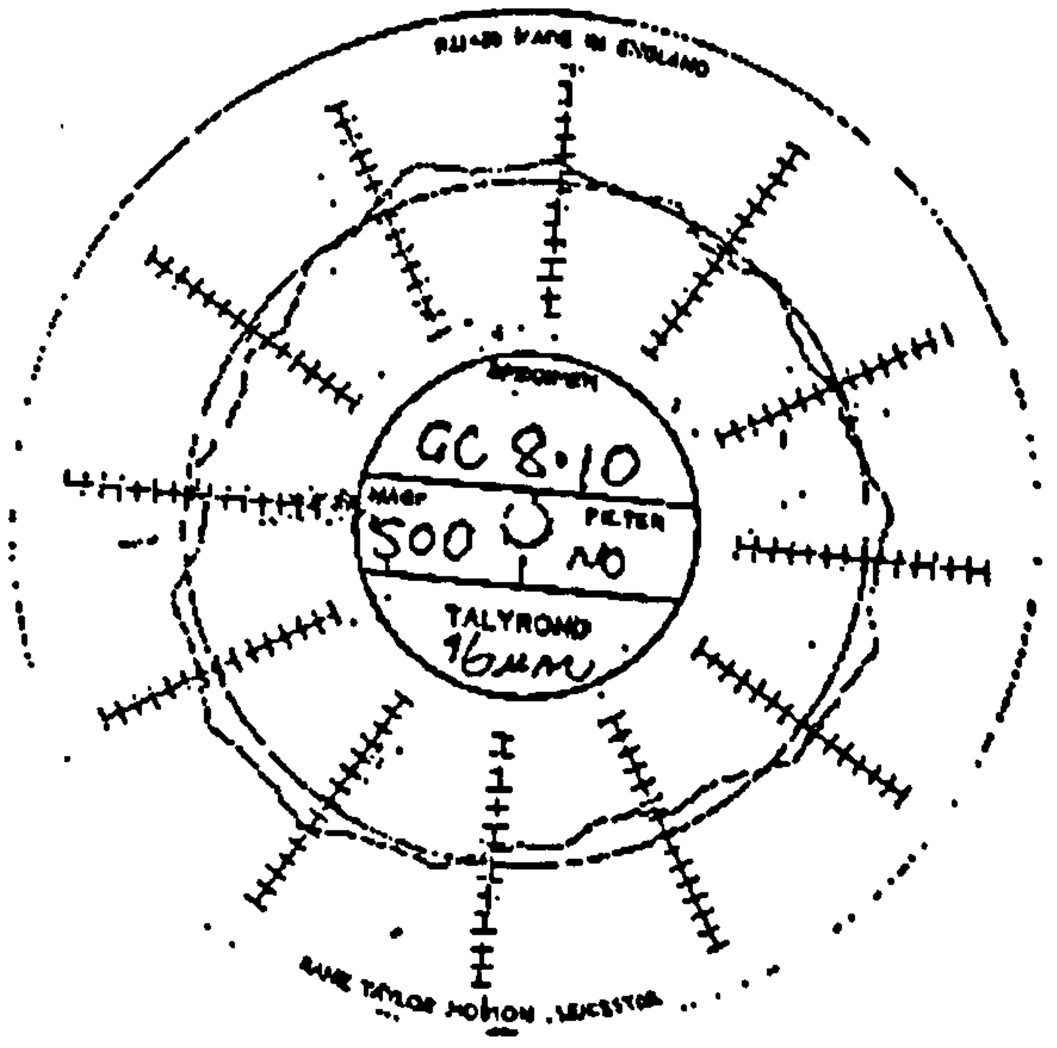
| Parameter | Compliant workpiece |
|---|---|
| Wheel speed '$v_s$' | 45 m/s |
| Wheel diameter '$d_s$' | 609 mm |
| Work speed '$v_w$' | 0.45 m/s |
| Infeed rate '$v_f$' | 0.1 mm/s |
| Average specific removal rate 'Z" | 7.92 $mm^2/s$ |

Table 20. Controlled grinding parameters.

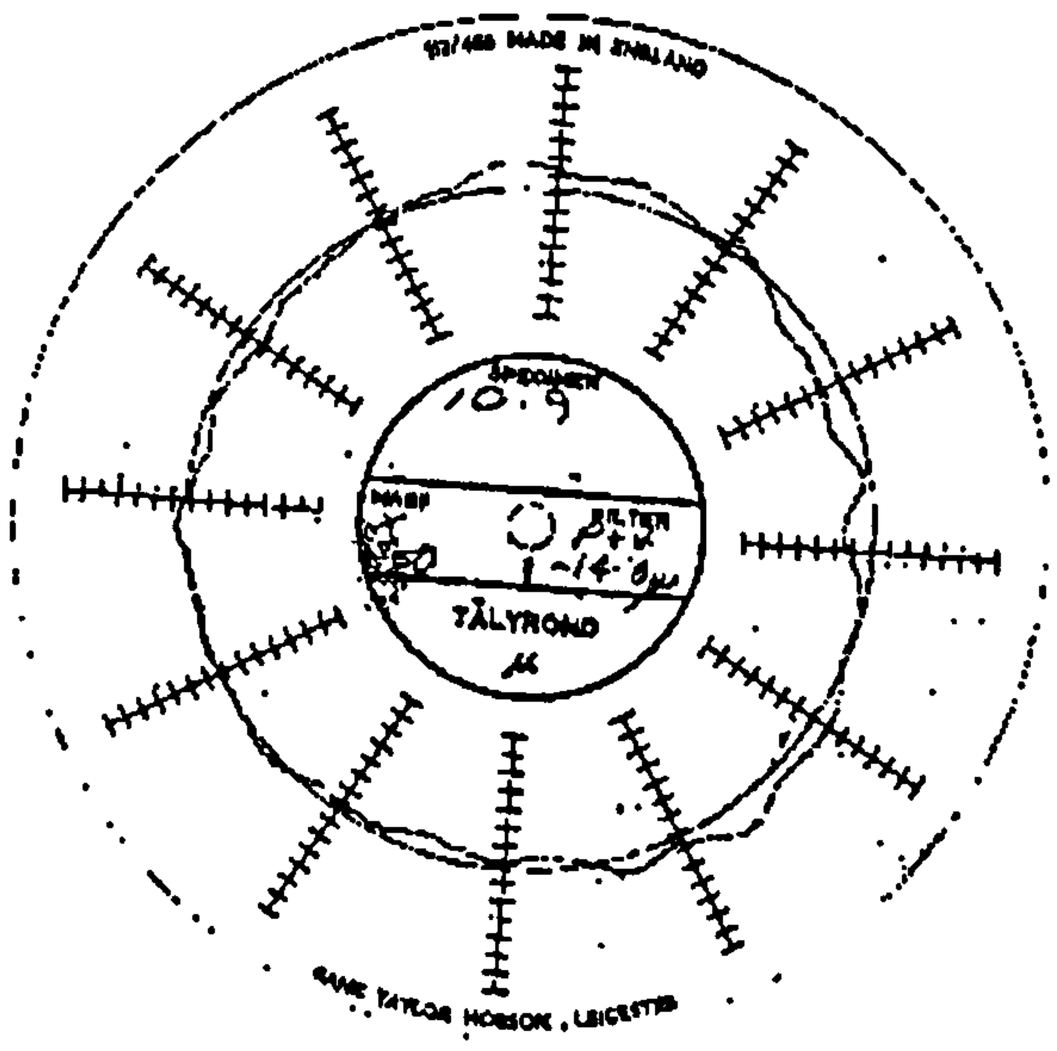
Before an investigation into the capability of the pecking cycle could be performed it was necessary to determine a dwell period for the cycle pecks. Although project objectives were that cycle times were minimised it was important that the dwell period should be large enough to provide satisfactory workpiece roundness. The Talyrond measurements in Figure 36 illustrate the variations in roundness achieved with various dwell periods. A dwell period of 10 seconds was considered satisfactory as an error of 8 μm was within the tolerances specified by GKN.



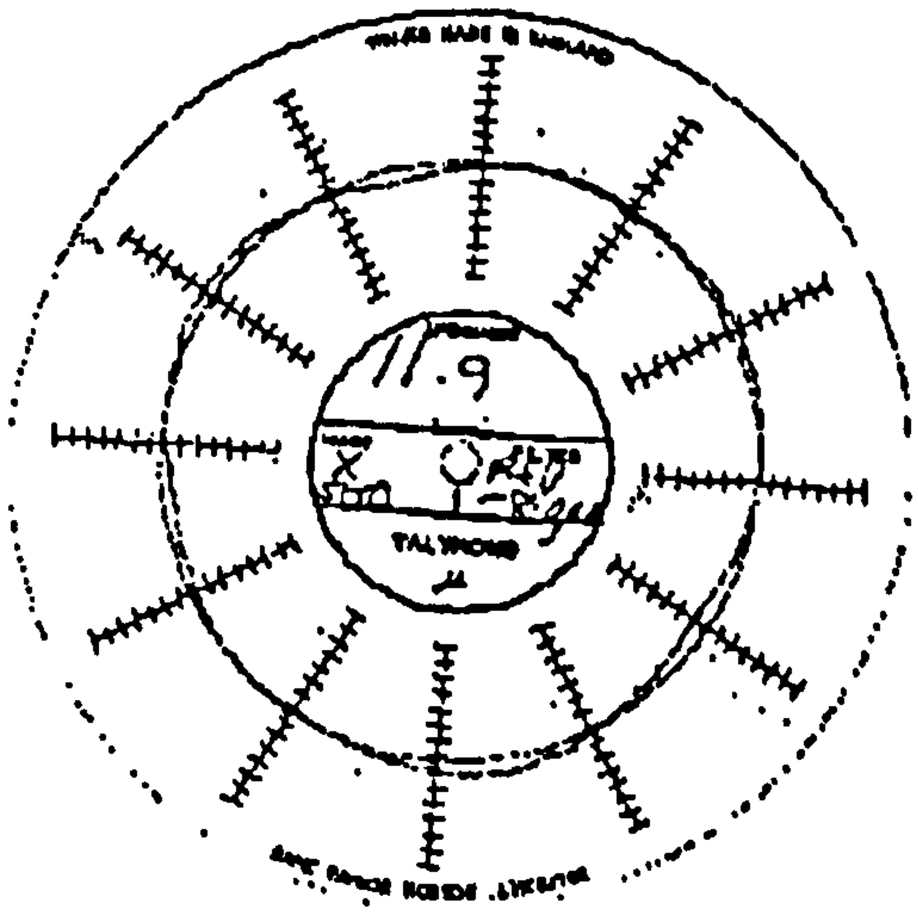

2.5 s dwell - 21 μm roundness




5.0 s dwell - 16 μm roundness




7.5 s dwell - 14 μm roundness




10.0 s dwell - 8 μm roundness

Figure 36. Talyrond charts of cylinder liners for various dwell periods.

After allowing two hours for the machine to reach a steady operating temperature grinding trials were performed on large numbers of compliant workpieces. A graph of deviation of workpiece size from the average batch diameter obtained from post process measurement is given in Figure 37. Measurements of workpiece diameter after grinding a number of compliant workpieces with a target diameter of 95.658 mm are summarised in Table 21. Graphs of the type shown in Figure 38 were produced from measurement of grinding wheel power consumption and workpiece size with external data logging equipment.

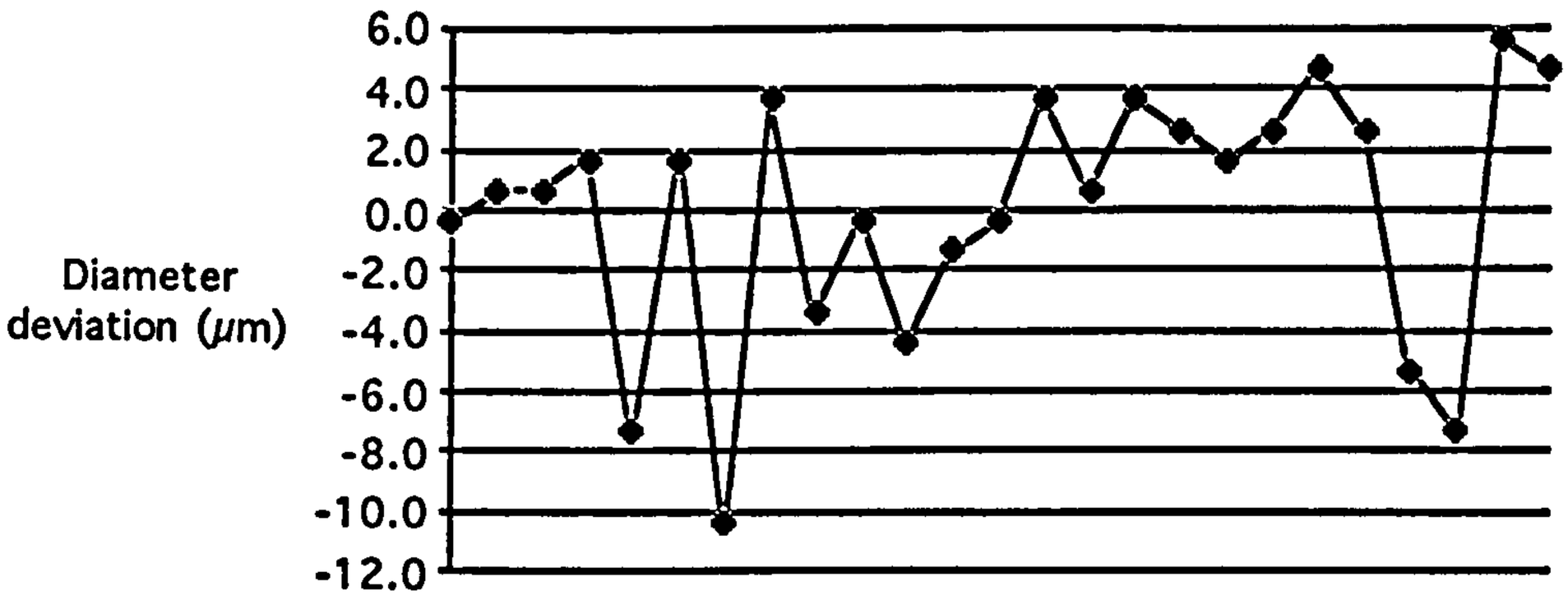



Figure 37. Deviation of workpiece size for compliant cylinder liners.

| Parameter | Compliant workpiece |
|---|---|
| Average diameter | 95.674 mm |
| Maximum diameter | 95.680 mm |
| Minimum diameter | 95.664 mm |
| Standard deviation | 4.107 μm |
| Average cycle time | 64s |

Table 21. Summary of grinding trial results.

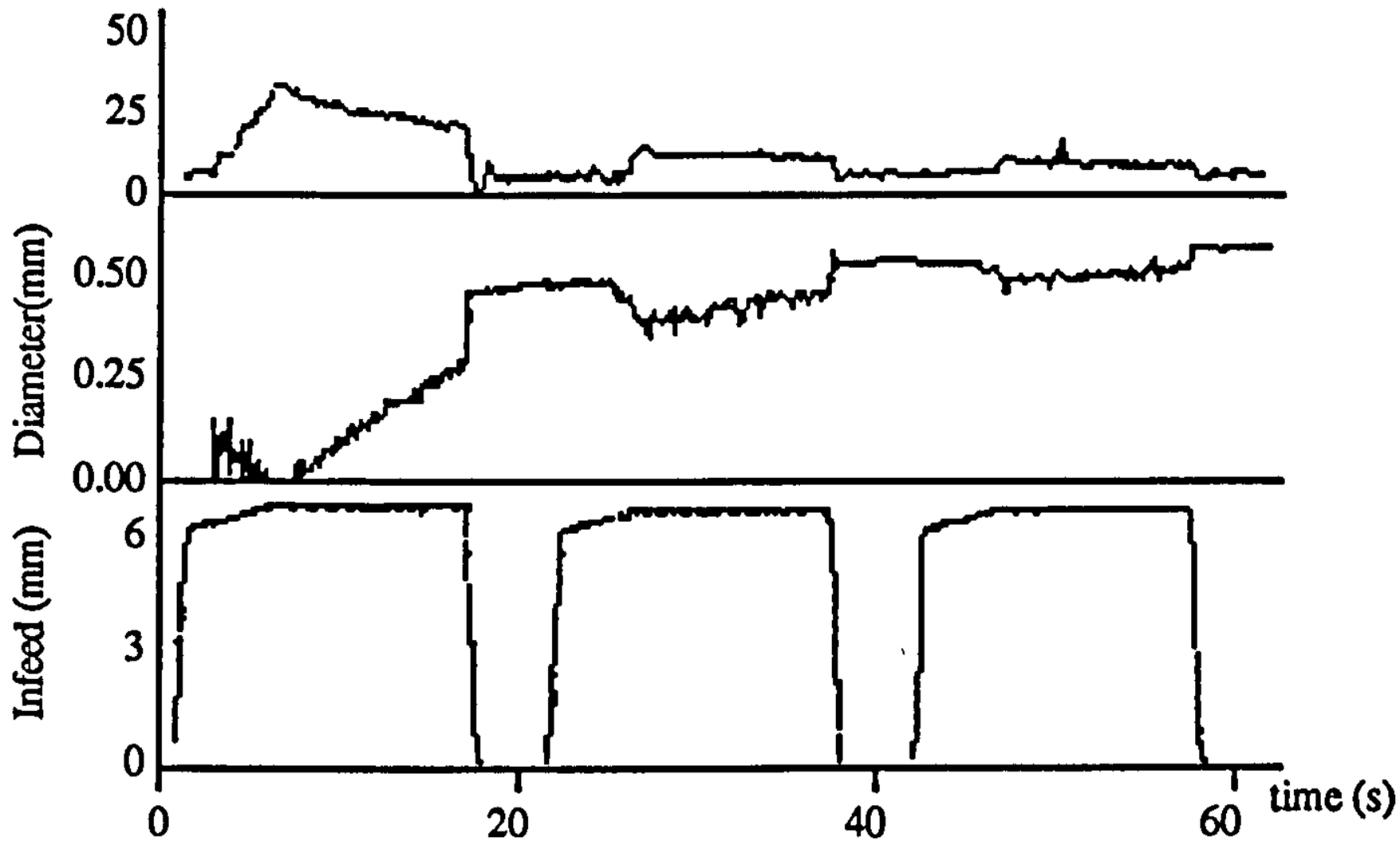



Figure 38. Data log of a pecking cycle for a compliant workpiece.

The pecking cycle was successfully implemented on the developed control system and grinding trials performed on highly compliant cylinder liners of the type produced by GKN. The three peck grinding cycle produced workpieces with a standard deviation of 4.107 μm. Assuming a normal distribution of results and a rejection rate of 0.5% the machine size capability on compliant workpieces was ±11.54 μm. The accuracy achieved was acceptable when compared with the tolerance of ±12.5 μm specified by GKN.

The learned values of system time constant were filtered and stored in the workpiece data file. A graph showing learned values of system time constant is given in Figure 39. The value of system time constant at the point at which the grinding wheel became dull and workpiece surface texture deteriorated was identified by the machine tool operator. As system time constant was used as a measure of grinding wheel sharpness the operator was able to monitor the value of time constant and dress when sharpness declined to the predetermined level.

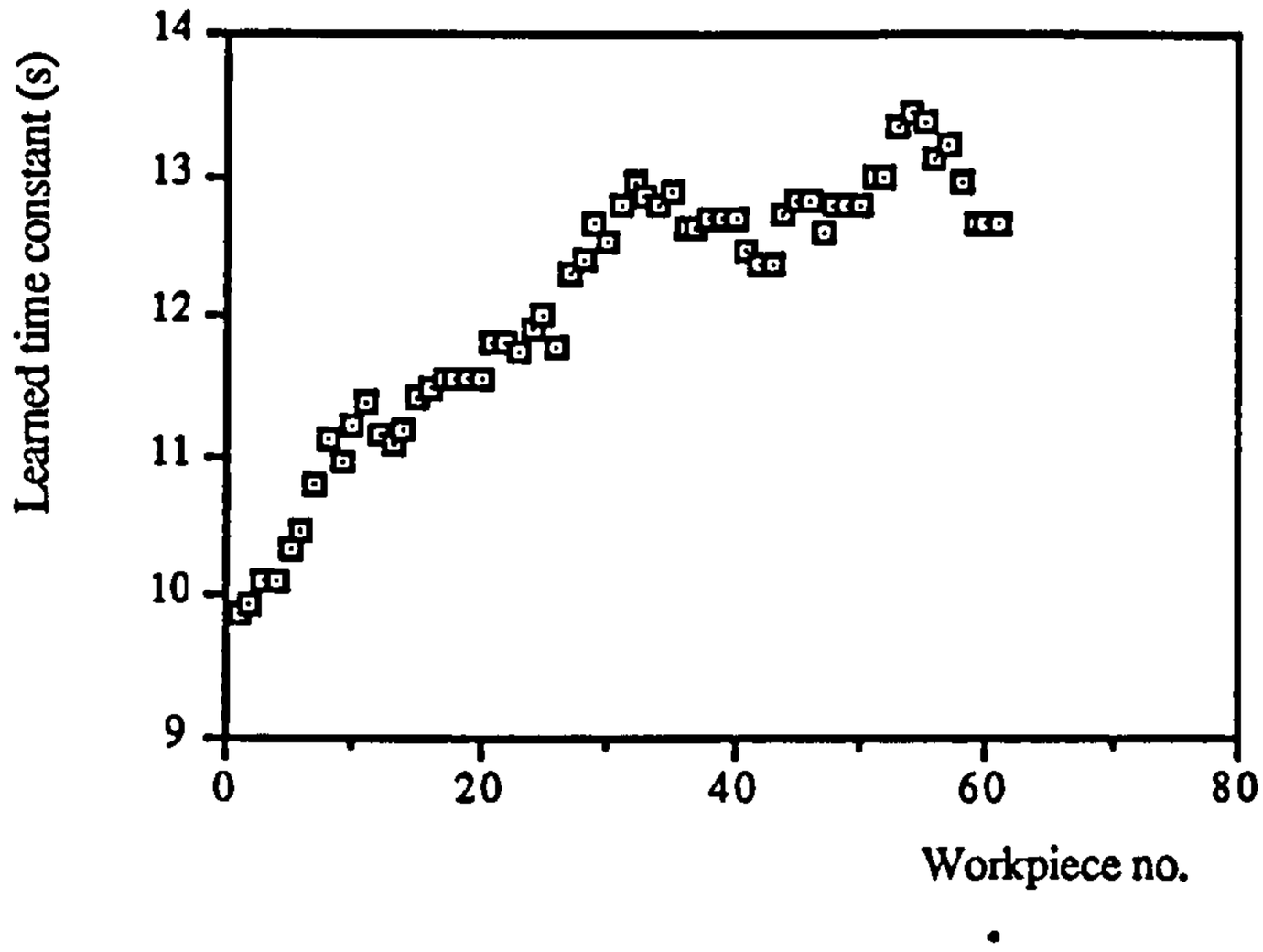



Figure 39. Learned values of system time constant.

The developed control system employed techniques that were designed to improve the centreless grinding process in two ways. Optimised grinding conditions and control of process parameters should result in improved removal rates and workpiece quality. Furthermore, it was intended that production rates were improved by integration of the roughing and finishing cycles into one operation. The production rate for the developed control system operating the pecking cycle was based on the time taken to produce a batch of 50 cylinder liners and is described in Table 22. An average production rate of approximately 80 seconds was achieved. The production rate with a three peck cycle resulted in a saving of 41% over the 136 second production rate achieved with a manually operated machine tool .

| Parameter | Compliant workpiece |
|---|---|
| Wheel dress | 300 s |
| Loading/unloading | 50 x 10 = 300 s |
| Actual grinding | 50 x 64 =3200 s |
| Workpiece floor to floor | (300+500+3200)/50 = 80 s |

Table 22. Summary of grinding trial results.

## 7.5 Evaluation of two peck cycle

Trials of the pecking cycle were performed on-site at GKN Sheepbridge with the aim of further reducing the cycle time. This was achieved by reducing the number of pecks to two. However, although floor to floor times reduced to approximately 60 seconds, these trials compounded problems of workpiece size accuracy, roundness and in particular taper. Figure 40 illustrates the taper experienced with a two peck grinding cycle.

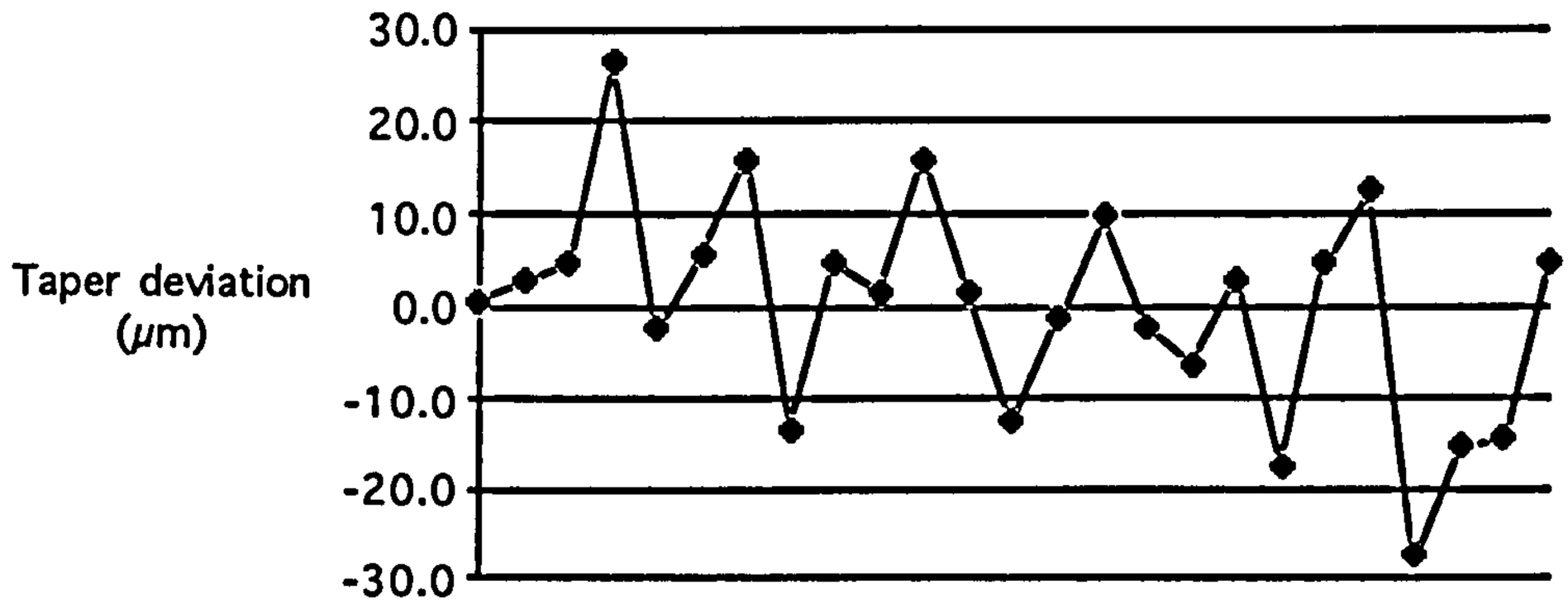



Figure 40. Workpiece taper for two peck cycle.

It was apparent that the shorter grinding cycle required an increase in material removal rate and therefore an increase in grinding forces. As the pecking cycle increased grinding forces through use of larger overshoot values for each peck, the workpiece distortion was increased. The increase in deflection exaggerated the errors arising from variations in compliance along the length of the workpiece.

For the grinding of cylinder liners with the three peck cycle it was experienced that the workpiece scrap rate was higher on-site at GKN than that predicted by results from the laboratory. The production of a scrap workpiece was principally due to workpiece taper problems arising from variations in wall thickness of the cylinder liners. This problem was compounded as batches of rough-turned cylinder liners from different lathes were mixed prior to the grinding operation. It was suggested that the

adoption of a streamed or cell production environment would reduce this particular problem as workpiece quality would be more consistent prior to grinding.

## Chapter 8. General discussion of findings

The work performed encompassed a broad range of objectives. Principally automation of the centreless grinding process for the production of highly compliant, cast iron cylinder liners was achieved. A conceptual framework for modular, intelligent control systems was developed. The modularity of the 8600 C.N.C. system was well suited to the modular design of the intelligent control system framework. Important features were particularly the multi-tasking capabilities, the amount of processing power and the range of input-output options available. The conceptual framework was integrated successfully within the C.N.C. system. Strategies were developed for the improvement of material removal rates with rigid and compliant workpieces and these were incorporated within the intelligent control system.

The performance of the developed machine tool automation was satisfactory for the purposes of the work performed. The modified grinding wheel drive provided enough power to remove material from EN9 steel workpieces of the type produced on such machines at a rate sufficient to cause thermal damage to the workpiece. Also, the speed capability of the drive mechanism was sufficient to run the grinding wheel at the maximum rated speed of most commercial wheels. The control wheel drive electronics were capable of achieving workpiece speeds within the range limited by chatter and roundness problems. The infeed axis provided good control of the infeed axis at typical grinding infeed rates yet maintained a high rapid traverse rate to minimise cycle times. Although infeed axis positioning capability was limited by the resolution of the position feedback transducer to $\pm 2$ μm, this was satisfactory given the tolerances of $\pm 12.5$ μm required by GKN.

However, certain areas of the modified machine tool mechanics were identified that required consideration for future work. It was noted that the grinding wheel speed temporarily fluctuated by up to 10% of the steady state value when a large load was applied suddenly. This effect was particularly exaggerated when contacting a rigid workpiece at a high infeed rate. An effect of the fluctuation in grinding wheel speed

was to cause a corresponding instability in power measurement. However, after a period of around 2 seconds, the grinding wheel drive regained stable operating speed. The prime reason for the temporary fluctuation in grinding wheel speed was the speed regulation capability of the grinding wheel drive electronics. The time constant of the d.c. thyristor drive in response to a sudden load was found to be in the order of 2 to 3 seconds. This sluggish response was due principally to the low resolution of the tachometer used by the drive to monitor motor speed. More modern a.c drives with resolver feedback typically better this value of time constant by a factor of 10 and produce correspondingly better speed regulation.

A further area for consideration involved the infeed axis drive mechanics. As mentioned in Chapter 5, backlash existed between the axis lead screw and the captive nut. This backlash presented a lag to the closed loop positioning system as the position transducer was positioned on one side of the backlash and the drive motor on the other. The amount of backlash dictated a relatively low value of gain for the positioning system in order to avoid instability. The low value of gain resulted in a sluggish response that was particularly evident when accelerating and decelerating the axis. Although such mechanics would cause problems in machines that interpolated two or more axes, the positioning capability of a single infeed axis was not found to be adversely affected.

Other researchers [29,32,35,39-41] advocated the measurement of grinding force as a requirement for control of grinding processes. The Wickman 2K research grinding machine was equipped previously with differential pressure transducers that provided measurement of normal grinding force. However, constraints of finance and time meant that it was not possible to apply such force measurement equipment to the Cincinnati No. 3 grinding machine. Power measurement was implemented as an economic alternative. The power achieved during a grinding cycle is related to tangential grinding force and was used to determine the specific energy and hence efficiency of the process. The values of specific energy achieved during a grinding

cycle were compared with the critical specific energy predicted by the thermal model. This comparison ensured that material removal rates did not exceed the levels at which thermal damage occurred to the workpiece. Throughout the programme of work no workpiece burn was experienced.

In-process measurement of workpiece size was implemented on the Wickman and Cincinnati grinding machines. In both cases the design of the sizing device employed was dictated by the lack of defined workpiece centre and the shortage of available space for calliper type gauges. In response to these constraints an incremental touch probe was employed that incorporated a linear scale. Use of a linear scale device for size measurement ensured high accuracy (2 μm), good repeatability, minimal drift and simple interfacing to the C.N.C. control system. As the probe was an incremental device it was necessary to first datum the probe prior to operation. The sizing device readings taken during a grinding cycle were used for the calculation of grinding cycle parameters and gap elimination purposes. Also, when grinding highly compliant workpieces the sizing devices indicated the level of workpiece distortion.

Both the Cincinnati No. 3 and Wickman 2K centreless grinding machines were electrically interfaced to the OSAI A-B 8600 C.N.C. system. In addition to normal machine control functions, the design of the electrical interface provided the safety features required for automated processes. The first level of safety incorporated hardwired emergency stop and emergency infeed retract functions. A further level of safety was provided by the monitoring of machine status signals by the C.N.C. through the software interface. The machine logic interface created using the SIPROM interface language was responsible for controlling machine functions according to the current mode of operation and provided diagnostics. In conclusion, the design of the machine tool electrical interface was considered to provide the level of safety required from modern grinding machines.

Integration of machine tool operation with that of the C.N.C. system involved elements of mechanical, electrical and software design. The development of the overall design and its application to the machine tools expended a large amount of project resources. However, the purpose of this integration was to provide a suitable platform for development and testing of the intelligent control system structure. The control system design was developed from the modular conceptual framework outlined in Chapter 5. A key requirement of the modular approach to control system design was the specification of an inter-module communication protocol. This protocol ensured the implementation of new module designs was relatively simple. Access to system resources such as the data base and machine I/O by the modules was facilitated by the open architecture design of the 8600 C.N.C. on which the control system was implemented. The concept of autonomous operation of modules has much in common with modern object oriented programming techniques. In brief, the control system featured machining cycle design, operator input and output, data base management and process modelling. Execution of the programmed cycles was controlled by the executor function.

It was intended that the control system design should be applicable to a range of machining processes. However, project objectives involved an investigation into optimisation of the centreless grinding process with particular reference to the production of highly compliant, thin walled cylinder liners. Consequently, there was a requirement for both the application of existing and design of new process models and cycle designs.

Process optimisation through maximising material removal rates required that the control system learned improved values of the controlled grinding parameters of infeed rate and work speed. The rules by which this learning procedure was achieved were based on the fact that material removal rates were optimised at or near the burn boundary [33]. Therefore, optimisation of grinding parameters was based on operation of the process at the boundaries of available machine power and thermal

damage to the workpiece. As machine power was monitored by the control system in order to implement the optimisation strategy it was necessary to model the thermal boundary by incorporating a thermal model. The thermal model [44] employed predicted the power level for the onset of thermal damage based on current grinding parameters and power consumption. This approach generated improved grinding parameters for specific workpiece and grinding wheel geometries. However, it was considered important to incorporate a technique by which improved grinding conditions were learned in a manner that was not specific. To achieve this aim the kinematic model developed by Rowe [33] was employed. By storing values of chip shape ratio and volume, kinematic modelling provided repeatable grinding conditions for a range of workpiece and grinding wheel geometries.

Whilst the strategies employed for optimisation of material removal rate were applicable to a wide range of grinding processes, the highly compliant cylinder liners produced by GKN required that process optimisation was achieved in an alternative manner. An effect of both machine and workpiece compliance is that deflections arising from grinding forces cause a lag between applied and actual infeed position. The extreme compliance of the cylinder liners led to large deflections and ensured that the material removal rates were limited as steady state grinding conditions were not achieved. Analysis of the manual production techniques in use at GKN indicated that only 25% of the time taken to produce a cylinder liner involved actual grinding. The remaining time was spent on the non-productive loading, unloading, dressing and measuring operations arising from the separate rough and finishing cycles. Therefore in order to optimise the process it was important to consider the production cycle as a whole rather than concentrate on removal rate objectives.

A major improvement in productivity was achieved through the removal of separate rough and finish grinding processes. This was possible due to improvements in grinding conditions arising from the use of high grinding wheel speeds. Under

these conditions it was found that satisfactory surface texture was obtained without compromising the removal rates necessary for the roughing cycle.

A compliance model [28] was integrated within the control system in order to predict the profile of true infeed position during a grinding cycle given knowledge of system compliance. A pecking cycle that used an overshoot position predicted by the compliance model was devised. The pecking cycle divided the grinding cycle into a number of rough and finish stages. For each stage or peck a target diameter was programmed and the compliance model calculated the target infeed position that produced the required true infeed position. After each peck the sizing devices were used to determine the true workpiece compliance from the error between predicted and actual material removal.

Trials performed with the pecking cycle indicated that three pecks provided optimum performance. The three peck cycle yielded a 41% reduction in cycle time over the production rate achieved with a manually controlled machine tool. Workpiece quality was within the range specified by GKN. A manually controlled machine tool produced workpieces at a rate of 26 per hour which, based on an estimated operating cost of £10 per hour, resulted in a cost of 38p per liner. However, the cost per liner was reduced with the automated process to 22p per liner. Therefore, assuming an approximate cost of £30,000 for automation of the machine tool, outlay was recovered after production of 187,500 cylinder liners. With the machine producing liners for 16 hours per day the payback period was approximately 1 year. It was evident, based on these figures, that automation by retro-fitting was an economic solution to the production of cylinder liners.

Various techniques and cycles were implemented before the three peck grinding cycle was proposed. A two peck cycle was rejected as errors in workpiece roundness before grinding produced unreliable sizing device measurements prior to the roughing cycle. These measurements led to errors in the compliance calculations and adversely affected the performance of the final finishing cycle. The three peck cycle improved

matters as the first peck was effectively a workpiece rounding cycle. Subsequent pecks based on reliable workpiece size information were able to accurately rough and finishing the workpiece.

The intelligent control system adapted readily to the different strategies employed in the production of the cylinder liners. From early system trials with conventional, rigid workpieces to the pecking cycle required only integration of the appropriate modules and creation of the necessary cycle description. The data base stored workpiece descriptions for a range of cylinder liners produced by GKN and the materials from which they were produced. Grinding wheel data was limited as GKN regularly used only one grade of grinding wheel.

However, workpiece taper was a problem beyond the capability of the control system at the point to which it was developed. Attempts to further reduce cycle times by increasing system deflection and reducing dwell periods led to both taper and roundness problems. The workpiece taper experienced varied widely from workpiece to workpiece and was the result of inaccuracies in the early stages of the manufacturing process. Whilst it was conceivable to incorporate a model of workpiece taper within the control system structure, this was beyond the scope of the project. Particularly, there was no method available on the machine tool by which it was possible to automate the control wheel angle adjustment and hence provide control over workpiece taper.

The internal and external diameters of the cylinder liners were rough turned prior to finishing by the centreless grinding process. A number of multi-spindle lathes were employed to perform the rough turning operation. The dimensional accuracy of a rough turned cylinder liner depended on the condition of the lathe and the spindle on which it was produced. The manufacturing process employed by GKN ensured that consecutive workpieces presented for grinding were not rough turned by the same lathe. It was suggested that streamed or cell production would improve the consistency

of rough turned workpieces supplied for grinding. It was considered that this approach would minimise the taper problem and allow further reductions in cycle time.

# Chapter 9. Conclusions

An improvement in the production rate of highly compliant cylinder liners by the centreless grinding process can be achieved through automation of the machine tool and integration of an intelligent control system within a commercial C.N.C. It has been shown that retro-fitting of an intelligent control system to a manually controlled grinding machine allows a payback period of 1 year to be attained.

Process modelling, learning strategies, adaptive strategies and process measurement are key requirements for intelligent control of grinding processes. Benefits of creating a conceptual framework for an intelligent control system that integrates these requirements with data base management, machine control and operator interaction have been demonstrated. Adoption of the conceptual framework allows an intelligent control system to be developed rapidly and in an organised manner. Definition of the distinct, modular control system elements and the interface between these elements simplifies the development and integration of the control system strategies.

Whilst the merits of integrating the control system framework within a host computer system can be argued it is concluded that there are benefits arising from integration of the control system within a commercial C.N.C. system. The primary reason for this conclusion is to reduce system complexity and cost. The 8600 C.N.C. system is amenable to integration of the control system framework by virtue of its multi-tasking operating system and the ability to incorporate high-level language programming modules.

For highly compliant workpieces an increase in material removal rate through the adaptive control of infeed rate and workpiece speed does not provide a significant increase in workpiece production rate over that achieved with a manually controlled machine. This is because separate rough and finishing processes ensure that only a small proportion of production time is actually spent grinding. Also, the high level of

compliance ensures that steady state deflections are not achieved during a grinding cycle. Therefore, the material removal rates that can be achieved are limited by the system compliance. The ability to reduce the non-productive cycle time is identified as the most effective way of improving production rate. Improvements in the capability of the manual machine tool can be achieved as a consequence of automation and interfacing to a C.N.C. In particular, employing a high speed, high power grinding wheel drive system permits improved material removal rates whilst maintaining satisfactory workpiece surface finish. It is therefore not necessary to perform separate rough and finish grinding cycles with an automated machine. Also, the control of the infeed axis position and feed rate that results from machine automation ensures that the grinding cycles specified by the control system are performed accurately.

System deflections that are experienced during a cycle are due to the effect of grinding conditions and hence grinding force on the compliant workpiece and machine. An intelligent control system is able to compensate for system deflections and produce highly compliant cylinder liners to the specified size tolerances. Use of an overshoot infeed position permits dwell times to be reduced for highly compliant workpieces. Incorporation of a compliance model within a control system allows the overshoot position required for a workpiece to be predicted based on a value of system time constant.

Implementation of a pecking cycle strategy allows a control system to compensate for variable system compliance. A pecking cycle requires that particular process models, learning strategies, process measurement and machine control cycles are integrated within the framework of a control system. Specifically, such a control system is able to model the effects of system compliance and learn the true compliance of each individual workpiece through in-process measurement of workpiece size. Therefore, a pecking cycle is able to use data gathered from roughing cycles in order to predict an appropriate finishing cycle. Changes in grinding conditions due to such factors as grinding wheel wear are characterised by a corresponding change in system

time constant. An intelligent control system can account for these changes by learning the current value of system time constant and modifying the grinding cycle accordingly. Results of grinding trials indicate that implementation of a pecking cycle within an intelligent control system allows highly compliant workpieces to be produced with significantly reduced floor to floor times.

# Chapter 10. Suggestions for future work

It is considered that use of the control system conceptual framework simplified development of the intelligent control system for the centreless grinding process. However, the framework was not applied to other processes during the course of this work. The modular nature of the control system suggests that it could be applied to other processes. In particular the external and internal cylindrical grinding process is amenable to the concepts proposed. The principal difference between external cylindrical and centreless grinding involves the use of a centred workpiece in the cylindrical process. Otherwise there are many similarities between the two processes. There has been much research into intelligent control of the cylindrical grinding process involving process modelling and adaptive control. It could be beneficial if the conceptual framework was used to structure and integrate these varied ideas into a realisable intelligent control system.

An improvement in material removal rates through the optimisation of grinding conditions for the centreless grinding process was a feature of the intelligent control system. Optimisation was performed using adaptive control of infeed rate and work speed to approach the burn boundary. A thermal model was employed to ensure grinding conditions did not produce workpiece burn. However, it was clear that high material removal rates were not possible for the cylinder liners due to the values of compliance experienced. Consequently the work concentrated on the development of alternative techniques for reducing overall cycle times. Further trials with the thermal model and optimisation strategies for more rigid workpieces would provide useful data for comparison with other optimisation techniques. The strategies employed could equally well be tested on other grinding processes. This would require development of appropriate thermal models and identification of the particular process limits.

The strategies employed for optimisation of grinding conditions were designed to adjust the controlled grinding parameters after each grinding cycle. This approach is suitable for high volume production where productivity is not affected significantly by the small number of grinding operations required for learning a workpiece and grinding times are relatively low. However, the control system was capable of performing adaptive control of infeed rate in real-time. Further system development and testing of techniques for controlling infeed rate to provide constant grinding power could be beneficial. It is suggested that constant power grinding could improve productivity in applications where long grinding times are required. Also, the ability to control infeed rate dynamically would allow implementation of gap elimination techniques based on the measured rise in power that occurs when the workpiece first contacts the grinding wheel.

System time constant describes the effect of grinding forces and system compliance on machine and workpiece deflections. Therefore, the value of system time constant depends on machine and workpiece compliance together with the current grinding conditions. Measurement of system time constant provides a system with the ability to cater for compliance and the changes in grinding forces that occur due to varying grinding conditions. A compliance model was developed for the centreless grinding of highly compliant cylinder liners. The model allowed the control system to predict grinding cycles that minimised dwell times by using an overshoot position. Extended dwell times are often required when grinding small diameter workpieces by the cylindrical grinding process. Further development of the compliance model could provide a system with the ability to reduce cycle times for the cylindrical process through use of an overshoot position. The pecking cycle might equally be employed to provide measurement of system time constant.

A rule employing a simple filter was employed for learning the changing value of system time constant. A similar rule was used for maximising power by adapting infeed rate on the basis of power achieved during the previous grinding cycle. Alternative strategies could be investigated that allow more information about the process to be extracted from the measured data than is possible with a filter of the type employed. Such learning strategies could be easily implemented within the developed control system framework.

Other techniques for the measurement of system time constant are possible using high speed data acquisition systems. In particular measurement of the exponential rise and fall of grinding wheel power during the infeed and dwell phases of the cycle could allow a system to derive values of system time constant in real-time. This could potentially eliminate the requirement for a pecking cycle to learn the compliance of an individual workpiece by performing multiple passes.

The work described utilised power measurement to provide an indication of grinding forces and to derive system time constant. This was a practical approach considering the design of the machine tool and economic restraints. The incorporation of force measurement by machine tools designers would provide a more accurate indication of grinding conditions. This approach would allow derivation of both normal and tangential grinding force whilst eliminating the effects of grinding wheel drive system response that are experienced with power measurement. It is envisaged that the cost of force measurement systems would be minimised when designed into a machine tool from the outset. Applicable techniques include the differential measurement of oil pressure from hydrostatic grinding wheel bearings and the use of load cells within the workplate arrangement.

An investigation into techniques for the correction of workpiece taper in the centreless grinding process could be beneficial. Taper problems are experienced with both rigid and compliant workpieces. Particular difficulties are experienced when

grinding flanged cylinders. Most centreless grinding machines feature manual adjustment of the control wheel angle in the horizontal plane to allow a machine setter to minimise taper. However, the degree of taper experienced may vary due to changes in grinding conditions and the corresponding variations in grinding forces. Continual adjustment of control wheel angle may therefore be required. It is suggested that automation of control wheel angle adjustment would provide the capability for an intelligent control system to control workpiece taper.

## References


1. R.S. Hahn and R.P. Lindsay. The influence of process variables on material removal, surface integrity, surface finish and vibration in grinding. Proc. Int. MTDR Conf., 17-19 September 1969, 95-117.
2. W.B. Rowe and J.L. Moruzzi. Adaptive control of the centreless grinding process. SERC research grant proposal. 1985.
3. P.A.S. Ralston and T.L. Ward. Adaptive control of machine tools: the past and projected role of numerical control computers. Computers ind. Engng., 1988, Vol. 14, No. 2, 85-94.
4. A.G. Ulsoy and Y. Koren. Application of adaptive control to machine tool process control. IEEE Control Systems Magazine, June 1989, 33-37.
5. W. Koenig and G. Werner. Adaptive control optimisation of high efficiency external grinding -concept, technological basics and application. Ann. CIRP, 1974, Vol. 23, No. 1, 101-102.
6. G. Amitay, S. Malkin and Y. Koren. Adaptive control optimisation of grinding. ASME Journal of Engineering for Industry, 1981, Vol. 103, 103-108.
7. E. Brinksmeier. A selftuning adaptive control system for grinding processes. Ann. CIRP, 1991, Vol. 40/1/1991, 355-358.
8. Y. Gao and B. Jones. An optimum size and roundness adaptive control method for the plunge grinding process. Proc Instn Mech Engrs, 1992, Vol. 206, 107-116.
9. H. Kaliszer. Adaptive control in grinding processes. Proc. 4th Intnl. Conf. Prod. Eng. (Tokyo), 1980, 579-593.
10. T.S. Stelson, R. Komanduri and M.C. Shaw. Manual adaptive control of a centreless grinding operation. Int. J. Prod. Res., March/April 1979, Vol. 17, No. 2, 111-122.
11. H. Xiu-Shou. The research of a practical adaptive control system for external cylindrical grinding process. Shanghai Machine Tool Works.

12. Y. Gao and B. Jones. A discrete control system model for the traverse grinding process. Proc Instn Mech Engrs, 1992, Vol. 206, 19-27.

13. J. Peters and R. Aerens. Optimisation of three phase grinding cycles of a series without intermediate dressing. Ann. CIRP, 1980, Vol. 29/1/1980, 195-197.

14. W.B. Rowe and M.M. Barash. Computer method for investigating the inherent accuracy of centreless grinding. Int. J. Mach. Tool Des. Res., 1964, Vol. 4, 91-116.

15. N.G.S. Udupa, M.S. Shunmugam and V. Radhakrishnan. Three dimensional geometric analysis of the plunge centreless grinding process. Proc Instn Mech Engrs, 1987, Vol. 201, 309-320.

16. G.J. Trmal. In-process control of size of a ground component. Proc. 20th MTDR, 1979, 405-411.

17. H. Kaliszer. In-process gauging as applied to the grinding process. Tech. Conf. Central London Poly., April 1977, Session 2, Paper 2.

18. A.G. Rao and S. Malkin. Process monitoring for intelligent control of grinding. SME, 1990, MR90-512, 1-11.

19. F. Hashimoto, A. Kanai and M. Miyashita. Growing mechanism of chatter vibrations in grinding processes and chatter stabilisation index of grinding wheel. Ann. CIRP, 1984, Vol. 33/1/1984, 259-263.

20. P.D. Singhal and H. Kaliszer. The effect of workpiece dimensions and wheel parameters on the surface waviness during grinding. Proc. Int. MTDR Conf., 13-15 September 1965.

21. W.B. Rowe, M.M. Barash and F. Koenigsberger. Some roundness characteristics of centreless grinding. Int. Jnl. MTDR, 5, 203-215.

22. W.B. Rowe. An experimental investigation of grinding machine compliances and improvements in productivity. Proc. Int. MTDR Conf., 1974, 479-485.

23. M. Frost, B.J. Orton and J.L. Tidd. Lobing control in centreless grinding. SME, 1988, MR88-610.

24. F. Hashimoto, J. Yoshioka and M. Miyashita. Development of an algorithm for giving optimum setup conditions for centreless grinding operations.SME, 1986, MR86-628, 1-14.

25. M. Miyashita, F. Hashimoto and A. Kanai. Diagram for selecting chatter free conditions for centreless grinding. Ann. CIRP, 1982, Vol. 31/1/1982, 221-223.

26. W.B. Rowe and K.J. Stout. Review of grinding process parameters. Production Technology, October 1971, Vol. 32, No.10, 41-48.

27. S. Malkin. Grinding technology, theory and application of machining with abrasives. Ellis Horwood, 1989, ISBN 0-85312-765-5.

28. D.R. Allanson, S. Kelly, S. Terry, J.L. Moruzzi and W.B. Rowe. Coping with compliance in the control of grinding processes. Ann. CIRP, 1989, Vol. 38/1/1989, 311-314.

29. R.S Hahn. Improving performance with force adaptive grinding. Manufacturing Engineering, October 1986, 73.

30. F. Hashimoto, A. Kanai A. and M. Miyashita. High precision trueing method of regulating wheel and effect on grinding accuracy. Ann. CIRP, 1983, Vol. 32/1/1983, 237-239.

31. R.I. King and R.S. Hahn. Handbook of modern grinding technology. Chapman and Hall, 1986, ISBN 0-412-01081-X.

32. R.S. Hahn. Some characteristics of controlled force grinding. Proc. Int. MTDR Conf., 13-15 September 1965.

33. W.B. Rowe, W.F. Bell and D. Brough. Limit charts for high removal rate centreless grinding. Proc. Int. MTDR Conf., 1986.

34. G. Sweeney. Trends in grinding. Seminar at Coventry Poly., June 1987.

35. D. Brough, W.F. Bell and W.B. Rowe. Achieving and monitoring high-rate centreless grinding. Proc. Int. MTDR Conf., 1980, 313-322.

36. R.L. Palmer. Grinding machine design and development. Tech. Conf. Central London Poly., April 1977, Session 2, Paper 1.

37. W.B. Rowe, S. Spragget and S. Gill. Improvements in centreless grinding machine design. Ann. CIRP, 1987, Vol. 36/1/1987, 207-210.

38. S. Malkin and Y. Koren. Off-line grinding optimisation with a micro-computer. Ann. CIRP, 1980, Vol. 29/1/1980, 213-215.

39. D.P. Stoten and J.E. Morgan. The development of a research grinding machine incorporating force active control. Dept. Mech. Eng., University of Bristol.

40. V.L. Romanov. Controlled force grinding corrects roundness errors. Source unknown.

41. H.K. Tönshoff, M. Zinngrebe and M. Kemmerling. Optimisation of internal grinding by micro-computer based force control. Ann. CIRP, 1986, Vol. 35/1/1986, 293-296.

42. S. Spiewak and H. Kaliszer. Multi-microprocessor control for plunge grinding. Proc. Int. MTDR Conf., 1983, 247-257.

43. S. Malkin. Burning limit for surface and cylindrical grinding of steels. Ann. CIRP, 1978, Vol. 27/1/1978, 233-236.

44. W.B. Rowe, J.A. Pettit, A. Boyle and J.L. Moruzzi. Avoidance of thermal damage in grinding and prediction of the damage threshold. Ann. CIRP, 1988, Vol. 37/1/1988. 327-330.

45. W.B. Rowe, M.N. Morgan, J.A. Pettit and L.S. Lavine. A discusion of thermal models in grinding. SME, 1990, MR90-516, 1-15.

46. N.R. Des Ruisseaux and R.D. Zerkle. Temperature in semi-infinite and cylindrical bodies subjected to moving heat sources and surface cooling. Trans. ASME, Journal of Heat Transfer, 456-464.

47. A.S. Lavine. Thermal aspects of grinding with CBN wheels. Ann. CIRP, 1989, 38/1/1989, 557-559.

48. T. Jianshe, P. Xuefeng, X. Hongjun X and Z. Youzhen. The prediction and control of workpiece burn during grinding process -Application of graphite penetrated wheels. Ann. CIRP, 1986, Vol. 35/1/1986, 227-230.

49. J.C. Outwater and M.C. Shaw. Surface temperature in grinding. Trans. ASME, 1952, Vol. 74, 73-85.

50. G.R. Shafto, T.D. Howes and C. Andrew. Thermal aspects of creep feed grinding. Proc. Int. MTDR Conf., 1975, 31.

51. OSAI A-B. 8600 MC-TC Application manual.

52. OSAI A-B. 8600 MC-TC Software characterisation and interface.

53. OSAI A-B. 8600 MC-TC Programming and operator's manual.

54. OSAI A-B. 8600 MC-TC ASSET programming manual.

55. OSAI A-B. 8600 MC-TC Siprom programmers manual.

56. OSAI A-B. 8600 MC-TC CSI programmers manual.

57. Intel software development tools. PL/M-86 User's guide for DOS systems. 1985.

58. Intel software development tools. ASM86 Assembly language reference manual. 1985.

59. S. Malkin and Y. Koren. Optimal infeed control for accelerated spark-out in plunge grinding. ASME Journal of Engineering for Industry, 1984, Vol. 106, 103-108.

60. R.P. Lindsay. Sparkout behaviour in precision grinding. SME, 1972, MR72-205, 1-25.

61. L.T. Notter and M.W. Bailey. New dressing techniques for grinding wheels. Tech. Conf. Central London Poly., April 1977, Session 1, Paper 2.

62. M. Miyashita, J. Yoshioka and F. Hashimoto. New concept of grinding technology for predictability of manufacturing operation.SME, 1986, MR86-645, 1-19.

63. J. Bryan. International status of thermal error research (1990), Ann. CIRP, 1990, Vol. 39/2/1990, 645-656.

64. D.E. Clough and S.J. Park. A novel dead-time adaptive controller. Proc. IFAC workshop, Frankfurt am Main, 1985, 21-26.

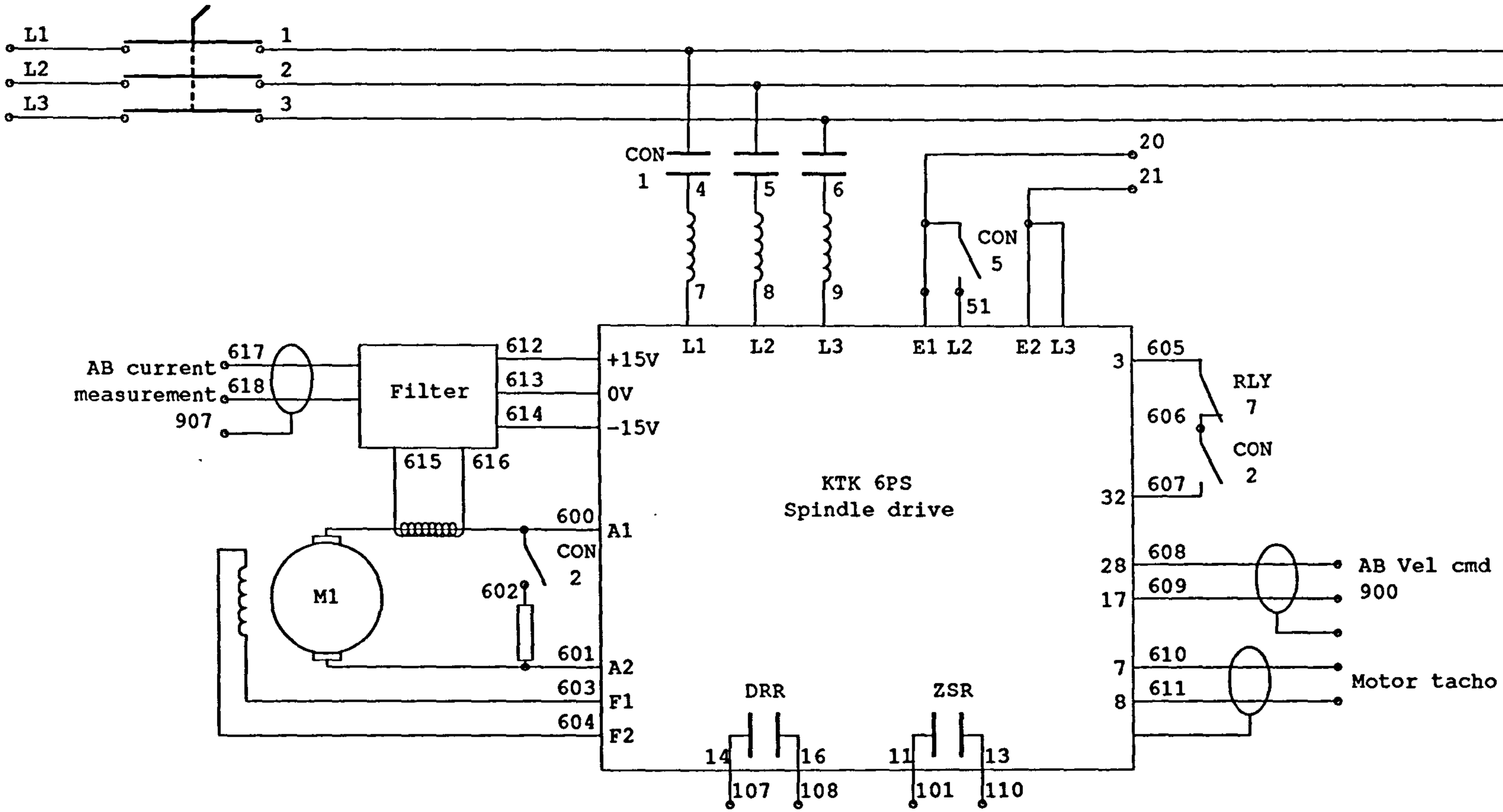
## Appendix 1. Publications based on the project work

This appendix includes papers related to the work described in this thesis that were published by the research group during the course of the project. The paper entitled 'Intelligent CNC for grinding' was awarded the 1991 Joseph Whitworth Prize by the Engineering Manufacturing Industries Division of the Institution of Mechanical Engineers. Other publications by members of the research group are listed in the reference section of this thesis.

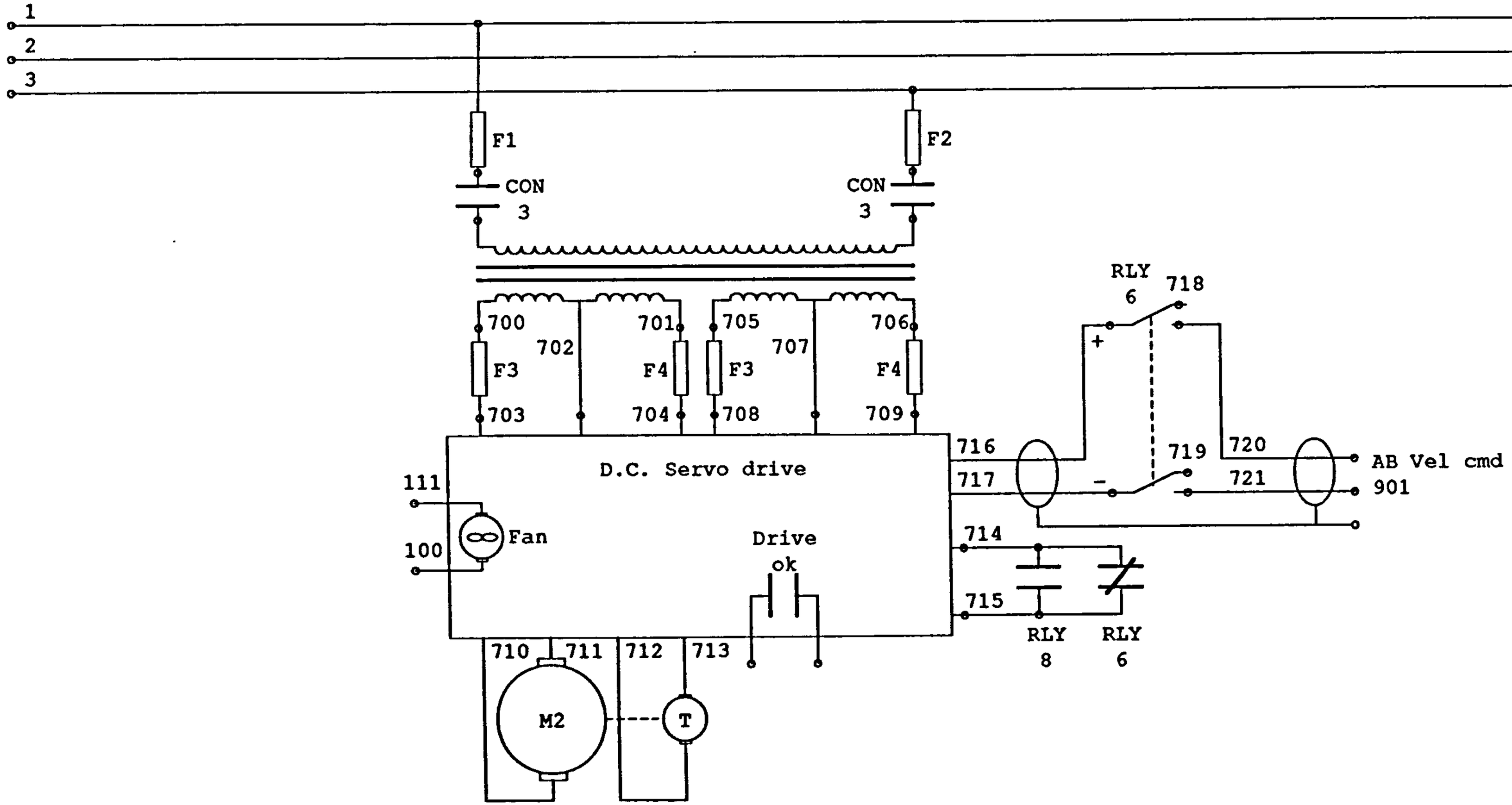
PAGE/PAGES
EXCLUDED
UNDER
INSTRUCTION
FROM
UNIVERSITY

## Appendix 2. Electrical interface diagrams

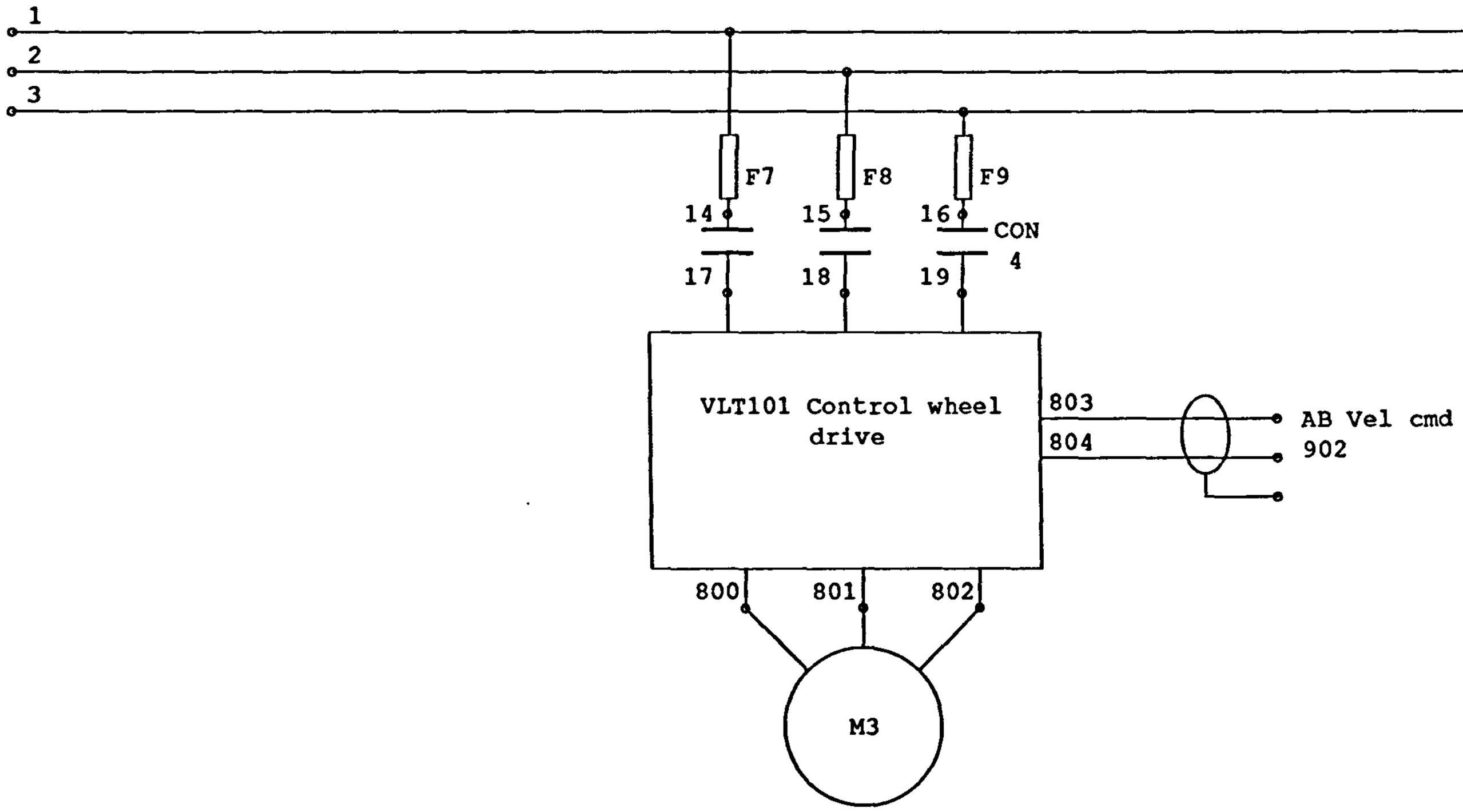
The design of the electrical interface required for integration of the automated Cincinnati No.3 centreless grinding machine with the OSAI A-B 8600 MC/TC C.N.C. system is included in this appendix. A double bay panel fitted with heat exchanger was used to house the electrical components and the C.N.C. rack. The C.N.C. console and operator push button station were housed in a pendant to allow the operator access to the control system whilst operating the machine.



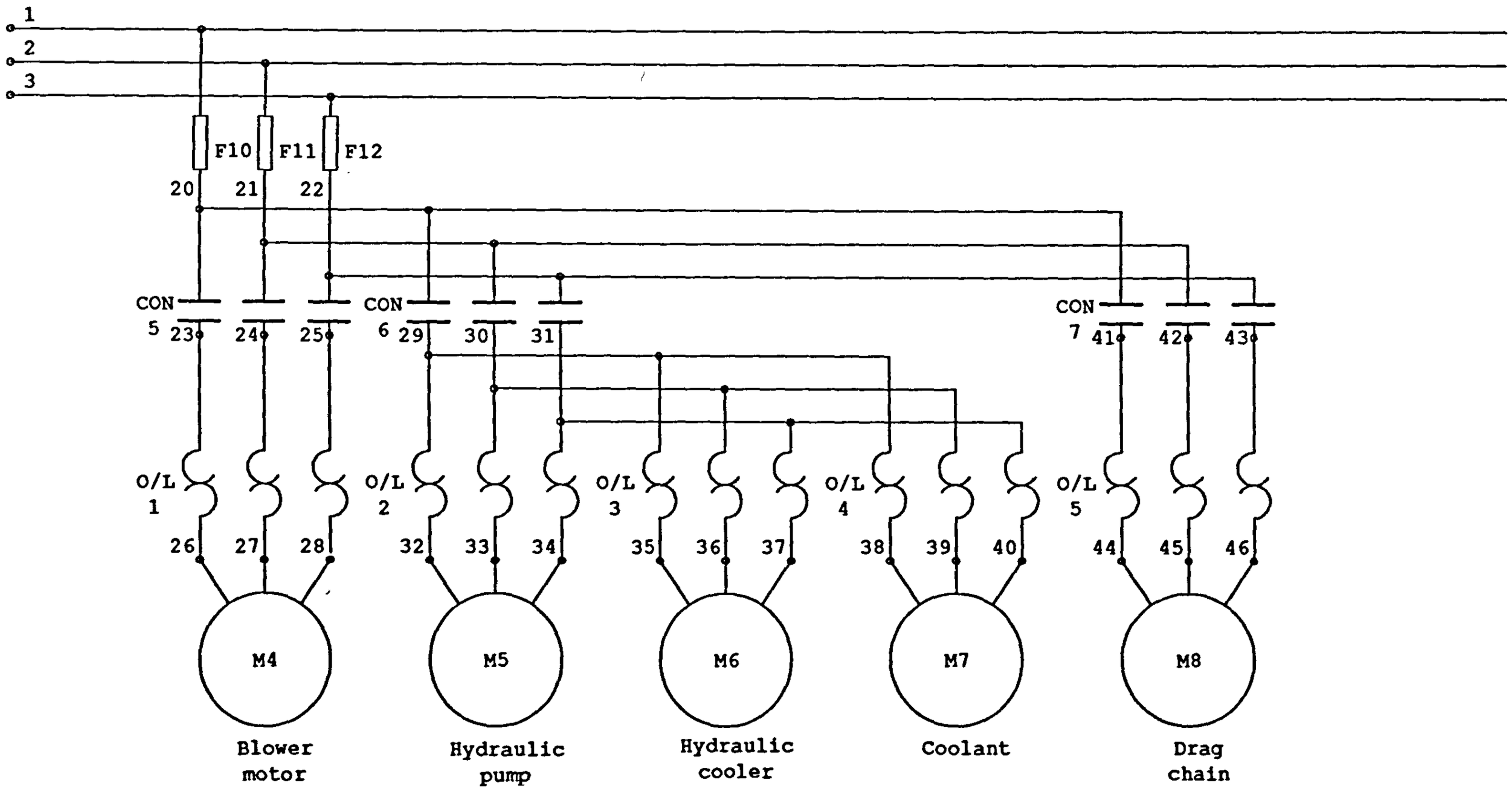

| | | |
|---|---|---|
| Cincinnati No.3 Centreless Grinding Machine | Spindle drive connections | Sheet 1 of 12 |



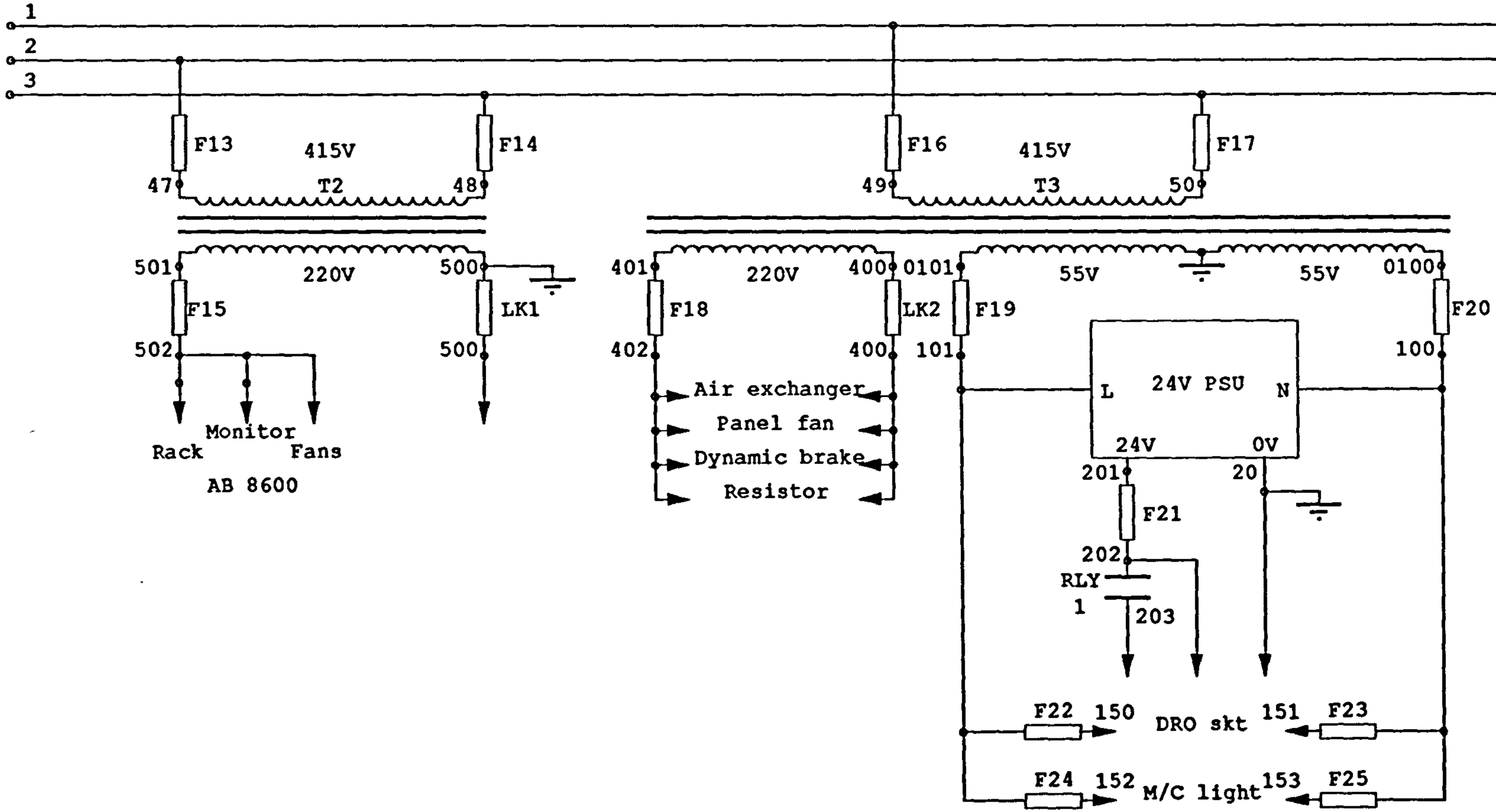

| | | |
|---|---|---|
| Cincinnati No.3 Centreless Grinding Machine | Infeed axis connections | Sheet 2 of 12 |



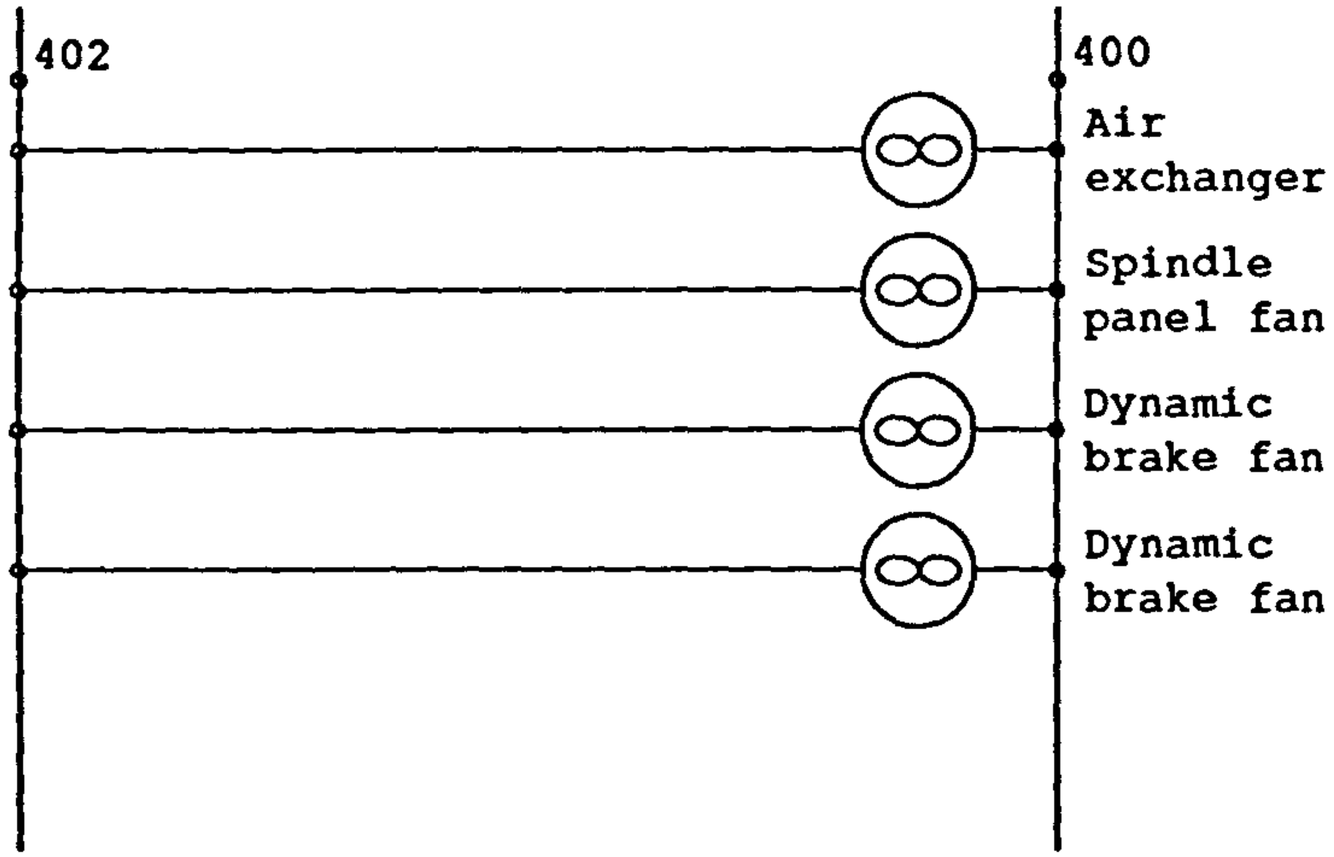

| | | |
|---|---|---|
| Cincinnati No.3 Centreless Grinding Machine | Control wheel connections | Sheet 3 of 12 |



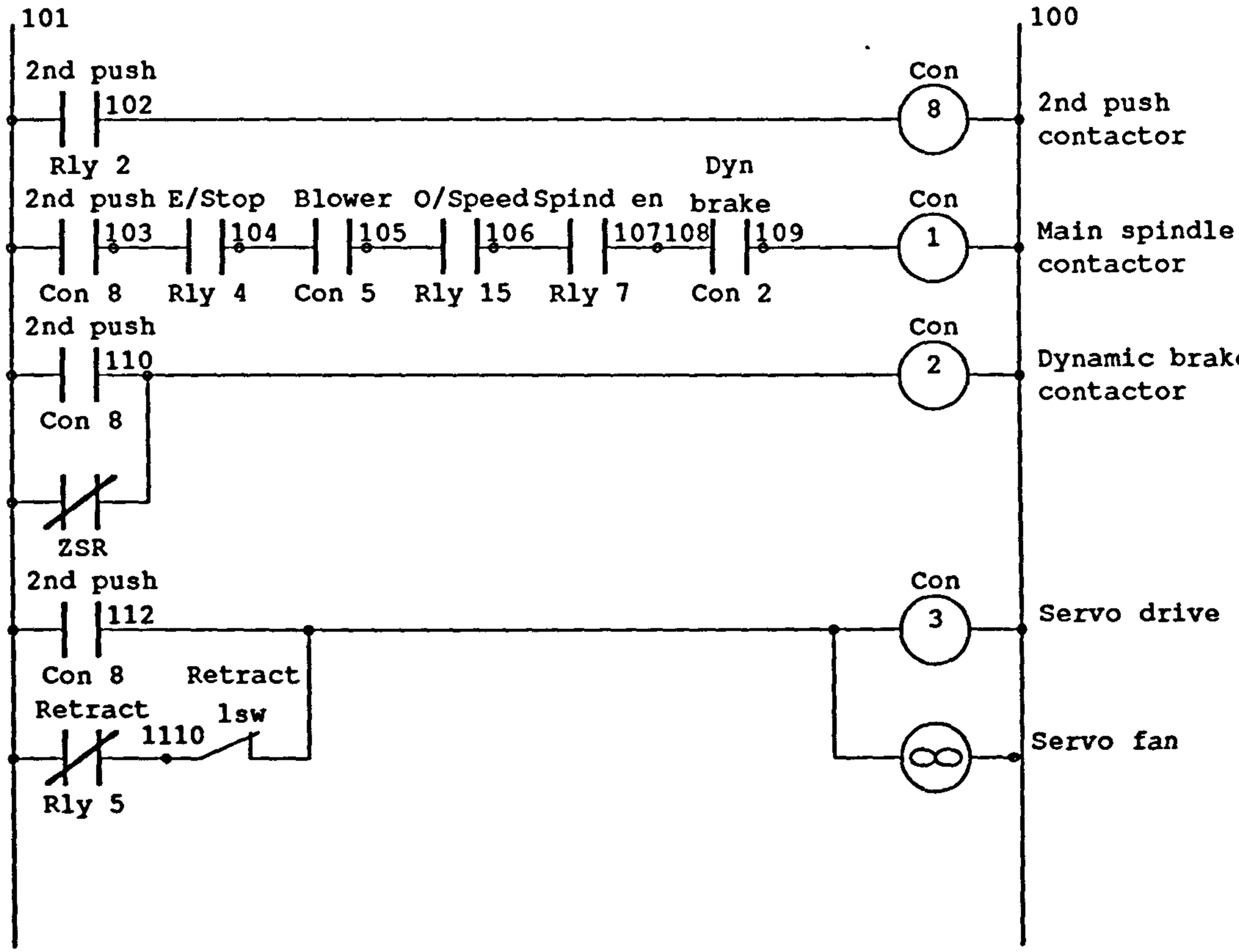

| | | |
|---|---|---|
| Cincinnati No.3 Centreless Grinding Machine | 3 phase motor connections | Sheet 4 of 12 |



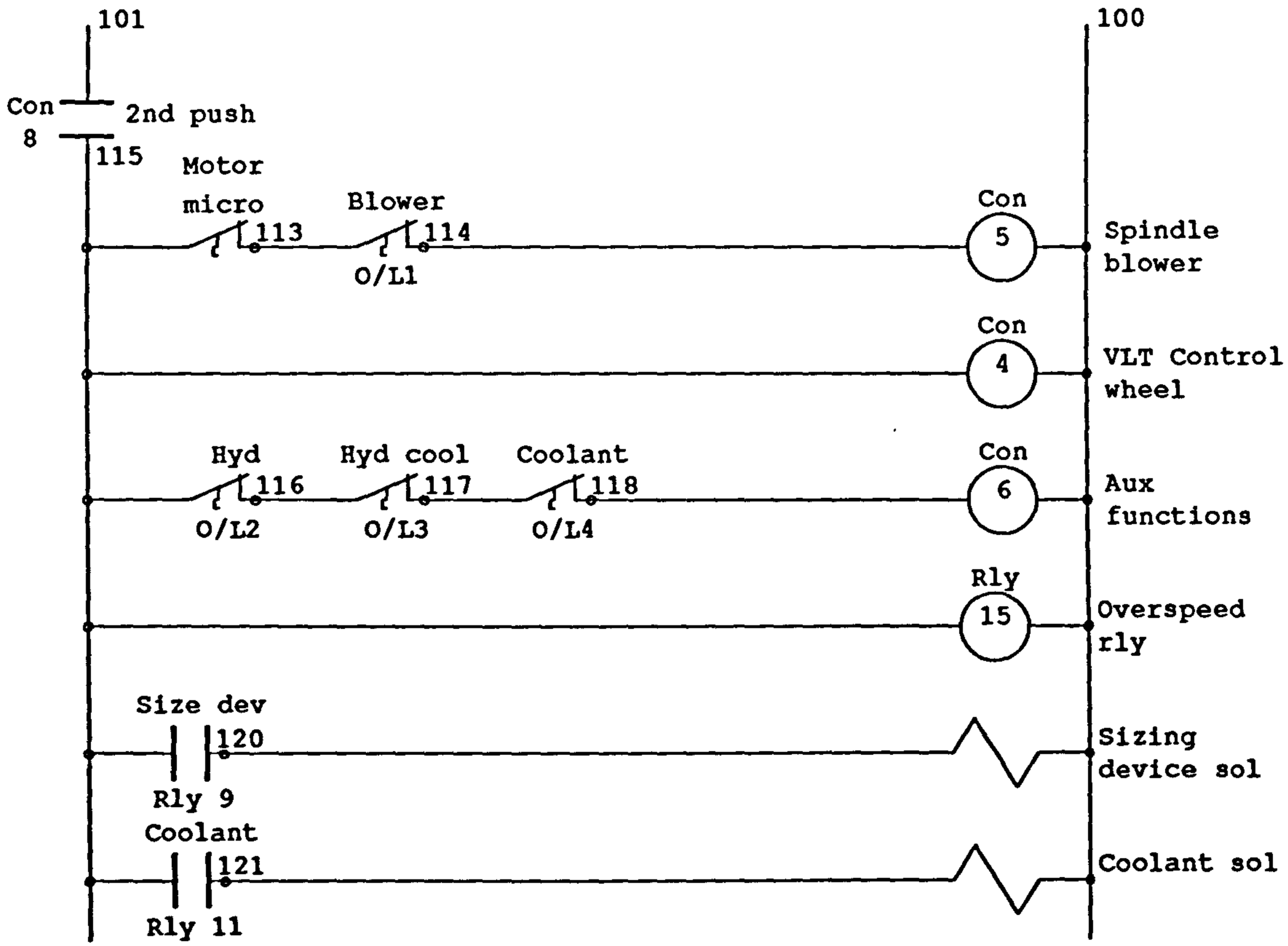

| | | |
|---|---|---|
| Cincinnati No.3 Centreless Grinding Machine | Power supply connections | Sheet 5 of 12 |



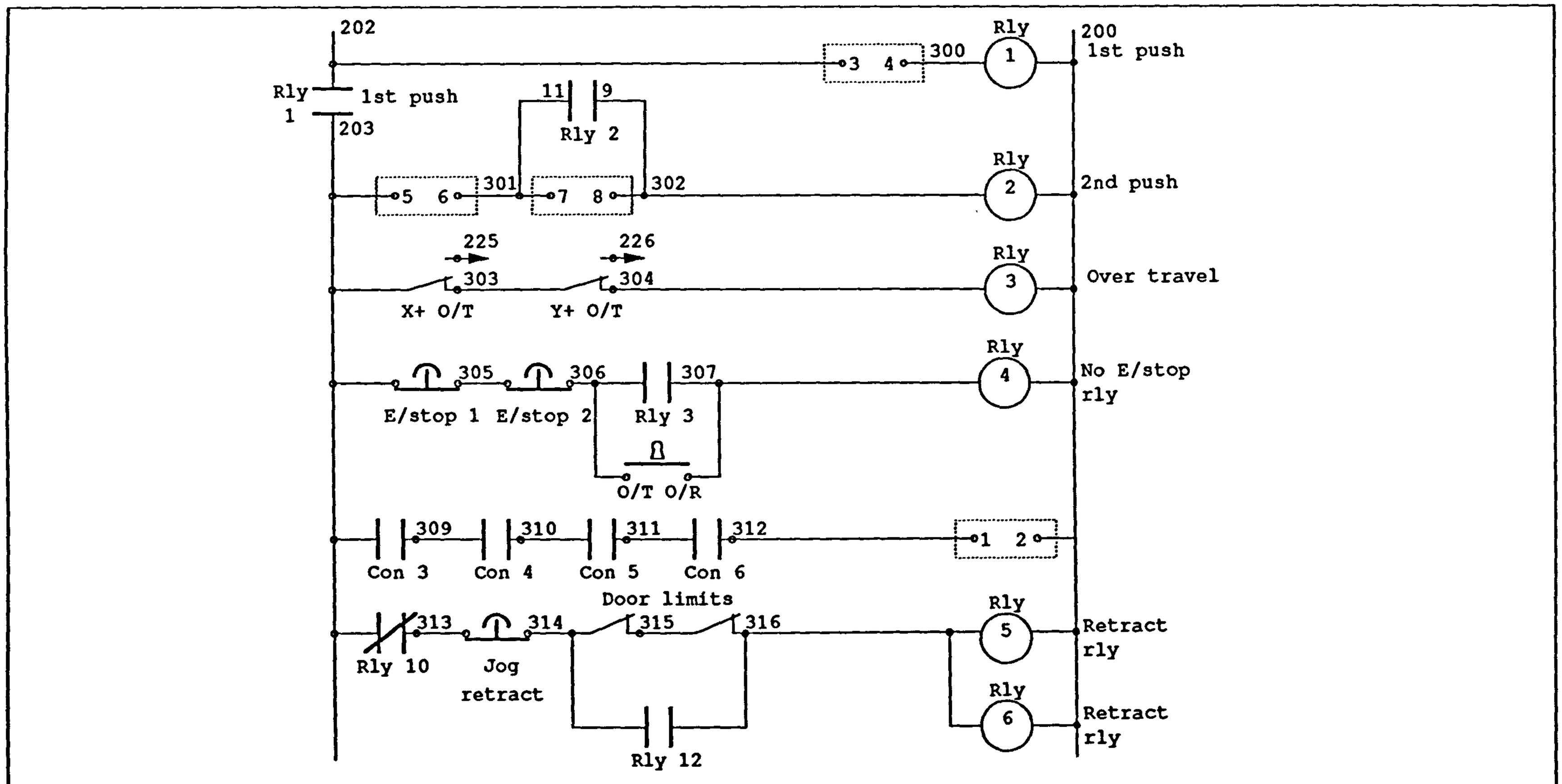

| | | |
|---|---|---|
| Cincinnati No.3 Centreless Grinding Machine | 220V connections | Sheet 6 of 12 |



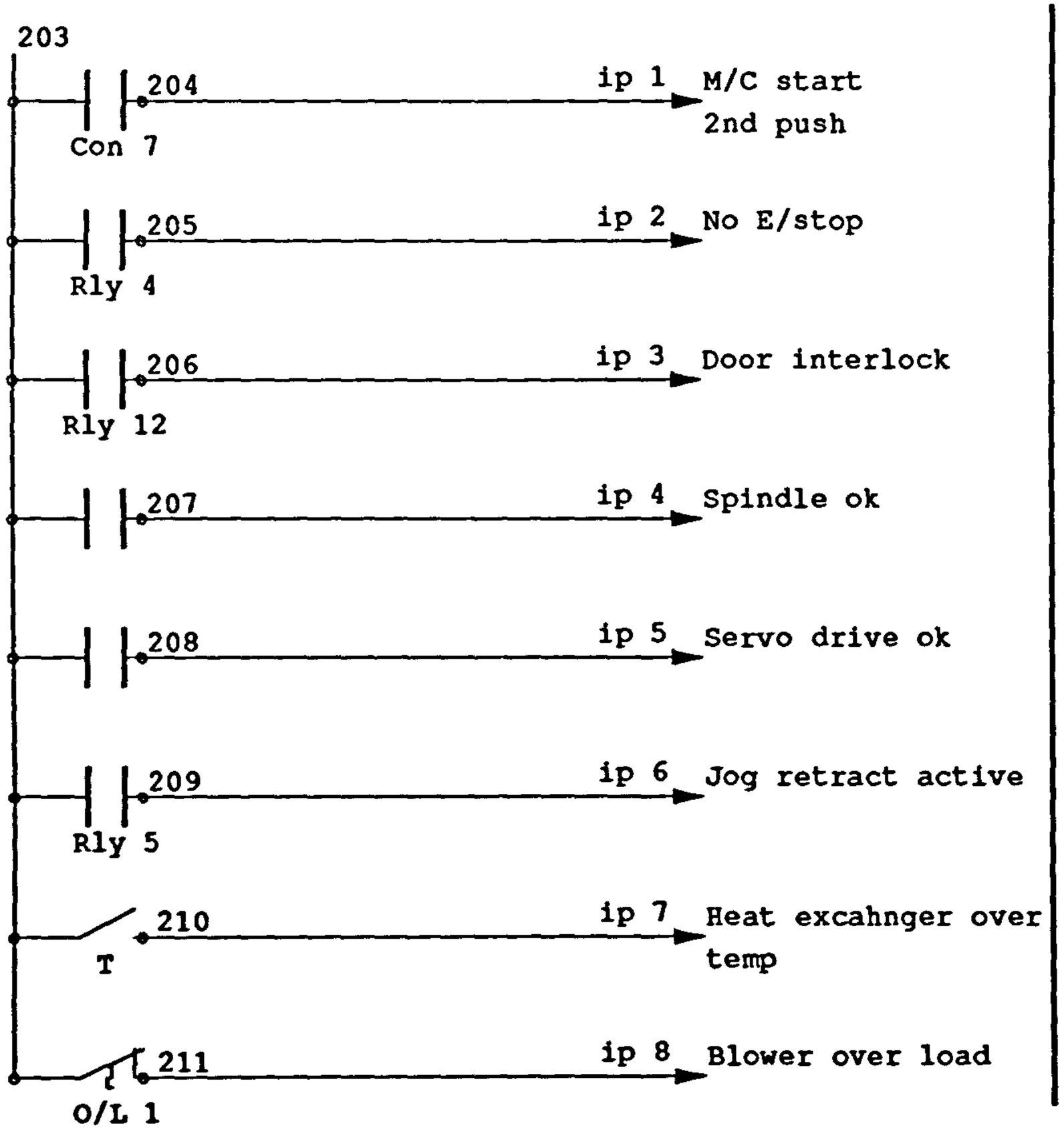

| | | |
|---|---|---|
| Cincinnati No.3 Centreless Grinding Machine | 110V connections | Sheet 7 of 12 |

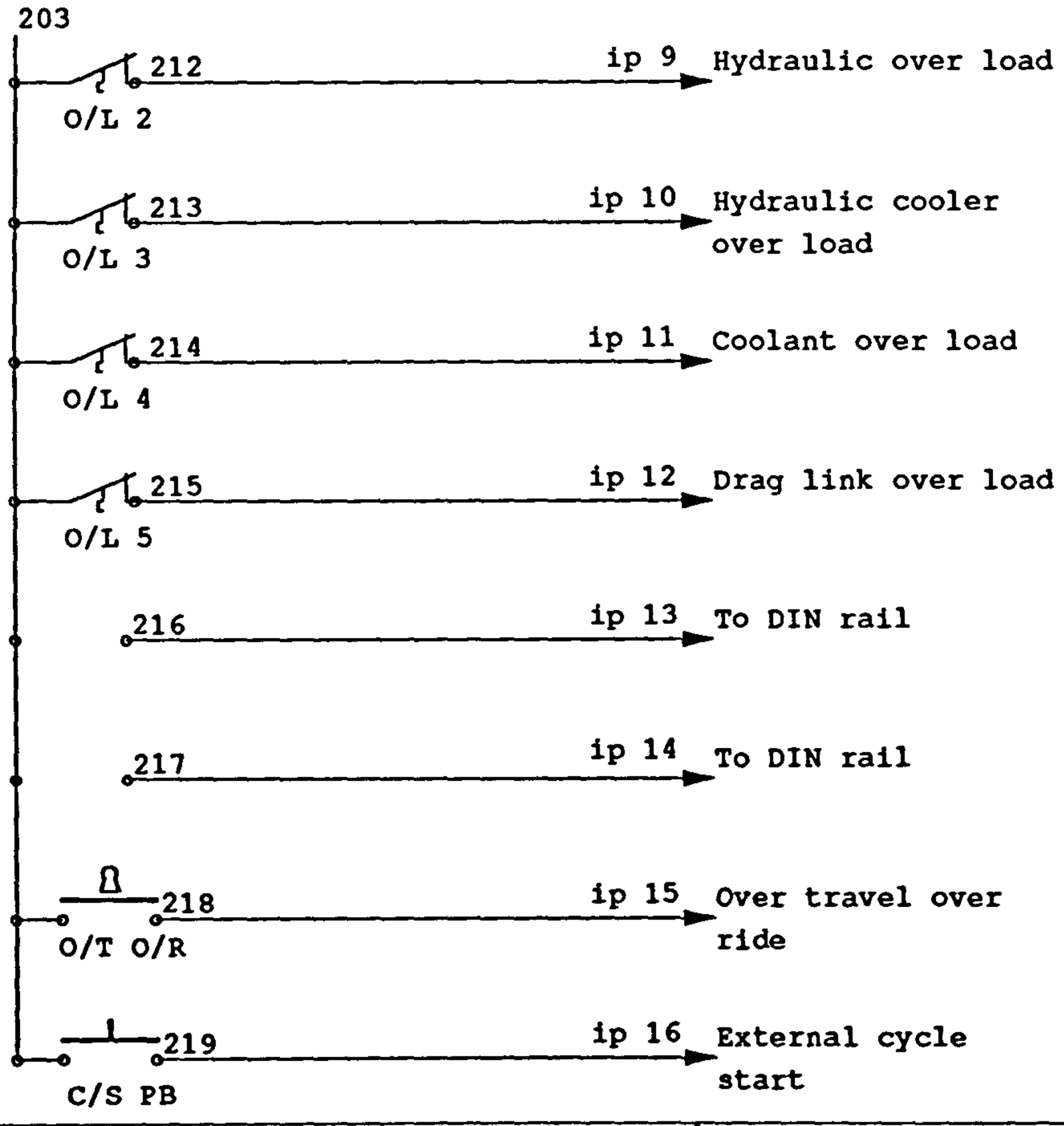



| | | |
|---|---|---|
| Cincinnati No.3 Centreless Grinding Machine | 110V connections | Sheet 8 of 12 |

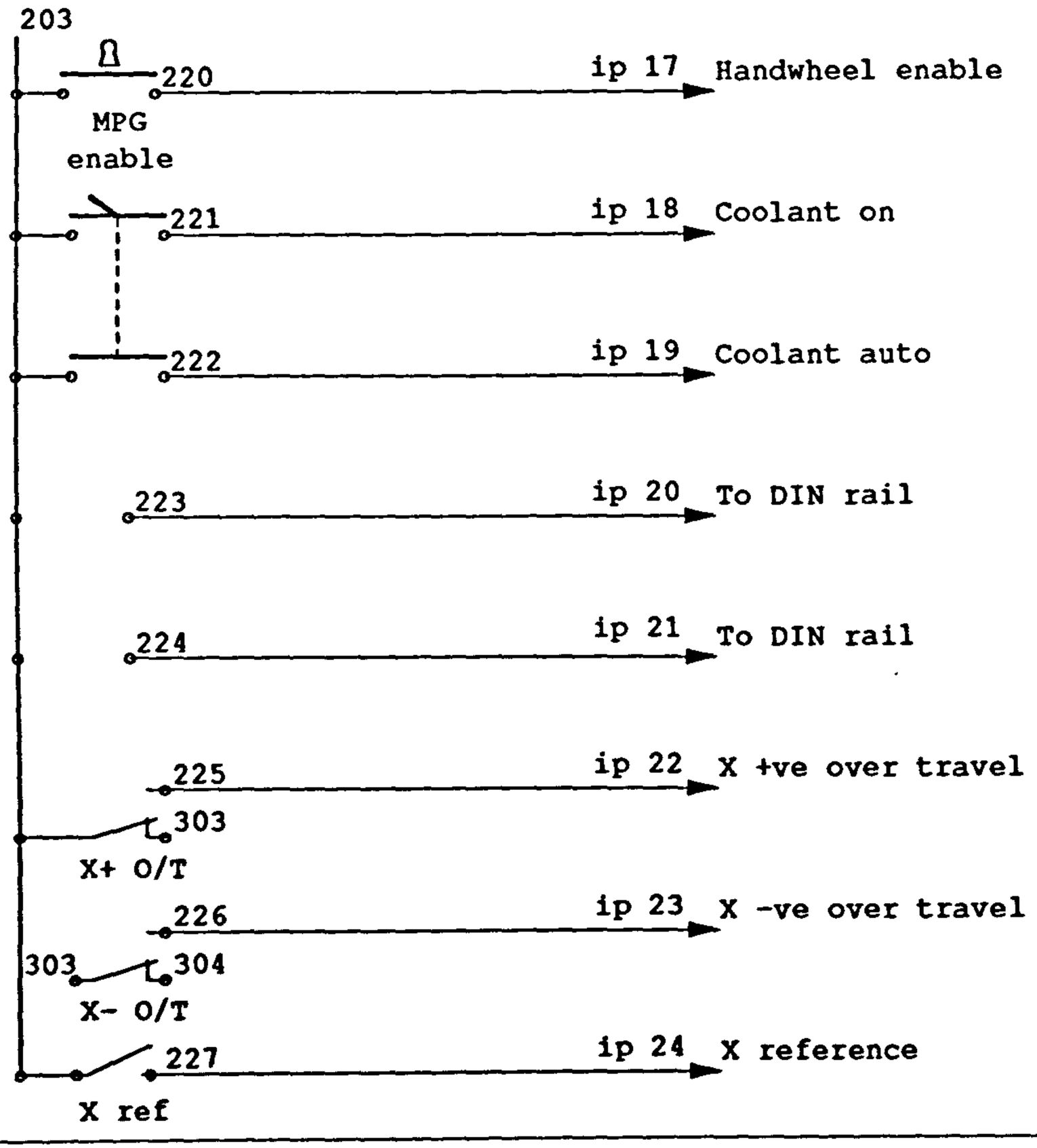


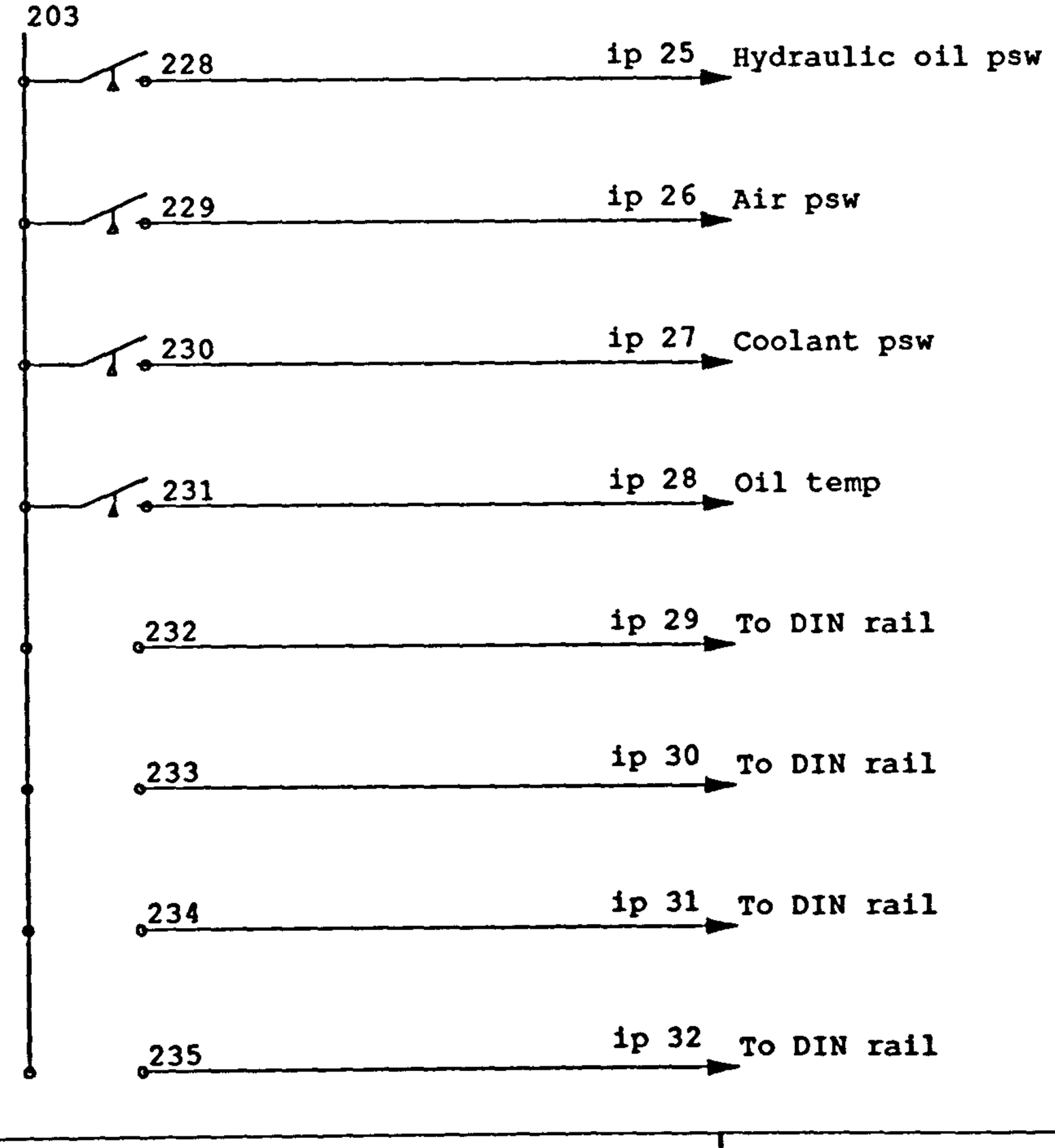

| | | |
|---|---|---|
| Cincinnati No.3 Centreless Grinding Machine | 24V connections | Sheet 9 of 12 |



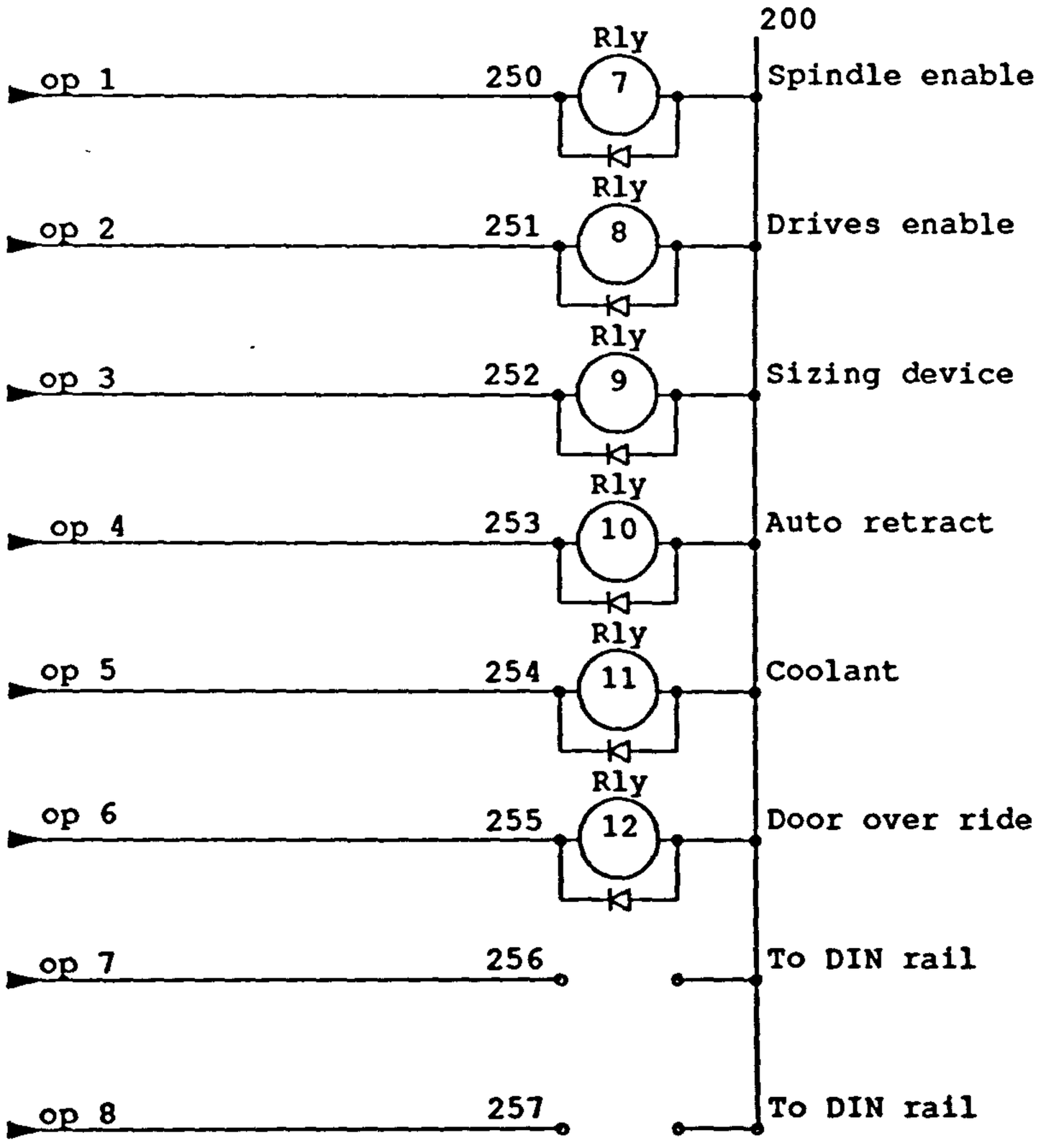



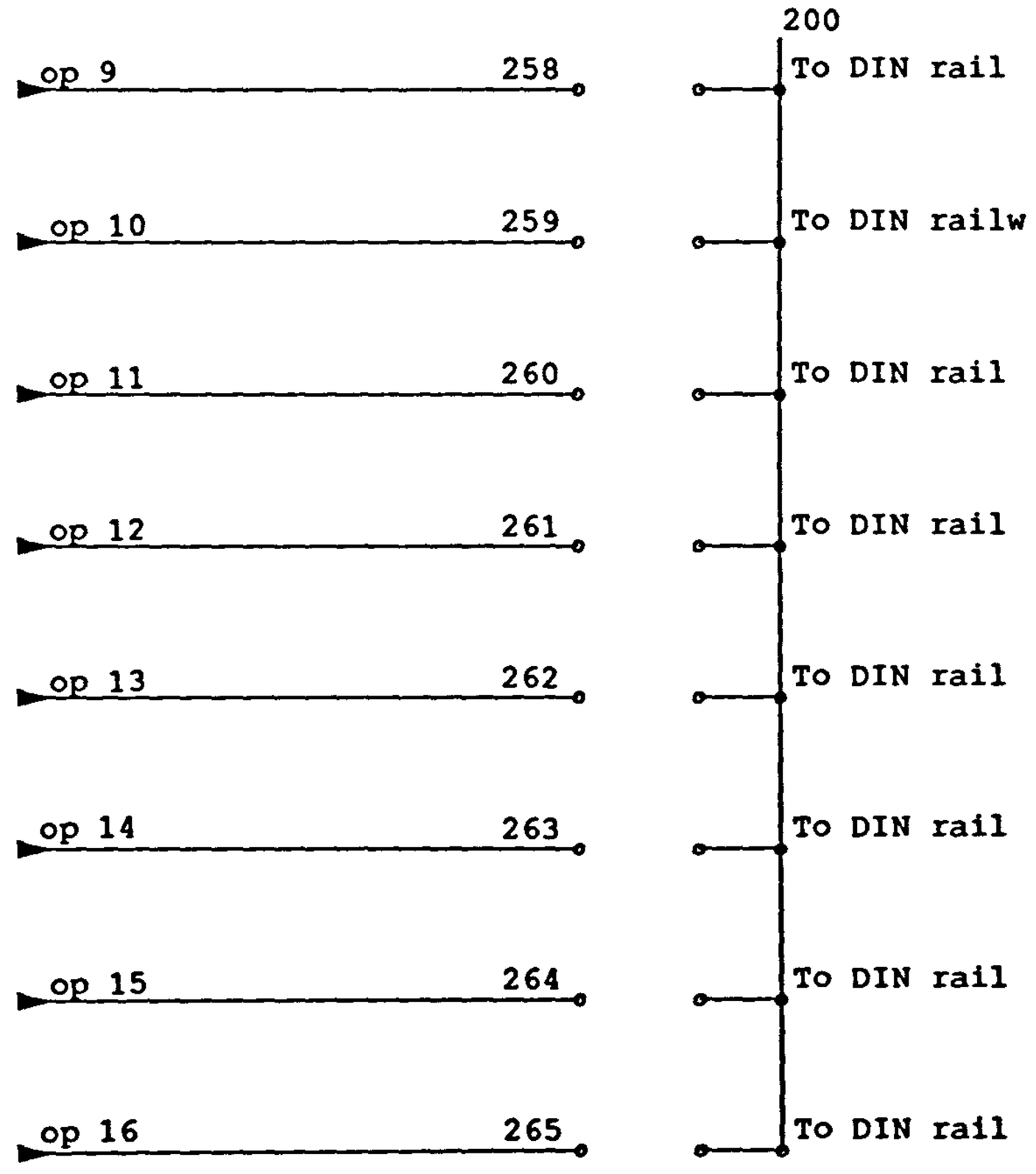

| | | |
|---|---|---|
| Cincinnati No.3 Centreless Grinding Machine | CNC input connections | Sheet 10 of 12 |




| | | |
|---|---|---|
| Cincinnati No.3 Centreless Grinding Machine | CNC input connections | Sheet 11 of 12 |




| | | |
|---|---|---|
| Cincinnati No.3 Centreless Grinding Machine | CNC output connections | Sheet 12 of 12 |

# Appendix 3. Control system software

This appendix includes listings of the software developed for the OSAI A-B 8600 MC/TC C.N.C. system in order to implement the intelligent control system described in this thesis.

## Appendix 3.1 Control system characterisation files

Characterisation files are required by the 8600 C.N.C. in order to customise operation of the system for the machine tool on which it is installed. The characterisation files listed in this appendix were used to customise the C.N.C. for use with the intelligent control system that was implemented on the automated Cincinnati No. 3 centreless grinding machine.

# PAGE NUMBERING AS ORIGINAL

# FCRSYS

## 8600 C.N.C system characterisation file

```
;FCRSYS
*1
COM=-
DSK=-
MBX=--
MPX=Y2Y2Y2--
EP1=Y8,8000,0004
EP2=Y2,7000,0002
EP3=Y2,6000,0004
EP4=Y2,E000,0004
EP5=N2,5000,0004
*2
XACONS,04
XAGRAF,0D
;XAREAD,09,1,3
XAT485,07,1,6
XAMEP2,0C,2,2EC8,02A8
;XELANP,0E,1,5
*3
XMAINP
*4
/MP2
/DY0
ISO
*6
RPL,INS,DEL
EOF,BOF,NEW,OLD
COMMAND ERROR
MEMORY OVERFLOW
RECORD OVERFLOW
WRONG KEY
KEY OVERFLOW
WRONG DIRECTION
FORMAT ERROR
I/O ERROR
*7
FILMS1,EDPERR/MP0
FILMS2,EDPMSG/MP0
FILMS3,IOERRM/MP0
FILMS4,PROERR/MP0
FILMS5,MESSAG/MP0
FORMAT,FORMAT/MP0
AXCONF,AXCFIL/MP0
;AXCONF,AXCFI2/MP0
IOCONF,IOCFIL/MP0
PGCONF,PGCFIL/MP0
FILCMD,PROTEC/MP0
*8
SIPROM
DEBUG
PIMP
CKSUM
COMVAX
*
```

# AXCFIL

## 8600 C.N.C axis characterisation file

```
;AXCFIL
;
*1
NBP=1,37FF,3800
TIM=10,10,0,0,0
PRO=1
IN1=2,XYABU,S,10,4
CAS=2,XYSABUs,10
;
*2
NAS=X
TPA=1,
NTC=1,1
RAP=405,34
PAS=500,-1
GAS=,
POS=0.002,10
SRV=0.010,0.7,1
MCZ=U135K0,0,405
MAN=405,34
GM0=405,8.5,6
;LOP=44,-4
LOP=28,-11
MFC=U135K1,U135K2
;
NAS=Y
TPA=1,
NTC=0,0
RAP=3000,500
PAS=5000,5
GAS=0,0
POS=.01,2.5
SRV=0,0,0
;MCZ=U56K1,1,100
MAN=3000,500
GM0=3000,7.5,16.667
LOP=0,0
;MFC=U54K19,U54K18
;
NAS=S
TPA=10,
NTC=0,2
PAS=400,1
GAS= ,
POS= ,
SRV=0,0,0
GM1=2081,10,10
TSM=20,17
ASM=X
POM=0,0
ADP=8,10
;
NAS=s
TPA=20,
NTC=3,3
PAS=400,1
GAS=0,0
POS=1,20
SRV=0,0,0
GM1=51,10,10
TSM=20,17
ASM=X
POM=0,0
;
NAS=A
TPA=2,
NTC=7,
PAS=500,1
SRV=,,
GAS=,
POS=,
;
NAS=B
TPA=2,
NTC=8,
PAS=500,1
SRV=,,
GAS=,
POS=,
;
NAS=U
TPA=1,
NTC=4,
PAS=500,1
SRV=,,
GAS=,
POS=,
*3
```

# PGCFIL

## 8600 C.N.C process characterisation file

```
NEW
;PGCFIL
;
*1
;TRI=BNC,BRE,
;TRI=UIO,UOI,
*2
PRO=1
;SIM=UAS,USA,,,,
SIM=E,,200,,,
SIM=NEW,PASS,60,2,127,3
SIM=NEW,D,20,6,62,3
SIM=NEW,IOERR,1,3,62,3
SIM=NEW,FINCR,1,3,62,3
SIM=NEW,COUNT,3,3,62,3
SIM=NEW,FILTR,1,6,62,3
SIM=NEW,XOFFS,1,6,62,3
*3
JCL=ORA,AXO,4
;JCL=RCM,D,
*4
ASS=UCV,1
ASS=UEP,1
ASS=RAP,1
ASS=VOL,1
NPL=20,200
NDD=MP2
PRO=1
FIL=FI1EOR/MP0,,,,FI1MOV/MP0
STR=5
SER=240
CHN=5
SCR=8064
*5
PRO=1
NIP=1
NAM=X
NPD=X,X
TOF=1
CWP=1F,0
*6
PRO=1
MAS=X
SSO=0.75,0.8,0.85,0.9,0.95,1.0,1.05,1.1,1.1
5,1.2,1.25,1.25
```

IOCFIL

8600 C.N.C I/O characterisation file

---

```
;IOCFIL
;
*1
ALM=5D00
;RUS=RTCSI2,1,
RUS=RTCSIO,2,
IN0=0,,,,,,,
OU0=1,,,,,,,
CLO=10,2
*2
PRO=1
M00=6,0,0
M01=2,4,0
M02=2,24,20
M03=45,0,21
M04=45,0,21
M05=6,0,21
M06=2,14,62
M07=45,0,44
M08=45,0,44
M09=6,0,44
M10=45,0,77
M11=6,0,77
M12=45,0,77
M13=45,0,21
M14=45,0,21
M19=45,0,21
M25=C5,0,56
M26=6,0,56
M30=82,24,0
M38=4D,0,34
M54=2,4,9
M55=45,0,8
M56=45,0,8
M60=2,4,3
M61=2,4,3
M62=2,4,3
H01=45,21
H02=45,21
*3
PRO=1
ASM=S
ADV=U
DAC=0,1,2,3,4,5
ADC=0,1,2,3,4,5
*4
```

## Appendix 3.2 Siprom interface program

The purpose of the Siprom interface program listed in this appendix was to interface the functions of the C.N.C. required for machine control with the electrical design of the automated Cincinnati No. 3 centreless grinding machine.

# GKNSIP

## Siprom machine logic interface program

```
;------------------------
; E-STOP REQUEST
;
P00=[W17K3>0]+I0K0
T14I(50)=P00+T14D*/T14U
U10K7=T14D
U140K30=T14D
;
;------------------------
; E-STOP REQUEST SYSTEM ERRORS
;
U130K0=/I0A1*/I0A21*/I0A22*I0A5*I0A0*U133K0
U130K1=U138K0*U1A0*/I0A3*U133K0
U130K2=U136K0*U1A1*/I0A4*U133K0
U130K3=/I0A5*/U1A3*I0A0*U133K0
U130K4=/I0A5*U1A3*I0A0*U133K0
U130K5=I0A7*I0A0*U133K0
U130K6=I0A8*I0A0*U133K0
U130K7=I0A9*I0A0*U133K0
U130K8=I0A10*I0A0*U133K0
U130K9=I0A11*I0A0*U133K0
U130K10=I0A21*/I0A14*I0A0*U133K0
U130K11=I0A22*/I0A14*I0A0*U133K0
U130K12=/I0A24*I0A0*U133K0
;
;------------------------
; INVERT INPUTS
U135K0=/I0A23
U135K1=/I0A21
U135K2=/I0A22
;
;------------------------
; ENABLE COMU AND CEFA
DOF: U133K0*I0A0*/U10K0
  U10K24=I0A0*U133K0
  U10K25=I0A0*U133K0
  U10K26=I0A0*U133K0
  U10K9=I0A0*U133K0
  U10K10=I0A0*U133K0
ENDF
;
DOF: /I0A0
  W10K1=0
  W10K3=0
ENDF
;
;------------------------
; FORCE PROCESS 1 AND DISPLAY 1
P03=I0A0
;
;------------------------
; END OF PROGRAM
; M30
DOF: [W3K0=30H]*I04K18*/I0K1*U133K0
;  U10K1=[1=1]
ENDF
;
;------------------------
; ENABLE HANDWHEEL PLUS RESET
;
DOF: I0A16*U133K0*/U139K0
  W15K0=255
  W15K1=8
  W15K2=1
  W15K3=255
  U139K0=[1=1]
ENDF
;
;------------------------
; SELECT DISPLAY NO. 1
T07I(10)=U10K1+P03+T07D+[W3K0=30H]*I04K18+U
139K0+P00
U16K24=T07D
U16K28=T07D
;
P04=U139K0+U133K0
U10K1=P04
;
;------------------------
; AUTO RETRACT
DOF: U134K0*U133K0+T12D
  U134K0=/T12U
  T12I(10)=/T12U
  U1A3=T12D
ENDF
;
$
;
;------------------------
; DISABLE HANDWHEEL
DOF:/I0A16*U139K0
  W15K0=0
  W15K1=0
  W15K2=0
  W15K3=0
  U139K0=[1=2]
  U10K8=[1=2]
  U136K0=[1=2]
  U136K1=[1=2]
  U1A1=[1=2]
ENDF
;
;------------------------
; 1 SECOND ON/OFF TIMER
T03I(10)=(T03D+/T04D)*/T03U
T04I(10)=(T04D+/T03D)*/T04U
;
;------------------------
; PANEL MONITOR
U140K26=I0A14*/I0A21*/I0A22+I0A2
U140K27=I0A14*(I0A21+I0A22)
U140K28=I0A15
```

```
U140K29=I0A16
;
;-------------------------
; DOOR OVER-RIDE
U1A5=I0A14*/I0A21*/I0A22
;
;-------------------------
; SIZING DEVICE TRIP INT. TO PP
U1A2=[W60K2=1]
;
;-------------------------
; POWER UP SEQUENCER
;
; M/C TOOL OFF
;-------------------------
; MACHINE TOOL OFF
T11I(1)=/I0A0
U10K0=T11U
;
U140K24=U10K0*/T14D
U140K25=I0A0*/U133K0
;
;-------------------------
; EXTRA RUN-TIME MESSAGES
U140K0=/I0A1*/I0A21*/I0A22*/I0A0*I0A5
U140K3=/I0A5*/U1A3*/I0A0
U140K4=/I0A5*U1A3*/I0A0
U140K5=I0A7*/I0A0
U140K6=I0A8*/I0A0
U140K7=I0A9*/I0A0
U140K8=I0A10*/I0A0
U140K9=I0A11*/I0A0
U140K10=I0A21*/I0A0
U140K11=I0A22*/I0A0
;
;-------------------------
; M/C TOOL POWERING UP
T10I(2)=I0A0
;
;-------------------------
```

```
; START SEQUENCE
DOF:T10D
  W1A0=0
  W17K3=0
  W60K0=0
  U10K3=[1=2]
  W146K0=0
  W146K1=0
  W140K0=0
  W140K1=0
  W140K2=0
  W140K3=0
  W141K0=0
  W131K3=1
  W150K0=0
ENDF
;
;-------------------------
; CHECK E-STOPS AND OT'S
DOF:T10U
;-------------------------
; CHECK INTERLOCKS
  DOF: [W131K3=1]
    U140K0=/I0A1*/I0A21*/I0A22+U140K0
    U140K3=/I0A5*/U1A3+U140K3
    U140K4=/I0A5*U1A3+U140K4
    U140K10=I0A21*/I0A14+U140K10
    U140K11=I0A22*/I0A14+U140K11

W131K3=MUX(0,0,0,0,0,2),(U140K0,U140K3,U140
K4,U140K10,U140K11,U131K24)
  ENDF
  DOF: [W131K3=2]
    U140K5=I0A7+U140K5
    U140K6=I0A8+U140K6
    U140K7=I0A9+U140K7
    U140K8=I0A10+U140K8
    U140K9=I0A11+U140K9

W131K3=MUX(0,0,0,0,0,4),(U140K5,U140K6,U140
K7,U140K8,U140K9,U131K25)
  ENDF
;-------------------------
; WAIT FOR OIL
  DOF: [W131K3=4]
    C00Z=[1=2]
    C00A=[1=2]
    C00I(255)=T03U
    U140K13=/I0A24*/C00R
    U140K14=/I0A24*C00R
    C00Z=C00R+I0A24

W131K3=MUX(0,4,5),(U140K14,U140K13,I0A24)
  ENDF
;
;-------------------------
; INIT CSI
  DOF: [W131K3=5]
    C00Z=[1=2]
    C00A=[1=2]
    C00I(5)=T03U
    U150K0=[1=1]
    U140K15=/[W150K0=255]*/C00R
    U140K16=/[W150K0=255]*C00R
    C00Z=C00R+U150K7

W131K3=MUX(0,5,6),(U140K16,U140K15,U150K7)
  ENDF
;
;-------------------------
; POWER UP SEQUENCE COMPLETE FLAG
U133K0=[W131K3=6]
;
;-------------------------
; E-STOP DISPLAY MESSAGES
  DOF: [W140K0>0]
```

```
W17K3=MUX(41,44,45,46,47,48),(U140K0,U140K3
,U140K4,U140K5,U140K6,U140K7)
  ENDF
  DOE: [W140K1>0]

W17K3=MUX(49,50,51,52,53),(U140K8,U140K9,U1
40K10,U140K11,U140K14)
  ENDE
  DOE: [W140K2>0]
    W17K3=MUX(54),(U140K16)
  ENDE
ENDF
;
;-------------------------
; E-STOP FAULT CATCHER
DOF: [W146K0=0]*[W146K1=0]

W146K0=MUX(1,2,4,8,16,32,64,128),(U130K0,U1
30K1,U130K2,U130K3,U130K4,U130K5,U130K6,U13
0K7)

W146K1=MUX(1,2,4,8,16),(U130K8,U130K9,U130K
10,U130K11,U130K12)
ENDF
;
;-------------------------
; E-STOP ERROR MESSAGES
DOF: [W146K0>0]

W17K3=MUX(41,42,43,44,45,46,47,48),(U146K0,
U146K1,U146K2,U146K3,U146K4,U146K5,U146K6,U
146K7)
ENDF
;
DOE: [W146K1>0]

W17K3=MUX(49,50,51,52,53),(U146K8,U146K9,U1
46K10,U146K11,U146K12)
ENDE
;
;-------------------------
; CLEAR OUTPUTS ON E-STOP
DOF: T11U
  W1A0=8*W1A0
  W131K0=0
  W131K3=0
  W133K0=0
  W137K0=0
  W138K0=0
  W17K0=0
  W10K2=0
  C00Z=[1=1]
ENDF
;
;-------------------------
; MESSAGE RECIRCULATION ROUTINE
;
DOF:
[W140K0=0]*[W140K1=0]*[W140K2=0]*[W140K3=0]
*[W141K0=0]
  W21K0=0
  W21K1=0
  W21K2=0
  W21K3=0
  W22K0=0
  W142K2=0
ENDF
;
;-------------------------
; START ROUTINE:
; SET W142K3 IF MESSAGE COUNTER PICKS
; OUT AN ERROR MESSAGE
;
DOF: /[W140K0=0]*[W142K0<9]
  W142K1=W142K0
  W142K2=DEC(W142K1)
  W142K3=W140K0*W142K2
  W143K0=1
ENDF
;
DOE: /[W140K1=0]*[W142K0<17]
  W142K1=[W142K0-8]
  W142K2=DEC(W142K1)
  W142K3=W140K1*W142K2
  W143K0=2
ENDE
;
DOE: /[W140K2=0]*[W142K0<25]
  W142K1=[W142K0-16]
  W142K2=DEC(W142K1)
  W142K3=W140K2*W142K2
  W143K0=3
ENDE
;
DOE: /[W140K3=0]*[W142K0<33]
  W142K1=[W142K0-24]
  W142K2=DEC(W142K1)
  W142K3=W140K3*W142K2
  W143K0=4
ENDE
;
DOE: /[W141K0=0]*[W142K0<41]
  W142K1=[W142K0-32]
  W142K2=DEC(W142K1)
  W142K3=W141K0*W142K2
  W143K0=5
ENDE
;
DOE:
  W142K3=0
ENDE
;
;-------------------------
; ONE SHOT: SET U143K10
; IF MESSAGE DETECTED
DOF:/[W142K3=0]*/U143K10
  U143K10=[1=1]
  W142K3=0
ENDF
```

```
;
;-------------------------
; DISPLAY MESSAGE NUMBER
DOF: U143K10*T00U
  W21K0=0
  W21K1=0
  W21K2=0
  W21K3=0
  W22K0=0
  DOF: [W143K0=1]
    W21K0=W142K2
  ENDF
  DOE: [W143K0=2]
    W21K1=W142K2
  ENDE
  DOE: [W143K0=3]
    W21K2=W142K2
  ENDE
  DOE: [W143K0=4]
    W21K3=W142K2
  ENDE
  DOE: [W143K0=5]
    W22K0=W142K2
  ENDE
U143K10=[1=2]
ENDF
;
;-------------------------
; BUMP COUNTER IF NO ERROR
DOF: /U143K10
  DOF: [W140K0=0]*([W142K0=0]+[W142K0=1])
    W142K0=8
  ENDF
  DOE: [W140K1=0]*[W142K0=9]
    W142K0=16
  ENDE
  DOE: [W140K2=0]*[W142K0=17]
    W142K0=24
  ENDE
  DOE: [W140K3=0]*[W142K0=25]
    W142K0=32
  ENDE
  DOE: [W141K0=0]*[W142K0=33]
    W142K0=40
  ENDE
  DOF: [W142K0>40]
    W142K0=0
  ENDF
  W142K0=[W142K0+1]
ENDF
;
T00I(10)=/T00U
;
;-------------------------
; INITIALISE PART-PROGRAM
T05I(10)=U133K0
;
DOF:T05U*U133K0
W17K0=3
W10K2=132
ENDF
;
;-------------------------
; SPINDLE ENABLE/DISABLE
; M03
DOF: [W3K0=3]*I4K18*U133K0+T01D
W11K0=8
W11K2=1
U1A0=[1=1]
T01I(10)=/T01U
U138K0=T01U
ENDF
;
; M05
DOF: [W3K0=5]*I4K18*U133K0+/U133K0
W11K0=0
W11K2=0
U1A0=[1=2]
U138K0=[1=2]
ENDF
;
;-------------------------
; SERVO DISABLE/ENABLE
; M10
DOF:([W3K0=10H]*I4K18)*U133K0+/U133K0
U1A1=[1=2]
U10K8=[1=2]
U136K0=[1=2]
U136K1=[1=2]
ENDF
;
; M11
DOF:(([W3K0=11H]*I4K18)+T02D+(U139K0+U139K1
)*/U136K1)*U133K0
U1A1=[1=1]
U10K8=[1=1]
T02I(2)=/T02U
U136K0=T02U
U136K1=[1=1]
ENDF
;
;-------------------------
; ENABLE SERVO AMP FROM CONSOLE
DOF:
([W8K3=64]+[W8K3=8]+[W8K3=16]+T06D)*U133K0*
/U136K1
U1A1=[1=1]
U10K8=[1=1]
T06I(10)=/T06U
U136K0=T06U
U136K2=[1=1]
ENDF
;
DOF:
/[W8K3=64]*/[W8K3=8]*/[W8K3=16]*U136K2*/U13
6K1+/U133K0
U1A1=[1=2]
U10K8=[1=2]
U136K0=[1=2]
U136K2=[1=2]
```

```
ENDF
;
;-------------------------
; COOLANT ON/OFF
; M08
DOF:[W3K0=08H]*I4K18*U133K0
  U137K0=[1=1]
ENDF
; M09
DOF:[W3K0=09H]*I4K18*U133K0
  U137K0=[1=2]
ENDF
;
U1A4=U133K0*I0A3*U137K0*I0A18+I0A17*U133K0
;
;-------------------------
; PART-PROGRAM GOODIES
; Quit from GRIN
DOF: "Q"
  W60K0=2
  U10K3=/U60K1
  U10K4=U60K1
ENDF
;
;-------------------------
; PP INTERRUPT
DOF: "I"
  W156K3=255
ENDF
;
;-------------------------
; CYCLE STOP ON SK240=1
DOF: U60K0
  U10K3=U60K0
  U10K4=/U60K0
ENDF
;
;-------------------------
; CYCLE START ON I0A15
DOF: I0A15*U60K0
  W61K0=0
  W60K0=0
  U10K3=/I0A15
  U10K4=I0A15
ENDF
;
;-------------------------
; PUT PROCESS INTO JOG SK200=1
DOF: [W50K0=1]*U133K0
  U15K2=[1=1]
  U15K3=[1=1]
  W15K1=8
  W15K2=1
  W15K3=255
  W50K0=255
  W10K0=0
  U139K1=[1=1]
ENDF
;
;-------------------------
; TAKE OUT OF JOG ON EXT C-S OR RESET
DOF:[W50K0=255]*I0A15*U133K0+U10K1
  W15K1=0
  W15K0=0
  W15K2=0
  W50K0=0
  U10K4=[1=1]
  U1A1=[1=2]
  U139K1=[1=2]
  U136K0=[1=2]
  U136K1=[1=2]
ENDF
;
;-------------------------
; REAL TIME MONITOR
U145K0="I00"+[W145K0=1]
U145K1="I02"+[W145K0=2]
U145K8="U00"+[W145K1=1]
U145K9="U02"+[W145K1=2]
U145K16="CAN"
;
DOF: [W145K0>0]
  W23K0=[49H]
  W23K1=MUX(W0A0,W0A2),(U145K0,U145K1)
  W23K2=MUX(W0A1,W0A3),(U145K0,U145K1)
ENDF
;
DOF: [W145K1>0]
  W23K3=[55H]
  W24K0=MUX(W1A0,W1A2),(U145K8,U145K9)
  W24K1=MUX(W1A1,W1A3),(U145K8,U145K9)
ENDF
;
DOF: [W145K2>0]
  W23K0=0
  W23K1=0
  W23K2=0
  W23K3=0
  W24K0=0
  W24K1=0
ENDF
;
;-------------------------
; AUTO CALIBRATE PROBES
;
DOF: ("C"+[W132K0=1])*U133K0
  W60K2=1
  T15I(30)=/T15U
  W150K2=0
  W132K0=MUX(2,1),(T15U,U60K16)
ENDF
;
DOF: [W132K0=2]*U133K0
  T16I(10)=/T16U
  U150K18=[1=1]
  W132K0=MUX(4,2),(T16U,U60K16)
ENDF
;
DOF: [W132K0=4]*U133K0
  W60K2=0
```

```
  W132K0=0
  T15I(0)=[1=2]
  T16I(0)=[1=2]
ENDF
;
DOF: ">"
  W60K2=1
ENDF
DOF: "<"
  W60K2=0
ENDF
```

# MESSAG

## Siprom machine logic message file

```
E-STOP BUTTON PRESSED
GRINDING WHEEL DRIVE FAULT
INFEED DRIVE FAULT
RAPID RETRACT -BUTTON OR DOOR
RAPID RETRACT -SOFTWARE
BLOWER CON O/L
HYD CON O/L
HYD COOL CON O/L
COOLANT CON O/L
DRAG LINK CON O/L
+X AXIS OVER-TRAVEL
-X AXIS OVER-TRAVEL
LOW OIL PRESSURE
WAITING FOR OIL PRESSURE
OIL PRESSURE TIME OUT
WAITING FOR CSI INIT
CSI INIT TIME OUT
18
19
20
21
22
23
24
M/C TOOL OFF
M/C TOOL POWERING UP
DOOR INTERLOCK OVER-RIDE
OVER-TRAVEL OVER-RIDE
EXTERNAL CYCLE START PRESSED
HANDWHEEL KEY SWITCH ENABLED
M/C TOOL POWERING DOWN
32
FILE HANDLING ERROR
OVER SHOOT CALC ERROR
35
36
37
38
39
40
E-STOP: BUTTON PRESSED
E-STOP: GRIND WHEEL DRIVE FAULT
E-STOP: INFEED DRIVE FAULT
E-STOP: RAPID RETRACT -BUTTON
E-STOP: RAPID RETRACT -SOFTWARE
E-STOP: BLOWER CON O/L
E-STOP: HYD CON O/L
E-STOP: HYD COOL CON O/L
E-STOP: COOLANT CON O/L
E-STOP: DRAG LINK CON O/L
E-STOP: +X AXIS OVER-TRAVEL
E-STOP: -X AXIS OVER-TRAVEL
E-STOP: LOW OIL PRESSURE
E-STOP: CSI FAILURE
```

## Appendix 3.3 CSI routines

The CSI modules listed in this appendix were created off-line on an IBM PC compatible computer using the Intel high level language PL/M-86 and assembler ASM-86. CSI programming was used in the intelligent control system for data base management, file handling, to provide custom screen displays and to perform complex mathematical operations whilst communicating with software at other levels in the system.

# MAIN.PLM

## Master CSI module

```
CSI_TSK: DO;

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

Main CSI driving routine for :

          SLOW_CSI: slow CSI routine

          FAST_CSI: fast CSI routine

          CSI_TASK: semafore activated file handling

V1.0:     Original version as delivered to GKN
          -Sean Kelly Feb 1988

V1.1      CSI_TASK Module. Task priority should be between
150/200
          Priority changed to 150 accordingly. 80287 declared.
          -Sean Kelly 25th Oct 1988.

          DISP1_PROG Module modified to warn part-program of
file
          initialisations through SK(636).
          -Sean Kelly 4th Nov 1988.

          CSI_INIT Routine. Current screen No. now detected
from
          K buffer i.e. SK25 lower nibble.
          -Sean Kelly 2nd Dec 1988.

          DISP1_PROG Module modified + all other modules so
          that BED_LENGTH_OFFSET now resides in MCDAT1, not
          MCDAT2. Check DISP1 file creation, reading, editing
          routines if problems (MCDAT1/MP, MCDAT2/MP).
          -Sean Kelly 2nd Dec 1988.

          FAST_CSI Routine modified to flag wheels at rated
          speed to Siprom and part-program through SK552.
                    U138K1 = 1 - Gdg wheel at speed.
                    U138K2 = 1 - Ctrl wheel at speed.
          -Sean Kelly 17th Jan 1989.

          Initialisations modified to include pointer to dual
buffer
          containing programmed Vs speed. BUFFER_INDEX toggles
between
          the two buffers. S_FUNC_PRES = 1 indicates that S
has been
          programmed -cancelled by M05. Used in DISP2.PLM and
to flag
          SK552 when at programmed speed.
          -Sean Kelly 23rd Jan 1989.

          FAST_CSI Routine modified to provide software jog
retract
          feature. SK536-1 if power exceeds 65kW. Siprom
manages the
          retraction and resets SK536.
          -Sean Kelly 6th Feb 1989.

V1.2      CALC_TOR AND CALC_LEARNED_OS in CALC.PLM
          Routines created to learn Tor
          from required and actual material removal then calc
overshoot
          based on new Tor. Must be run sequentially as global
variables
          XT and TOR used.
          -Sean Kelly 6th March 1989

          MCDAT1/MP File modified to include material
allowance parameter.
          P-P uses the value to produce cmpts. above or below
the calibrated
          size reading.
          -Sean Kelly 7th March 1989

          CSI Task module. If first peck flag set then filter
Tor and save.
          -Sean Kelly 8th March 1989

V1.3      Slow CSI modified so that SK(638) = vf current /
rapid traverse.
          Used for setting feed rate over-ride in Siprom if
gap detector
          in use. SK(608) = 32 triggers calculation.
          SK(608) = 0FFH Calc finished OK
          SK(608) = 00H feed rate over-ride outside range 0-
125%
          -Sean Kelly 24th May 1989

          DISP.PLM modified to include CYCTAB/MP0 cycle
definition table.
          Much the same as the rest of the rest of file
handlers.
          GKNIO2 modified accordingly.
          -Sean Kelly 14th June 1989

          CALC.PLM modified: CALC_OS to use CYCLE(INDEX).TOR
-not PART.TOR
          Where INDEX is the peck number from E119 set by part
program.
          -Sean Kelly 15th June 1989

          Further modification to CALC.PLM module. Learned
CYCLE(INDEX).TOR
          values -stored in CYCTAB/MP0, to represent
normalised/average
          component. After the first peck the true component
TOR is used to
          determine the TOR_ratio: true TOR/average TOR.
Subsequent overshoot
          calcs use CYCLE(INDEX).TOR * TOR_ratio. Ratio
updated after every
          peck.
          -Sean Kelly 23rd June 1989

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

$INCLUDE(MATHS.PLM)
$INCLUDE(GRAPHICS.PLM)

DECLARE MAIN_ROUTINE STRUCTURE (
          FIRST (12) BYTE,
          PTR_ROUTINE POINTER,
          SECOND (6) BYTE)
DATA      ('/64M55RTCSIO',
          @CSI_INIT,
          '55M46\');
/*-----------------------------------------------*/

DECLARE SLOW_CSI_PTR POINTER AT (404);   /*TRAP 101 FOR CSI
SLOW*/
DECLARE FAST_CSI_PTR POINTER AT (400);   /*TRAP 100 FOR CSI
FAST*/

DECLARE ENC_1 DWORD AT (10800H);
DECLARE ENC_2 DWORD AT (10800H+4);       /*ENCODER AREA*/
DECLARE ENC_3 DWORD AT (10800H+8);
DECLARE ENC_4 DWORD AT (10800H+12);

DECLARE DTD (2048) BYTE PUBLIC;                    /*DYNAMIC
TASK DESCRIPTOR*/

DECLARE SEM (20) BYTE PUBLIC;            /*SEMAPHORE
ACTIVATION*/

DECLARE SEMA LITERALLY '01H';
DECLARE START LITERALLY '01H';                     /*A FEW
LITERAL DECLARES*/
DECLARE NOT_START LITERALLY '0FFH';
/*-----------------------------------------------*/

DECLARE PROC_1_PTR POINTER PUBLIC;

DECLARE SYM_TAB_PTR POINTER PUBLIC;
DECLARE K_BUFF_PTR POINTER PUBLIC;
DECLARE TIM_TAB_PTR POINTER PUBLIC;
DECLARE PROC_OUT_PTR POINTER PUBLIC;               /*PUBLIC
POINTERS */
DECLARE PROG_VS_SPEED_PTR POINTER PUBLIC;          /*PROC_OUT
BUFFER NO. 1*/
DECLARE S_FUNC_PRES_PTR POINTER PUBLIC;            /*POINTER
TO INDICATE S PROG*/
DECLARE BUFFER_INDEX_PTR POINTER PUBLIC;           /*POINTER
TO BUFFER NO.*/
DECLARE INTERP_TABLE_PTR POINTER PUBLIC;
DECLARE PROFILE_SPEED_PTR POINTER PUBLIC;
DECLARE PROC_AXIS_CONT_PTR POINTER PUBLIC;
DECLARE END_POINT_PTR POINTER PUBLIC;
DECLARE TTC_CON_PTR POINTER PUBLIC;
DECLARE FR_POT_PTR POINTER PUBLIC;
DECLARE VIDEO_NO_PTR POINTER PUBLIC;
DECLARE AXIS_TAB_PTR POINTER PUBLIC;
DECLARE AXIS_DES_PTR POINTER PUBLIC;

DECLARE E_PAR_PTR POINTER PUBLIC;
DECLARE SYVAR_PTR POINTER PUBLIC;
DECLARE PASS_PTR POINTER PUBLIC;

/*-----------------------------------------------
          DATA AREAS INITIALISED BY CSI_INIT
-----------------------------------------------*/
```

```
DECLARE TRAP200 POINTER AT (320H);
        /*TRAP200*/

DECLARE SYS_TAB BASED TRAP200 (*) POINTER;
        /*TABLE 1*/
DECLARE PROC_1_TAB BASED PROC_1_PTR (*) POINTER;    /*TABLE 7*/
DECLARE SYM_TAB BASED SYM_TAB_PTR (*) BYTE;
DECLARE SK BASED K_BUFF_PTR (*) BYTE;
        /*K BUFFER*/
DECLARE SYS_CLK BASED TIM_TAB_PTR INTEGER;
DECLARE PROC_OUT BASED PROC_OUT_PTR (*) BYTE;
DECLARE INTERP_TABLE BASED INTERP_TABLE_PTR (*) BYTE;
DECLARE PROFILE_SPEED BASED PROFILE_SPEED_PTR REAL;
DECLARE PROC_AXIS_TAB BASED PROC_AXIS_CONT_PTR (*) BYTE;
DECLARE END_POINT BASED END_POINT_PTR (8) BYTE;
DECLARE PROG_VS_SPEED BASED PROG_VS_SPEED_PTR REAL;
DECLARE S_FUNC_PRES BASED S_FUNC_PRES_PTR BYTE;
DECLARE BUFFER_INDEX BASED BUFFER_INDEX_PTR BYTE;
DECLARE TTC_CON_TAB BASED TTC_CON_PTR (*) BYTE;
DECLARE FR_POT BASED FR_POT_PTR REAL;
DECLARE E_PAR BASED E_PAR_PTR (*) STRUCTURE( VAL (8) BYTE);
DECLARE VIDEO_NO BASED VIDEO_NO_PTR BYTE;
DECLARE PASS BASED PASS_PTR (*) BYTE;
DECLARE AXIS_TAB BASED AXIS_TAB_PTR (*) BYTE;
DECLARE AXIS_DES BASED AXIS_DES_PTR (*) BYTE;

/*-------------------------------------------------*/
/*ODDS AND SODS*/

DECLARE    FLAG BYTE;

DECLARE OS_FLAG WORD PUBLIC;
DECLARE TOR_FLAG WORD PUBLIC;
DECLARE CLOCK_I WORD PUBLIC;                /*SYSTEM CLOCK ms*/
DECLARE DISP3_DATA_LENGTH WORD PUBLIC;  /*INITIAL DISP3 LEN
-INIT ONLY ONCE*/
DECLARE DISP2_DATA_LENGTH WORD PUBLIC;  /*INITIAL DISP2 LEN
-INIT ONLY ONCE*/
DECLARE DISP1_DATA_LENGTH WORD PUBLIC;  /*INITIAL DISP1 LEN
-INIT ONLY ONCE*/
DECLARE SPEED BYTE;                         /*USED TO TURN ON
WHEEL SPEED CALC*/
DECLARE (VF_SPEED,VS_SPEED,VW_SPEED) REAL PUBLIC;/*ACTUAL
WHEEL SPEEDS IN RPM*/
DECLARE (VW_RPM,VS_RPM) (8) BYTE PUBLIC;          /*PROG WHEEL
SPEEDS IN RPM*/

/*POWER MEASUREMENT*/
DECLARE SPIND_POWER WORD;
DECLARE (GRIND_POWER,GRIND_POWER_NEW) REAL PUBLIC;

DECLARE PROC_P_MAX REAL PUBLIC;
DECLARE P_NO_LOAD REAL PUBLIC;
DECLARE P_CAL REAL PUBLIC;
DECLARE P_MAX REAL PUBLIC;

/*SPECIFIC ENERGY CALCULATION*/
DECLARE US_MAX REAL PUBLIC;
DECLARE EC (8) BYTE PUBLIC;
DECLARE US REAL;

DECLARE SIZE1_ZERO (8) BYTE PUBLIC;
DECLARE SIZE2_ZERO (8) BYTE PUBLIC;
DECLARE SIZE1 (8) BYTE PUBLIC;
DECLARE SIZE2 (8) BYTE PUBLIC;
DECLARE SIZE1_DIFF (8) BYTE PUBLIC;
DECLARE SIZE2_DIFF (8) BYTE PUBLIC;
DECLARE SIZE1_OLD (8) BYTE PUBLIC;
DECLARE SIZE2_OLD (8) BYTE PUBLIC;
DECLARE SIZE1_NEW (8) BYTE PUBLIC;
DECLARE SIZE2_NEW (8) BYTE PUBLIC;

DECLARE RESULT REAL;
DECLARE STRING_RES WORD;
DECLARE SIZER DWORD;
DECLARE CLK REAL;

DECLARE (COUNT,NEW_COUNT) INTEGER;
DECLARE (VS_COUNT,VW_COUNT,NEW_VS_COUNT,NEW_VW_COUNT) DWORD;

DECLARE (VW_RES,VS_RES,A_RES,B_RES) REAL PUBLIC;
DECLARE X_RAPID REAL PUBLIC;

DECLARE XT (8) BYTE PUBLIC;
DECLARE XA (8) BYTE PUBLIC;
DECLARE TOR (8) BYTE PUBLIC;
DECLARE T_INFEED (8) BYTE PUBLIC;
DECLARE OS_DIST (8) BYTE PUBLIC;
DECLARE TOR_RATIO (8) BYTE PUBLIC;

DECLARE BFC_BUFF (30) BYTE PUBLIC;
DECLARE IOBUF1 (128) BYTE PUBLIC;

/*-------------------------------------------------
          VARIOUS PUBLIC STRUCTURES
-------------------------------------------------*/

DECLARE MC_DATA1 STRUCTURE (
          CMPT_NO INTEGER,
          WHEEL_NO INTEGER,
          N1 (8) BYTE,
          N2 (8) BYTE,
          N3 (8) BYTE,
          BED_LEN (8) BYTE,
          SAFETY_DIST (8) BYTE,
          CYCLE_DIST (8) BYTE,
          CW_DIAM (8) BYTE,
          DWELL (8) BYTE,
          HOME (8) BYTE,
          SAFETY (8) BYTE,
          STRT (8) BYTE,
          TARGET (8) BYTE,
          OFFSET (8) BYTE,
          ALLOW (8) BYTE) PUBLIC;

DECLARE MC_DATA2 STRUCTURE (
          VF_RELAX (8) BYTE,
          VW_RELAX (8) BYTE,
          MIN_REVS (8) BYTE,
          Q_MIN (8) BYTE,
          P_RELAX (8) BYTE,
          SIZE_RELAX (8) BYTE,
          TOR_RELAX (8) BYTE,
          P_LIMIT_K (8) BYTE) PUBLIC;

DECLARE MC_DATA3 STRUCTURE (
          MAN_OR_SWITCH (6) BYTE,
          ADAPT_OR_SWITCH (6) BYTE,
          OS_OR_SWITCH (6) BYTE,
          PECK_OR_SWITCH (6) BYTE,
          OR_VS (8) BYTE,
          OR_VW (8) BYTE,
          OR_VF (8) BYTE,
          OR_OS (8) BYTE) PUBLIC;

DECLARE   PART STRUCTURE (
          NAME (12) BYTE,
          FINAL_DIM (8) BYTE,
          START_DIM (8) BYTE,
          INTRN_DIM (8) BYTE,
          CMPT_LEN (8) BYTE,
          TOR (8) BYTE,
          MATL_NO INTEGER) PUBLIC;

DECLARE CYCLE (4) STRUCTURE (
          ALLOW (8) BYTE,
          TOR (8) BYTE) PUBLIC;

DECLARE WHEEL STRUCTURE (
          NAME (8) BYTE,
          MAX_SPEED (8) BYTE,
          GRIT_SIZE (8) BYTE,
          INITIAL_DIAM (8) BYTE,
          CURRENT_DIAM (8) BYTE,
          FINAL_DIAM (8) BYTE,
          WIDTH (8) BYTE,
          DIFF (8) BYTE,
          COND (8) BYTE) PUBLIC;

DECLARE MATL STRUCTURE (
          NAME (6) BYTE,
          T_CRIT (8) BYTE,
          DIFF (8) BYTE,
          T_COND (8) BYTE,
          VS (8) BYTE,
          AR (8) BYTE,
          VM (8) BYTE) PUBLIC;


DECLARE AXIS STRUCTURE (
          VS_CALC (8) BYTE,
          VS_CURR (8) BYTE,

          VW_CALC (8) BYTE,
          VW_CURR (8) BYTE,
          VW_MAX (8) BYTE,
          VW_MIN (8) BYTE,

          VF_REAL_CALC (8) BYTE,
          VF_REAL_CURR (8) BYTE,
          VF_REAL_NEW (8) BYTE,

          VF_FEED_CALC (8) BYTE,
          VF_FEED_CURR (8) BYTE,
          VF_FEED_NEW (8) BYTE) PUBLIC;

DECLARE AX_POS STRUCTURE (
          PATT (8)BYTE,
          PINT (8)BYTE,
          E     (4)BYTE,
          VFF   (4)BYTE) PUBLIC;

DECLARE AB_POS STRUCTURE (
          PATT (8)BYTE,
          PINT (8)BYTE,
          E     (4)BYTE,
          VFF   (4)BYTE) PUBLIC;
```

```
DECLARE PASS_WORD STRUCTURE (
          LEVEL (8) BYTE) PUBLIC;


/*-------------------------------------------------
          EXTERNAL PROCEDURES:
-------------------------------------------------*/

/*-------------------------------------------------
          PROC TO INITIALISE STRUCTURES, INTERFACE
          OF DATA FILES TO PART PROGRAM FOR EDITING
          AND GENERATE REAL TIME DISPLAY NO. 1
-------------------------------------------------*/
DISP1:
          PROCEDURE EXTERNAL;
END DISP1;

/*-------------------------------------------------
          PROC TO GENERATE SECOND REAL TIME
          SCREEN DISPLAY NO. 2
-------------------------------------------------*/
DISP2:
          PROCEDURE EXTERNAL;
END DISP2;

/*-------------------------------------------------
          PROC TO GENERATE SECOND REAL TIME
          SCREEN DISPLAY NO. 3
-------------------------------------------------*/
DISP3:
          PROCEDURE EXTERNAL;
END DISP3;


/*-------------------------------------------------
          PROC TO WRITE FILTERED VALUE OF
          LEARNED TOR TO DATA BASE
-------------------------------------------------*/
LEARN_TOR:
          PROCEDURE EXTERNAL;
END LEARN_TOR;


/*-------------------------------------------------
          CALCULATE VF,VW,VS CHECKS THEM ALSO
          AXIS POSITIONS, PEAK POWER, ETC.
          PASSES DATA TO E PARAMETERS
-------------------------------------------------*/
CALC_VARS:
          PROCEDURE EXTERNAL;
END CALC_VARS;

/*-------------------------------------------------
          CALC CRITICAL ENERGY BASED ON CURRENT
          COMPONENT, MATERIAL, FEEDS AND SPEEDS
-------------------------------------------------*/
CALC_CRIT_ENERGY:
          PROCEDURE EXTERNAL;
END CALC_CRIT_ENERGY;

/*-------------------------------------------------
          ADAPTIVE INFEED RATE PROC
-------------------------------------------------*/
VF_UPDATE:
          PROCEDURE EXTERNAL;
END VF_UPDATE;

/*-------------------------------------------------
          ADAPTIVE CONTROL WHEEL SPEED PROC
-------------------------------------------------*/
VW_UPDATE:
          PROCEDURE EXTERNAL;
END VW_UPDATE;

/*-------------------------------------------------
          CALC ACTUAL SPECIFIC ENERGY -

          US = POWER/(0.5*PI*DW*LEN*VF)
-------------------------------------------------*/
CALC_US: PROCEDURE
(LR_POWER_PTR,LR_LEN_PTR,LR_DIAM_PTR,RE_VF_PTR,LR_US_PTR)
          EXTERNAL;
           DECLARE
(LR_POWER_PTR,LR_LEN_PTR,LR_DIAM_PTR,RE_VF_PTR,LR_US_PTR)
          POINTER;
END CALC_US;


/*-------------------------------------------------
          PROCEDURE TO DETERMINE THE VALUE OF
          OVERSHOOT FOR A GIVEN REQUIRED METAL
          REMOVAL BASED ON DATA BASE TOR
-------------------------------------------------*/
CALC_OS:
          PROCEDURE (OS_ERROR_FLAG_PTR) EXTERNAL;
          DECLARE OS_ERROR_FLAG_PTR POINTER;

END CALC_OS;

/*-------------------------------------------------
          PROCEDURE TO DETERMINE THE VALUE OF
          OVERSHOOT FOR A GIVEN REQUIRED METAL
          REMOVAL BASED ON RE-CALCULATED TOR
-------------------------------------------------*/
CALC_LEARNED_OS:
          PROCEDURE (OS_ERROR_FLAG_PTR) EXTERNAL;
          DECLARE OS_ERROR_FLAG_PTR POINTER;

END CALC_LEARNED_OS;

/*-------------------------------------------------
          PROCEDURE TO DETERMINE THE VALUE OF
          TOR FOR A GIVEN METAL REMOVAL
-------------------------------------------------*/
CALC_TOR:
          PROCEDURE (TOR_ERROR_FLAG_PTR) EXTERNAL;
          DECLARE TOR_ERROR_FLAG_PTR POINTER;

END CALC_TOR;

/*-------------------------------------------------
          PROCEDURE TO CALCULATE WHEEL SPEEDS
          FROM THE NEW AND OLD COUNTS AND RES
-------------------------------------------------*/

CALC_SPEED:
          PROCEDURE(DW_NEW_PTR,DW_OLD_PTR,RE_RES_PTR,RE_SPEED_
PTR)
          EXTERNAL;
          DECLARE
(DW_NEW_PTR,DW_OLD_PTR,RE_RES_PTR,RE_SPEED_PTR) POINTER;

END CALC_SPEED;

/*-------------------------------------------------
          PROC TO CALCULATE ACTUAL FEEDRATE
          FROM PROFILE SPEED AND CLOCK COUNT
-------------------------------------------------*/
CALC_VF_ACT:
          PROCEDURE(PROFILE_SPEED_PTR,CLOCK_I_PTR,VF_SPEED_PTR
)
          EXTERNAL;
          DECLARE (PROFILE_SPEED_PTR,CLOCK_I_PTR,VF_SPEED_PTR)
          POINTER;
END CALC_VF_ACT;

/*-------------------------------------------------
          FILTER SIZING DEVICE READINGS -
          OLD = OLD + RELAX*(NEW-OLD)
-------------------------------------------------*/
FILTER:
          PROCEDURE(NEW_PTR,OLD_PTR,RELAX_PTR)
          EXTERNAL;
          DECLARE (NEW_PTR,OLD_PTR,RELAX_PTR)
          POINTER;
END FILTER;

/*-------------------------------------------------
          TRIAL ROUTINE TO MONITOR KBD
-------------------------------------------------*/
/*READ_KBD:
          PROCEDURE EXTERNAL;

END READ_KBD;*/

/*-------------------------------------------------
          VARIOUS PROCS TO LOAD AND SAVE DATA BASE
          CONTENTS TO AND FROM E PARAMETERS
-------------------------------------------------*/
READ_CMPT_FILE:
          PROCEDURE EXTERNAL;
END READ_CMPT_FILE;

SAVE_CMPT_FILE:
          PROCEDURE EXTERNAL;
END SAVE_CMPT_FILE;

READ_MATL_FILE:
          PROCEDURE EXTERNAL;
END READ_MATL_FILE;

SAVE_MATL_FILE:
          PROCEDURE EXTERNAL;
END SAVE_MATL_FILE;

READ_WHEEL_FILE:
          PROCEDURE EXTERNAL;
END READ_WHEEL_FILE;

SAVE_WHEEL_FILE:
          PROCEDURE EXTERNAL;
END SAVE_WHEEL_FILE;

READ_CYCLE_FILE:
          PROCEDURE EXTERNAL;
END READ_CYCLE_FILE;

SAVE_CYCLE_FILE:
```

```
        PROCEDURE EXTERNAL;
END SAVE_CYCLE_FILE;

READ_MC_DATA1_FILE:
        PROCEDURE EXTERNAL;
END READ_MC_DATA1_FILE;

SAVE_MC_DATA1_FILE:
        PROCEDURE EXTERNAL;
END SAVE_MC_DATA1_FILE;

READ_MC_DATA2_FILE:
        PROCEDURE EXTERNAL;
END READ_MC_DATA2_FILE;

SAVE_MC_DATA2_FILE:
        PROCEDURE EXTERNAL;
END SAVE_MC_DATA2_FILE;

READ_MC_DATA3_FILE:
        PROCEDURE EXTERNAL;
END READ_MC_DATA3_FILE;

SAVE_MC_DATA3_FILE:
        PROCEDURE EXTERNAL;
END SAVE_MC_DATA3_FILE;

/*-------------------------------------------------
        VARIOUS TASK CREATION PROCS
-------------------------------------------------*/

ECTASK:
        PROCEDURE
(NDP,TASK_ADDR,DTD_ADDR,LENGTH,PRIORITY,SWITCH,
        MASKM,MASKS) EXTERNAL;
        DECLARE (NDP,PRIORITY,SWITCH,MASKM,MASKS) BYTE;
        DECLARE LENGTH WORD;
        DECLARE (TASK_ADDR,DTD_ADDR) POINTER;
END ECTASK;

ECSEM:
        PROCEDURE (SEMAFORE_ADDR,SEMAFORE_TYPE,INIT_VAL)
EXTERNAL;
        DECLARE SEMAFORE_ADDR POINTER;
        DECLARE (SEMAFORE_TYPE,INIT_VAL) BYTE;
END ECSEM;

ESEND:
        PROCEDURE (SEMAFORE) EXTERNAL;
        DECLARE SEMAFORE POINTER;
END ESEND;

EWAIT:
        PROCEDURE (SEMAFORE) EXTERNAL;
        DECLARE SEMAFORE POINTER;
END EWAIT;

/*-------------------------------------------------
        GET AXIS POSITION RELATIVE TO OFFSET
-------------------------------------------------*/
GET_POS_SPIAZ:
        PROCEDURE(AXIS_NAME,AX_POS,STATUS) EXTERNAL;
        DECLARE AXIS_NAME BYTE;
        DECLARE    AX_POS POINTER;
        DECLARE STATUS POINTER;
END GET_POS_SPIAZ;
```

```
DECLARE STATUS WORD;

DECLARE    (X1_POS,Y1_POS,X2_POS,Y2_POS,RADIUS,X_RAD,Y_RAD)
INTEGER;

/*-------------------------------------------------
        MAIN PROCEDURES:
-------------------------------------------------*/

CSI_INIT:
        PROCEDURE PUBLIC;

        K_BUFF_PTR = SYS_TAB (1);
        PROC_1_PTR = SYS_TAB (9);
        TIM_TAB_PTR = SYS_TAB(4);

        SYM_TAB_PTR = PROC_1_TAB (0);
        PROC_AXIS_CONT_PTR = PROC_1_TAB(5);
        PROC_OUT_PTR = PROC_1_TAB(13);
        INTERP_TABLE_PTR = PROC_1_TAB(15);
        TTC_CON_PTR = PROC_1_TAB (30);
        AXIS_TAB_PTR = PROC_1_TAB(27);

        CLOCK_I = 0;
        CALL MOVB(@INTERP_TABLE(1),@CLOCK_I,1);
        PROFILE_SPEED_PTR = @INTERP_TABLE(22);

        END_POINT_PTR = @PROC_AXIS_TAB(63);
        BUFFER_INDEX_PTR = @PROC_OUT(0);
        FR_POT_PTR = @TTC_CON_TAB(9);

        VIDEO_NO_PTR = @SK(25);

        CALL ECSEM(@SEM,1,0);
        CALL ECTASK(1,@CSI_TASK,@DTD,2048,150,2,0FFH,0FFH);
        CALL INIT_GRAF;

        SLOW_CSI_PTR = @SLOW_CSI;
        FAST_CSI_PTR = @FAST_CSI;

END CSI_INIT;

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

SLOW_CSI:
        PROCEDURE PUBLIC;

DECLARE TEMP_RE REAL;
DECLARE TEMP_WORD WORD;
DECLARE FR_OR REAL;

DO;

IF SK(600) = 1 THEN DO;

        SK(630) = 0;

        CALL SYM_TAB_SEARCH(@('E     '),@E_PAR_PTR,@FLAG);
        IF FLAG = 0 THEN SK(630) = SK(630) OR 1;

        CALL SYM_TAB_SEARCH(@('SYVAR'),@SYVAR_PTR,@FLAG);
        IF FLAG = 0 THEN SK(630) = SK(630) OR 2;

        CALL SYM_TAB_SEARCH(@('PASS '),@PASS_PTR,@FLAG);
        IF FLAG = 0 THEN SK(630) = SK(630) OR 4;
```

```
        CALL SEARCH_RES('A',@A_RES,@FLAG);
        IF FLAG = 0 THEN SK(630) = SK(630) OR 8;

        CALL SEARCH_RES('B',@B_RES,@FLAG);
        IF FLAG = 0 THEN SK(630) = SK(630) OR 16;

        CALL SEARCH_RES('S',@VS_RES,@FLAG);
        IF FLAG = 0 THEN SK(630) = SK(630) OR 32;

        CALL SEARCH_RES('W',@VW_RES,@FLAG);
        IF FLAG = 0 THEN SK(630) = SK(630) OR 64;

        CALL SEARCH_RAPID('X',@X_RAPID,@FLAG);
        IF FLAG = 0 THEN SK(630) = SK(630) OR 128;

/*-------------------------------------------------
        TEST ALL FLAGS
-------------------------------------------------*/

        IF SK(630) = 0 THEN SK(600) = 0FFH;
        ELSE SK(600) = 0;

        DISP3_DATA_LENGTH = 0;
        DISP2_DATA_LENGTH = 0;
        DISP1_DATA_LENGTH = 0;

        SPEED = 1;
        GRIND_POWER = 0.0;
        CALL R2LR(@GRIND_POWER,@E_PAR(60).VAL);

        PROC_P_MAX = 0.0;
        CALL R2LR(@PROC_P_MAX,@E_PAR(62).VAL);

        US_MAX = 0.0;
        CALL R2LR(@US_MAX,@E_PAR(63).VAL);

        P_NO_LOAD = 0.0;
        CALL R2LR(@P_NO_LOAD,@E_PAR(61).VAL);

        TEMP_RE = 0.0;
        CALL R2LR(@TEMP_RE,@TOR);

        TEMP_RE = 0.0;
        CALL R2LR(@TEMP_RE,@EC);

/*-------------------------------------------------
        ZERO SIZERS
-------------------------------------------------*/

        CALL LR_SUB(@SIZE1_ZERO,@SIZE1_ZERO,@SIZE1_ZERO);
        CALL LR_SUB(@SIZE1_DIFF,@SIZE1_DIFF,@SIZE1_DIFF);
        CALL LR_SUB(@SIZE1,@SIZE1,@SIZE1);
        CALL LR_SUB(@SIZE1_OLD,@SIZE1_OLD,@SIZE1_OLD);

        CALL LR_SUB(@SIZE2_ZERO,@SIZE2_ZERO,@SIZE2_ZERO);
        CALL LR_SUB(@SIZE2_DIFF,@SIZE2_DIFF,@SIZE2_DIFF);
        CALL LR_SUB(@SIZE2,@SIZE2,@SIZE2);
        CALL LR_SUB(@SIZE2_OLD,@SIZE2_OLD,@SIZE2_OLD);

        END;

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

IF (SK(601) AND SEMA) = 1 THEN DO;
```

```
          CALL ESEND(@SEM);
          SK(601) = (SK(601) AND NOT SEMA);
          END;

IF SK(600) = 0FFH AND SK(612) = 1 THEN DO;

/*-------------------------------------------------*/
IF SK(602) = 1 THEN DO;
          CALL CALC_VARS;
          SK(602) = 0FFH;
          END;


IF SK(602) = 3 THEN DO;
          CALL CALC_CRIT_ENERGY;
          SK(602) = 0FFH;
          END;

/*-------------------------------------------------
          CALIBRATE NO LOAD POWER
          - MODIFIED TO CONTINUOUSLY
            CALIBRATE NO-LOAD POWER
            UNLESS GRINDING-LOCKED TO P_MAX SEARCH
          - AVERAGE OVER 100 SAMPLES

          - SK603=2 START CALIBRATING
          - SK603=<> 2 STOP CALIBRATING
-------------------------------------------------*/

IF SK(603) = 2 THEN DO;
          P_NO_LOAD = P_NO_LOAD+(GRIND_POWER_NEW-
P_NO_LOAD)/100.0;
          CALL R2LR(@P_NO_LOAD,@E_PAR(61).VAL);
          END;

/*-------------------------------------------------
          PASS AXIS STATUS TO E0.BY
-------------------------------------------------*/
CALL LR_CMP(@AX_POS.PATT,@MC_DATA1.HOME,@RESULT);

IF RESULT <= 0.004 AND RESULT >= -0.004 THEN
          CALL MOVB(@(1),@E_PAR(0).VAL,1);

ELSE CALL MOVB(@(0),@E_PAR(0).VAL,1);

/*-------------------------------------------------
          IF PECK OVERRIDE 'ON' THEN SK628=255
          FOR PART PROGRAM -OTHERWISE SK628 = 0
-------------------------------------------------*/

STRING_RES = CMPB(@MC_DATA3.PECK_OR_SWITCH,@('OFF    '),6);

IF STRING_RES = 0FFFFH THEN SK(628) = 0;

STRING_RES = CMPB(@MC_DATA3.PECK_OR_SWITCH,@('ON     '),6);

IF STRING_RES = 0FFFFH THEN SK(628) = 255;

/*-------------------------------------------------
          UPDATE VF  AND VW ACCORDING TO GRIND
-------------------------------------------------*/
IF SK(608) = 1 THEN DO;
          CALL VF_UPDATE;
          SK(608) = 0FFH;
          END;

IF SK(608) = 2 THEN DO;
          CALL VW_UPDATE;
          SK(608) = 0FFH;
          END;

IF SK(608) = 4 THEN DO;
          CALL CALC_OS(@OS_FLAG);
          CALL MOVB(@OS_FLAG,@SK(637),1);
          SK(608) = 0FFH;
          END;

IF SK(608) = 8 THEN DO;
          CALL CALC_TOR(@TOR_FLAG);
          CALL MOVB(@TOR_FLAG,@SK(637),1);
          SK(608) = 0FFH;
          END;

IF SK(608) = 16 THEN DO;
          CALL CALC_LEARNED_OS(@OS_FLAG);
          CALL MOVB(@OS_FLAG,@SK(637),1);
          SK(608) = 0FFH;
          END;

IF SK(608) = 32 THEN DO;
          CALL LR2R(@AXIS.VF_FEED_CURR,@TEMP_RE);

          FR_OR = FLOAT(INT(CLOCK_I))*TEMP_RE/(X_RAPID*10.0);

IF FR_OR > 0.0 AND FR_OR <= 125.0 THEN DO;
          CALL R2W(@FR_OR,@TEMP_WORD);
          CALL MOVB(@TEMP_WORD,@SK(638),1);
          SK(608) = 0FFH;
          END;

ELSE SK(608) = 00H;

          END;

/*IF SK(619) = 1 THEN DO;
          CALL READ_KBD;
          SK(619) = 0FFH;
          END;*/

END;
END;

END SLOW_CSI;


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

CSI_TASK:
          PROCEDURE PUBLIC;

DO WHILE 1;

          CALL EWAIT(@SEM);

          IF SK(610) = 01H THEN
                    CALL READ_CMPT_FILE;
          ELSE IF SK(610) = 02H THEN
                    CALL SAVE_CMPT_FILE;
          ELSE IF SK(610) = 04H THEN
                    CALL READ_MATL_FILE;
          ELSE IF SK(610) = 08H THEN
                    CALL SAVE_MATL_FILE;
          ELSE IF SK(610) = 010H THEN
                    CALL READ_WHEEL_FILE;
          ELSE IF SK(610) = 020H THEN
                    CALL SAVE_WHEEL_FILE;
          ELSE IF SK(610) = 040H THEN
                    CALL READ_CYCLE_FILE;
          ELSE IF SK(610) = 080H THEN
                    CALL SAVE_CYCLE_FILE;


          IF SK(611) = 01H THEN
                    CALL DISP1;
          ELSE IF SK(611) = 02H THEN
                    CALL READ_MC_DATA1_FILE;
          ELSE IF SK(611) = 04H THEN
                    CALL SAVE_MC_DATA1_FILE;
          ELSE IF SK(611) = 08H THEN
                    CALL READ_MC_DATA2_FILE;
          ELSE IF SK(611) = 10H THEN
                    CALL SAVE_MC_DATA2_FILE;
          ELSE IF SK(611) = 20H THEN
                    CALL READ_MC_DATA3_FILE;
          ELSE IF SK(611) = 40H THEN
                    CALL SAVE_MC_DATA3_FILE;

/*-------------------------------------------------
          LOAD COUNTERS
-------------------------------------------------*/

          IF SK(613) = 02H THEN
                    CALL READ_MC_DATA1_FILE;
          ELSE IF SK(613) = 04H THEN
                    CALL SAVE_MC_DATA1_FILE;

          IF SK(602) = 2 THEN DO;
                    CALL DISP2;
                    SK(602) = 0FFH;
                    END;

          IF SK(602) = 8 THEN DO;
                    CALL DISP3;
                    SK(602) = 0FFH;
                    END;

          IF SK(618) = 1 THEN DO;
                    CALL LEARN_TOR;
                    SK(618) = 0FFH;
                    END;


          IF SK(615) = 1 THEN DO;
                    CALL LINER_DRAW;
                    SK(615) = 0FFH;
                    END;

          IF SK(615) = 2 THEN DO;
                    CALL MOVB(@GRAPHICS_ON,@GRAPHICS_ON,1);
                    CALL BIRD_PLOT;
                    CALL CLEAR_GRAPH;
                    CALL MOVB(@GRAPHICS_OFF,@GRAPHICS_OFF,1);
                    SK(615) = 0FFH;
                    END;

          SK(610) = 255;
          SK(611) = 255;
```

```
        SK(613) = 255;

END;
END CSI_TASK;

/*-------------------------------------------------
        Search symbol table for variable

        INPUT:    SYM_NAME_PTR -pointer to (5) bytes name
i.e."E"
                  SYM_PTR  -pointer to parameter pointer
(?)
                  ERROR_PTR -pointer to error flag

        OUTPUT:   ERROR -byte = 0 variable found
                                FFH variable not found

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

SYM_TAB_SEARCH:
        PROCEDURE (SYM_NAME_PTR,SYM_PTR,ERROR_PTR) PUBLIC;

        DECLARE SYM_NAME_PTR POINTER;
        DECLARE SYM_PTR POINTER;
        DECLARE I BYTE;
        DECLARE SYM_TAB_RES WORD;
        DECLARE ERROR_PTR POINTER;

I = 0;
CALL MOVB(@(0),ERROR_PTR,1);
DO WHILE I < 50;
        SYM_TAB_RES = CMPB(SYM_NAME_PTR,@SYM_TAB(I*19),5);
        IF SYM_TAB_RES = 0FFFFH THEN DO;
                CALL MOVB(@SYM_TAB(I*19+14),SYM_PTR,4);
                CALL MOVB(@(0FFH),ERROR_PTR,1);
                I = 50;
                END;
        I = I + 1;
        END;

END SYM_TAB_SEARCH;

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

FAST_CSI:
        PROCEDURE PUBLIC;

DECLARE TEMP_LR (8) BYTE;
DECLARE TEMP_RE REAL;
DECLARE TEMP_WORD WORD;
DECLARE P_LIMIT_ENABLE BYTE;
DECLARE (INIT_FR_OR,FR_OR) REAL;
DECLARE (VF_RAMP,VF_CURR,PK) REAL;

DO;

IF SPEED = START THEN DO;
        COUNT = SYS_CLK;
        CALL MOVB(@ENC_2,@VS_COUNT,4);
        CALL MOVB(@ENC_3,@VW_COUNT,4);
        SPEED = NOT_START;
        END;

/*-------------------------------------------------
        FIND PROGRAMMED SPINDLE SPEED FROM BUFFER
                1 OR 2
-------------------------------------------------*/

IF BUFFER_INDEX=0 THEN DO;
        PROG_VS_SPEED_PTR = @PROC_OUT(19);
        S_FUNC_PRES_PTR = @PROC_OUT(18);
        END;

IF BUFFER_INDEX=1 THEN DO;
        PROG_VS_SPEED_PTR = @PROC_OUT(388);
        S_FUNC_PRES_PTR = @PROC_OUT(387);
        END;

/*-------------------------------------------------
        CALC WHEEL SPEEDS EVERY SECOND
-------------------------------------------------*/

NEW_COUNT = SYS_CLK;

IF NEW_COUNT <> COUNT THEN DO;

        CALL MOVB(@ENC_2,@NEW_VS_COUNT,4);
        CALL MOVB(@ENC_3,@NEW_VW_COUNT,4);

        CALL
CALC_SPEED(@NEW_VS_COUNT,@VS_COUNT,@VS_RES,@VS_SPEED);
        CALL
CALC_SPEED(@NEW_VW_COUNT,@VW_COUNT,@VS_RES,@VW_SPEED);

        VS_SPEED = ABS(VS_SPEED);
        VW_SPEED = ABS(VW_SPEED);
        COUNT = NEW_COUNT;
        VS_COUNT = NEW_VS_COUNT;
        VW_COUNT = NEW_VW_COUNT;

        END;

/*-------------------------------------------------
        FLAG SK552 STATUS OF WHEEL SPEEDS
-------------------------------------------------*/

IF      VS_SPEED > 0.95*ABS(PROG_VS_SPEED) AND
        VS_SPEED < 1.05*ABS(PROG_VS_SPEED) AND
        S_FUNC_PRES > 0 THEN
                SK(552) = SK(552) OR 02H;

ELSE    SK(552) = SK(552) AND 0FDH;

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

CALL LR2R(@VW_RPM,@TEMP_RE);

IF      VW_SPEED > 0.95*TEMP_RE AND
        VW_SPEED < 1.05*TEMP_RE AND
        TEMP_RE > 0.0 THEN
                SK(552) = SK(552) OR 04H;

ELSE    SK(552) = SK(552) AND 0FBH;

/*-------------------------------------------------
        CALC SPINDLE POWER
        GRIND_POWER_NEW = FLOAT(INT(128-SPIND_POWER))*180.0
        *VS_SPEED*0.236/(1000.0*128.0);
-------------------------------------------------*/

CALL A2D(8,@SPIND_POWER);
GRIND_POWER_NEW = FLOAT(INT(128-
SPIND_POWER))*VS_SPEED/4708.10;

/*CALL MOVB(@E_PAR(29).VAL,@GRIND_POWER_NEW,4);*/
CALL LR2R(@MC_DATA2.P_RELAX,@TEMP_RE);

GRIND_POWER = GRIND_POWER +TEMP_RE*(GRIND_POWER_NEW-
GRIND_POWER)/100.0;
P_CAL = GRIND_POWER - P_NO_LOAD;

/*-------------------------------------------------
        SOFTWARE JOG RETRACT IF POWER > 65kW
-------------------------------------------------*/

IF GRIND_POWER > 60.00 AND SK(536) = 0 THEN SK(536) = 1;

/*-------------------------------------------------
        POWER LIMITER SECTION
        IF P > PMAX VF = VF + K*(PMAX/P - 1)
-------------------------------------------------*/

IF SK(616) = 1 THEN DO;
        SK(60) = SK(60) OR 010H;
/*      CALL MOVB(@(100),@SK(64),1);*/
        CALL LR2R(@AXIS.VF_FEED_CURR,@VF_RAMP);
        CALL LR2R(@AXIS.VF_FEED_CURR,@VF_CURR);
        CALL LR2R(@MC_DATA2.P_LIMIT_K,@PK);
        P_LIMIT_ENABLE = 0;
        SK(616) = 2;
        END;

IF GRIND_POWER > P_MAX AND SK(616) = 2 THEN DO;

IF P_LIMIT_ENABLE = 0 THEN DO;
        INIT_FR_OR = FLOAT(INT(SK(64)));
        P_LIMIT_ENABLE = 0FFH;
        END;

        VF_RAMP = VF_RAMP*(1.0+PK*(((P_MAX-
P_NO_LOAD)/P_CAL)-1.0)/100.0);

        IF VF_RAMP > 0.9*VF_CURR THEN DO;
                FR_OR = INIT_FR_OR*VF_RAMP/VF_CURR;
                CALL R2W(@FR_OR,@TEMP_WORD);
                CALL MOVB(@TEMP_WORD,@SK(64),1);
                END;

        ELSE DO;
                SK(536) = 1;
                SK(616) = 0FFH;
                END;

        END;

/*-------------------------------------------------
        DISABLE POWER LIMITER
-------------------------------------------------*/

IF SK(616) = 255 THEN DO;
        SK(60) = SK(60) AND 0EFH;
        CALL MOVB(@(100),@SK(64),1);
        SK(616) = 0;
        END;

/*-------------------------------------------------
```

```
          GET AXIS POSITION
-----------------------------------------------*/

CALL GET_POS_SPIAZ ('X',@AX_POS,@STATUS);

/*-----------------------------------------------
          SIZING DEVICE READING AND FILTER -REMOVED FOR DRO
-----------------------------------------------*/

CALL GET_POS_SPIAZ ('A',@AB_POS,@STATUS);
CALL MOVB(@AB_POS.PATT,@SIZE1_NEW,8);

CALL GET_POS_SPIAZ ('B',@AB_POS,@STATUS);
CALL MOVB(@AB_POS.PATT,@SIZE2_NEW,8);

IF SK(609) = 1 THEN DO;
          CALL
FILTER(@SIZE1_NEW,@SIZE1_OLD,@MC_DATA2.SIZE_RELAX);
          CALL
FILTER(@SIZE2_NEW,@SIZE2_OLD,@MC_DATA2.SIZE_RELAX);
          END;

ELSE DO;
          CALL MOVB(@SIZE1_NEW,@SIZE1_OLD,8);
          CALL MOVB(@SIZE2_NEW,@SIZE2_OLD,8);
          END;

/*-----------------------------------------------
          IF SK602=4 THEN ZERO 'COUNTER' AND  THEN USE
          TO CALCULATE AND DISPLAY ACTUAL DIAMETER
-----------------------------------------------*/

IF SK(602) = 4 THEN DO;
          CALL MOVB(@SIZE1_OLD,@SIZE1_ZERO,8);
          CALL MOVB(@SIZE2_OLD,@SIZE2_ZERO,8);
          SK(602)=255;
          END;

/*-----------------------------------------------
          DONATE ACTUAL SIZE TO DISP2
-----------------------------------------------*/

CALL LR_SUB(@SIZE1_OLD,@SIZE1_ZERO,@SIZE1_DIFF);
CALL LR_ADD(@PART.FINAL_DIM,@SIZE1_DIFF,@SIZE1);

/*-----------------------------------------------
          SEND  SIZE2 TO DISP2
-----------------------------------------------*/

CALL LR_SUB(@SIZE2_OLD,@SIZE2_ZERO,@SIZE2_DIFF);
CALL LR_ADD(@PART.FINAL_DIM,@SIZE2_DIFF,@SIZE2);

/*-----------------------------------------------
          CALC MAX_POWER AND MAX SPECIFIC ENERGY
-----------------------------------------------*/

IF SK(603) = 1 THEN DO;
          PROC_P_MAX = 0.0;
          US_MAX = 0.0;
          SK(603) = 8;
          END;

CALL CALC_VF_ACT(@PROFILE_SPEED,@CLOCK_I,@VF_SPEED);

IF SK(603) = 8 THEN DO;
          IF GRIND_POWER > 0.0 THEN DO;
                    IF GRIND_POWER > PROC_P_MAX THEN
PROC_P_MAX = GRIND_POWER;

                    IF P_CAL > 0.0 THEN DO;
                              CALL
CALC_US(@P_CAL,@PART.CMPT_LEN,@PART.FINAL_DIM,

@AXIS.VF_REAL_CURR,@US);
                              IF US > US_MAX THEN US_MAX =
US;
                              END;

                    END;
          END;

/*-----------------------------------------------
          VIDEO HANDLER
-----------------------------------------------*/

IF (VIDEO_NO AND 0FH) = 2 AND SK(614) = 1 THEN
          CALL MOVB(@GRAPHICS_ON,@GRAPHICS_ON,1);

IF SK(614) > 0 THEN DO;
          IF (VIDEO_NO AND 0FH) <> 2 OR SK(614) = 255 THEN
                    CALL MOVB(@GRAPHICS_OFF,@GRAPHICS_OFF,1);
          END;

/*-----------------------------------------------
          FAST CSI THAT INTERFERES WITH E PARAMS -DISABLED
          FOR PART-PROGRAM EDITING
-----------------------------------------------*/

IF SK(612) = 1 THEN DO;

/*-----------------------------------------------
          PASS AXIS POSN TO E66
-----------------------------------------------*/

CALL MOVB(@AX_POS.PATT,@E_PAR(66).VAL,8);

/*-----------------------------------------------
          GIVE SIZE DIFF TO E57 AND E58
-----------------------------------------------*/

CALL MOVB(@SIZE1_DIFF,@E_PAR(57).VAL,8);
CALL MOVB(@SIZE2_DIFF,@E_PAR(58).VAL,8);

/*-----------------------------------------------
          PASS WHEEL SPEEDS TO E28 AND E59
-----------------------------------------------*/

CALL MOVB(@VS_SPEED,@E_PAR(28).VAL,4);
CALL R2LR(@VW_SPEED,@E_PAR(59).VAL);

/*-----------------------------------------------
          PASS POWER READINGS TO E PARAMETERS
-----------------------------------------------*/

CALL R2LR(@P_CAL,@E_PAR(65).VAL);
CALL R2LR(@GRIND_POWER,@E_PAR(60).VAL);

/*-----------------------------------------------
          PASS PROCESS MAXIMUMS TO E62 AND E63
-----------------------------------------------*/

CALL R2LR(@PROC_P_MAX,@E_PAR(62).VAL);
CALL R2LR(@US_MAX,@E_PAR(63).VAL);

END;

END;

END FAST_CSI;

/*-----------------------------------------------
Procedure to find and search the axis descriptor
table for axis resolution

Input : Axis name -i.e. 'S'
          Axis resolution pointer -location of axis res.

Output : Real value of axis resolution
       : ERROR_FLAG = 0 if axis not found

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

SEARCH_RES:
          PROCEDURE (AXIS_NAME,AXIS_RES_PTR,ERROR_PTR) PUBLIC;

DECLARE AXIS_RES_PTR POINTER;
DECLARE AXIS_RES BASED AXIS_RES_PTR REAL;
DECLARE AXIS_NAME BYTE;

DECLARE I BYTE;
DECLARE ERROR_PTR POINTER;

/*-----------------------------------------------
Search through axis table until axis name found
whilst count is less than number of axes (AXIS_TAB(0))
-----------------------------------------------*/

I = 1;
CALL MOVB(@(0),ERROR_PTR,1);

DO WHILE I <=  AXIS_TAB(0);
          IF AXIS_TAB(1+(I-1)*11) = AXIS_NAME THEN DO;
                    CALL MOVB(@AXIS_TAB(3+(I-
1)*11),@AXIS_DES_PTR,4);
                    CALL MOVB(@AXIS_DES(120),@AXIS_RES,4);
                    I = AXIS_TAB(0);
                    CALL MOVB(@(0FFH),ERROR_PTR,1);
          END;
          I = I+1;
END;

END SEARCH_RES;


/*-----------------------------------------------
Procedure to find and search the axis descriptor
table for axis maximum speed

Input : Axis name -i.e. 'S'
          Axis rapid pointer -location of axis rapid.

Output : Real value of axis rapid
       : ERROR_FLAG = 0 if axis not found

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

SEARCH_RAPID:
```

```
        PROCEDURE (AXIS_NAME,AXIS_RAPID_PTR,ERROR_PTR)
PUBLIC;

DECLARE AXIS_RAPID_PTR POINTER;
DECLARE AXIS_RAPID BASED AXIS_RAPID_PTR REAL;
DECLARE AXIS_NAME BYTE;

DECLARE I BYTE;
DECLARE ERROR_PTR POINTER;

/*-------------------------------------------------
Search through axis table until axis name found
whilst count is less than number of axes (AXIS_TAB(0))
-------------------------------------------------*/

I = 1;
CALL MOVB(@(0),ERROR_PTR,1);

DO WHILE I <=  AXIS_TAB(0);
        IF AXIS_TAB(1+(I-1)*11) = AXIS_NAME THEN DO;
                CALL MOVB(@AXIS_TAB(3+(I-
1)*11),@AXIS_DES_PTR,4);
                CALL MOVB(@AXIS_DES(62),@AXIS_RAPID,4);
                I = AXIS_TAB(0);
                CALL MOVB(@(0FFH),ERROR_PTR,1);
        END;
        I = I+1;
END;

END SEARCH_RAPID;

/*-------------------------------------------------
A2D : Procedure to place the contents of A/D coverter
into a specified variable.

Input: Channel -byte value of which input

Output: A2D string -word value.

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

A2D:
        PROCEDURE (CHANNEL,A2D_STRING_PTR) PUBLIC;

DECLARE CHANNEL BYTE;
DECLARE A2D_STRING_PTR POINTER;
DECLARE A2D_PTR POINTER;
DECLARE A2D_WORD BASED A2D_PTR (8) WORD;

A2D_PTR = 11100H;

CALL MOVB(@A2D_WORD(CHANNEL-1),A2D_STRING_PTR,2);

END A2D;

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


LINER_DRAW:
        PROCEDURE PUBLIC;

DO;
        X1_POS = 100;
        Y1_POS = 120;
        X_RAD = 45;
        Y_RAD = 65;
        CALL ELLIPSE_PLOT(@X1_POS,@Y1_POS,@X_RAD,@Y_RAD);
        X_RAD = 32;
        Y_RAD = 52;
        CALL ELLIPSE_PLOT(@X1_POS,@Y1_POS,@X_RAD,@Y_RAD);

        X1_POS = 107;
        Y1_POS = 120;
        X_RAD = 45;
        Y_RAD = 65;
        CALL
SEMI_ELLIPSE_PLOT(@X1_POS,@Y1_POS,@X_RAD,@Y_RAD);

        Y1_POS = 120;
        X_RAD = 35;
        Y_RAD = 60;
DO X1_POS = 125 TO 255 BY 5;
        CALL
SHADE_ELLIPSE_PLOT(@X1_POS,@Y1_POS,@X_RAD,@Y_RAD);
        END;

        X1_POS = 260;
        Y1_POS = 120;
        X_RAD = 35;
        Y_RAD = 60;
        CALL
SEMI_ELLIPSE_PLOT(@X1_POS,@Y1_POS,@X_RAD,@Y_RAD);


/*-------------------------------------------------*/
        X1_POS = 100;
        Y1_POS = 35;
        X2_POS = 260;
        Y2_POS = 35;
        CALL LINE_PLOT(@X1_POS,@Y1_POS,@X2_POS,@Y2_POS);

        X1_POS = 100;
        Y1_POS = 35;
        X2_POS = 105;
        Y2_POS = 37;
        CALL LINE_PLOT(@X1_POS,@Y1_POS,@X2_POS,@Y2_POS);

        X1_POS = 100;
        Y1_POS = 35;
        X2_POS = 105;
        Y2_POS = 33;
        CALL LINE_PLOT(@X1_POS,@Y1_POS,@X2_POS,@Y2_POS);

        X1_POS = 260;
        Y1_POS = 35;
        X2_POS = 255;
        Y2_POS = 37;
        CALL LINE_PLOT(@X1_POS,@Y1_POS,@X2_POS,@Y2_POS);

        X1_POS = 260;
        Y1_POS = 35;
        X2_POS = 255;
        Y2_POS = 33;
        CALL LINE_PLOT(@X1_POS,@Y1_POS,@X2_POS,@Y2_POS);

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

        X1_POS = 305;
        Y1_POS = 180;
        X2_POS = 305;
        Y2_POS = 60;
        CALL LINE_PLOT(@X1_POS,@Y1_POS,@X2_POS,@Y2_POS);

        X1_POS = 305;
        Y1_POS = 180;
        X2_POS = 307;
        Y2_POS = 175;
        CALL LINE_PLOT(@X1_POS,@Y1_POS,@X2_POS,@Y2_POS);

        X1_POS = 305;
        Y1_POS = 180;
        X2_POS = 303;
        Y2_POS = 175;
        CALL LINE_PLOT(@X1_POS,@Y1_POS,@X2_POS,@Y2_POS);

        X1_POS = 305;
        Y1_POS = 60;
        X2_POS = 307;
        Y2_POS = 65;
        CALL LINE_PLOT(@X1_POS,@Y1_POS,@X2_POS,@Y2_POS);

        X1_POS = 305;
        Y1_POS = 60;
        X2_POS = 303;
        Y2_POS = 65;
        CALL LINE_PLOT(@X1_POS,@Y1_POS,@X2_POS,@Y2_POS);

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

        X1_POS = 45;
        Y1_POS = 170;
        X2_POS = 45;
        Y2_POS = 70;
        CALL LINE_PLOT(@X1_POS,@Y1_POS,@X2_POS,@Y2_POS);

        X1_POS = 45;
        Y1_POS = 170;
        X2_POS = 47;
        Y2_POS = 165;
        CALL LINE_PLOT(@X1_POS,@Y1_POS,@X2_POS,@Y2_POS);

        X1_POS = 45;
        Y1_POS = 170;
        X2_POS = 43;
        Y2_POS = 165;
        CALL LINE_PLOT(@X1_POS,@Y1_POS,@X2_POS,@Y2_POS);

        X1_POS = 45;
        Y1_POS = 70;
        X2_POS = 47;
        Y2_POS = 75;
        CALL LINE_PLOT(@X1_POS,@Y1_POS,@X2_POS,@Y2_POS);

        X1_POS = 45;
        Y1_POS = 70;
        X2_POS = 43;
        Y2_POS = 75;
        CALL LINE_PLOT(@X1_POS,@Y1_POS,@X2_POS,@Y2_POS);


        X1_POS = 194;
        Y1_POS = 250;
        X2_POS = 194;
        Y2_POS = 205;
        CALL LINE_PLOT(@X1_POS,@Y1_POS,@X2_POS,@Y2_POS);
```

```
            X1_POS = 194;
            Y1_POS = 205;
            X2_POS = 383;
            Y2_POS = 205;
            CALL LINE_PLOT(@X1_POS,@Y1_POS,@X2_POS,@Y2_POS);

            X1_POS = 194;
            Y1_POS = 217;
            X2_POS = 383;
            Y2_POS = 217;
            CALL LINE_PLOT(@X1_POS,@Y1_POS,@X2_POS,@Y2_POS);

            SK(610) = 255;
            END;

END         LINER_DRAW;

END CSI_TSK;
```

# TABLES.PLM

Header file for declaration of 8600 system data tables.

---

```
/*-----------------------------------------------
          SYSTEM POINTERS
-----------------------------------------------*/

DECLARE TRAP200 POINTER EXTERNAL;
DECLARE PROC_1_PTR POINTER EXTERNAL;

DECLARE SYM_TAB_PTR POINTER EXTERNAL;
DECLARE PROC_OUT_PTR POINTER EXTERNAL;
DECLARE INTERP_TABLE_PTR POINTER EXTERNAL;
DECLARE PROFILE_SPEED_PTR POINTER EXTERNAL;
DECLARE PROC_AXIS_CONT_PTR POINTER EXTERNAL;
DECLARE END_POINT_PTR POINTER EXTERNAL;
DECLARE TTC_CON_PTR POINTER EXTERNAL;
DECLARE FR_POT_PTR POINTER EXTERNAL;
DECLARE VIDEO_NO_PTR POINTER EXTERNAL;

DECLARE E_PAR_PTR POINTER EXTERNAL;
DECLARE K_BUFF_PTR POINTER EXTERNAL;
DECLARE SYVAR_PTR POINTER EXTERNAL;
DECLARE PASS_PTR POINTER EXTERNAL;
DECLARE PROG_VS_SPEED_PTR POINTER EXTERNAL;

DECLARE FALSE LITERALLY '0';
DECLARE TRUE LITERALLY '0FFH';

DECLARE SYS_TAB BASED TRAP200 (*) POINTER;
DECLARE PROC_1_TAB BASED PROC_1_PTR (*) POINTER;
DECLARE SYM_TAB BASED SYM_TAB_PTR (*) BYTE;
DECLARE E_PAR BASED E_PAR_PTR (*) STRUCTURE( VAL (8) BYTE);
DECLARE SK BASED K_BUFF_PTR (*) BYTE;
DECLARE SYVAR BASED SYVAR_PTR (*) BYTE;
DECLARE PASS BASED PASS_PTR (*) BYTE;
DECLARE PROC_OUT BASED PROC_OUT_PTR (*) BYTE;
DECLARE INTERP_TABLE BASED INTERP_TABLE_PTR (*) BYTE;
DECLARE PROFILE_SPEED BASED PROFILE_SPEED_PTR REAL;
DECLARE PROC_AXIS_TAB BASED PROC_AXIS_CONT_PTR (*) BYTE;
DECLARE END_POINT BASED END_POINT_PTR (8) BYTE;
DECLARE PROG_VS_SPEED BASED PROG_VS_SPEED_PTR REAL;
DECLARE TTC_CON_TAB BASED TTC_CON_PTR (*) BYTE;
DECLARE FR_POT BASED FR_POT_PTR REAL;
DECLARE VIDEO_NO BASED VIDEO_NO_PTR BYTE;
```

# DISP.PLM

## CSI module for file management and display of initial start up screen.

---

```
/*-------------------------------------------------

A PROGGY TO HANDLE FILE OPERATIONS AND GENERATE AN
INITIAL SCREEN DISPLAY.

(c) Sean Kelly March 1988

i.      Uses GRIND_DATA to retrieve:
            a) component
            b) material
            c) wheel

ii.     Then display on screen 9.

iii.    Also allows editing of data base:
            a) component
            b) material
            c) wheel
            d) grind data
        through blasting them into and from E's and SYVAR's
        to be used by GKNIO/MP2 etc in part program.

iv)     Will format and load data base files if
        non-existant.

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

DISP1_PROG: DO;

$INCLUDE(CSI.PLM)
$INCLUDE(FILES.PLM)
$INCLUDE(MATHS.PLM)
$INCLUDE(TABLES.PLM)

/*-------------------------------------------------
        FILTER TOR -
        OLD = OLD + RELAX*(NEW-OLD)
-------------------------------------------------*/
FILTER:
        PROCEDURE(NEW_PTR,OLD_PTR,RELAX_PTR)
        EXTERNAL;
        DECLARE (NEW_PTR,OLD_PTR,RELAX_PTR)
        POINTER;
END FILTER;

/*-------------------------------------------------
        PROGGY VARIABLES
-------------------------------------------------*/

DECLARE CHANNEL (300) WORD;
DECLARE STAT INTEGER;
DECLARE IOSTAT POINTER;
DECLARE DISP1_DATA_LENGTH WORD EXTERNAL;

DECLARE   PART STRUCTURE (
        NAME (12) BYTE,
        FINAL_DIM (8) BYTE,
        START_DIM (8) BYTE,
        INTRN_DIM (8) BYTE,
        CMPT_LEN (8) BYTE,
        TOR (8) BYTE,
        MATL_NO INTEGER) EXTERNAL;

DECLARE PART_DEFAULT STRUCTURE (
        NAME (12) BYTE,
        FINAL_DIM REAL,
        START_DIM REAL,
        INTRN_DIM REAL,
        CMPT_LEN REAL,
        TOR REAL,
        MATL_NO INTEGER)

        DATA ('GKN.01      ',
        103.248,
        103.650,
        98.200,
        225.00,
        1.0,
        1);

DECLARE PART_FORMAT (15) BYTE DATA
(7,'A',12,'L',3,'L',3,'L',3,
                                        'L',3,'L',3,'I',1);

DECLARE CYCLE (4) STRUCTURE (
        ALLOW (8) BYTE,
        TOR (8) BYTE) EXTERNAL;

DECLARE CYCLE_DEFAULT (4) STRUCTURE (
        ALLOW REAL,
        TOR REAL)

        DATA (
        0.0,
        0.0,
        0.0,
        0.0,
        0.0,
        0.0,
        0.0,
        0.0);

DECLARE CYCLE_FORMAT (17) BYTE DATA
(8,'L',3,'L',3,'L',3,'L',3,

'L',3,'L',3,'L',3,'L',3);

DECLARE MATL STRUCTURE (
        NAME (6) BYTE,
        T_CRIT (8) BYTE,
        DIFF (8) BYTE,
        T_COND (8) BYTE,
        VS (8) BYTE,
        AR (8) BYTE,
        VM (8) BYTE) EXTERNAL;

DECLARE MATL_DEFAULT STRUCTURE (
        NAME (6) BYTE,
        T_CRIT REAL,
        DIFF REAL,
        T_COND REAL,
        VS REAL,
        AR REAL,
        VM REAL)

        DATA ('C.I.  ',
        330.0,
        14.7,
        53.7,
        60.0,
        22590.0,
        54.7);

DECLARE MATL_FORMAT (15) BYTE DATA (7,'A',6,'L',3,'L',3,

        'L',3,'L',3,'L',3,'L',3);

DECLARE WHEEL STRUCTURE (
        NAME (8) BYTE,
        MAX_SPEED (8) BYTE,
        GRIT_SIZE (8) BYTE,
        INITIAL_DIAM (8) BYTE,
        CURRENT_DIAM (8) BYTE,
        FINAL_DIAM (8) BYTE,
        WIDTH (8) BYTE,
        DIFF (8) BYTE,
        COND (8) BYTE) EXTERNAL;

DECLARE WHEEL_DEFAULT STRUCTURE (
        NAME (8) BYTE,
        MAX_SPEED REAL,
        GRIT_SIZE REAL,
        INITIAL_DIAM REAL,
        CURRENT_DIAM REAL,
        FINAL_DIAM REAL,
        WIDTH REAL,
        DIFF REAL,
        COND REAL)

        DATA ('UNIVERS1',
        45.0,
        00.39,
        609.2,
        609.2,
        305.0,
        300.0,
        1.145,
        0.55);

DECLARE WHEEL_FORMAT (19) BYTE DATA (9,'A',8,'L',3,'L',3,

        'L',3,'L',3,'L',3,'L',3,
                                        'L',3,'L',3);

DECLARE MC_DATA1 STRUCTURE (
        CMPT_NO INTEGER,
        WHEEL_NO INTEGER,
        N1 (8) BYTE,
        N2 (8) BYTE,
        N3 (8) BYTE,
```

```
        BED_LEN (8) BYTE,
        SAFETY_DIST (8) BYTE,
        CYCLE_DIST (8) BYTE,
        CW_DIAM (8) BYTE,
        DWELL (8) BYTE,
        HOME (8) BYTE,
        SAFETY (8) BYTE,
        STRT (8) BYTE,
        TARGET (8) BYTE,
        OFFSET (8) BYTE,
        ALLOW (8) BYTE) EXTERNAL;

DECLARE MC_DATA1_DEFAULT STRUCTURE (
        CMPT_NO INTEGER,
        WHEEL_NO INTEGER,
        N1 REAL,
        N2 REAL,
        N3 REAL,
        BED_LEN REAL,
        SAFETY_DIST REAL,
        CYCLE_DIST REAL,
        CW_DIAM REAL,
        DWELL REAL,
        HOME REAL,
        SAFETY REAL,
        STRT REAL,
        TARGET REAL,
        OFFSET REAL,
        ALLOW REAL)

        DATA (1,1,
        0.0,
        0.0,
        50.0,
        130.0,
        0.1,
        5.0,
        343.0,
        10.0,
        0.0,
        0.0,
        0.0,
        0.0,
        0.0,
        0.0);


DECLARE MC_DATA1_FORMAT (33) BYTE DATA
(16,'I',1,'I',1,'L',3,'L',3,

        'L',3,'L',3,'L',3,'L',3,'L',3,'L',3,'L',3,

        'L',3,'L',3,'L',3,'L',3,'L',3,'L',3,'L',3);


DECLARE MC_DATA2 STRUCTURE (
        VF_RELAX (8) BYTE,
        VW_RELAX (8) BYTE,
        MIN_REVS (8) BYTE,
        Q_MIN (8) BYTE,
        P_RELAX (8) BYTE,
        SIZE_RELAX (8) BYTE,
        TOR_RELAX (8) BYTE,
        P_LIMIT_K (8) BYTE) EXTERNAL;

DECLARE MC_DATA2_DEFAULT STRUCTURE (
        VF_RELAX REAL,
        VW_RELAX REAL,
        MIN_REVS REAL,
        Q_MIN REAL,
        P_RELAX REAL,
        SIZE_RELAX REAL,
        TOR_RELAX REAL,
        P_LIMIT_K REAL)

        DATA (
        20.0,
        20.0,
        1.0,
        80.0,
        5.0,
        5.0,
        20.0,
        1.0);


DECLARE MC_DATA2_FORMAT (17) BYTE DATA (8,'L',3,'L',3,'L',3,
                                        'L',3,'L',3,'L',3,'L',3,'L',3);

DECLARE MC_DATA3 STRUCTURE (
        MAN_OR_SWITCH (6) BYTE,
        ADAPT_OR_SWITCH (6) BYTE,
        OS_OR_SWITCH (6) BYTE,
        PECK_OR_SWITCH (6) BYTE,
        OR_VS (8) BYTE,
        OR_VW (8) BYTE,
        OR_VF (8) BYTE,
        OR_OS (8) BYTE) EXTERNAL;

DECLARE MC_DATA3_DEFAULT STRUCTURE (
        MAN_OR_SWITCH (6) BYTE,
        ADAPT_OR_SWITCH (6) BYTE,
        OS_OR_SWITCH (6) BYTE,
        PECK_OR_SWITCH (6) BYTE,
        OR_VS REAL,
        OR_VW REAL,
        OR_VF REAL,
        OR_OS REAL)

        DATA(
        'OFF   ',
        'OFF   ',
        'OFF   ',
        'OFF   ',
        1410.00,
        25.00,
        0.1,
        0.0);

DECLARE MC_DATA3_FORMAT (17) BYTE
DATA(8,'A',6,'A',6,'A',6,'A',6,

        'L',3,'L',3,'L',3,'L',3);

DECLARE PASS_WORD STRUCTURE (
        LEVEL (8) BYTE) EXTERNAL;

DECLARE INIT_PASS (24) BYTE
        DATA(
        'OPERAT01',
        'FOREMAN1',
        'SUPERV01');
```



```
DECLARE PASS_WORD_FORMAT (3) BYTE DATA (1,'A',8);

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

DECLARE READ LITERALLY '1';
DECLARE WRITE LITERALLY '2';
DECLARE RE_WR LITERALLY '3';
DECLARE SEQ LITERALLY '00H';
DECLARE RAN LITERALLY '0FFH';
DECLARE FIXED LITERALLY '00H';
DECLARE VARY LITERALLY '0FFH';

DECLARE DEVICE (4) BYTE DATA ('/MP0');
DECLARE MC_DATA1_FILE (6) BYTE DATA ('MCDAT1');
DECLARE MC_DATA2_FILE (6) BYTE DATA ('MCDAT2');
DECLARE MC_DATA3_FILE (6) BYTE DATA ('MCDAT3');
DECLARE PART_FILE (6) BYTE DATA ('PARTAB');
DECLARE WHEEL_FILE (6) BYTE DATA ('WHETAB');
DECLARE MATL_FILE (6) BYTE DATA ('MATTAB');
DECLARE CYCLE_FILE (6) BYTE DATA ('CYCTAB');
DECLARE PASS_FILE (6) BYTE DATA ('PASTAB');

DECLARE MC_DATA1_LEN LITERALLY '116';
DECLARE MC_DATA2_LEN LITERALLY '64';
DECLARE MC_DATA3_LEN LITERALLY '56';
DECLARE PART_LEN LITERALLY '54';
DECLARE WHEEL_LEN LITERALLY '72';
DECLARE MATL_LEN LITERALLY '54';
DECLARE CYCLE_LEN LITERALLY '64';
DECLARE PASS_LEN LITERALLY '8';

DECLARE REC_COUNT WORD;
DECLARE INDEX WORD;

DECLARE ASC_BUFFER (20) BYTE;
DECLARE BLANKS (20) BYTE DATA ('                    ');
DECLARE STAT4 (2) WORD;

/*-------------------------------------------------
          VARIABLES FOR LEARN TOR
-------------------------------------------------*/

DECLARE XT (8) BYTE EXTERNAL;
DECLARE XA (8) BYTE EXTERNAL;
DECLARE TOR (8) BYTE EXTERNAL;
DECLARE T_INFEED (8) BYTE EXTERNAL;
DECLARE OS_DIST (8) BYTE EXTERNAL;

DECLARE GRIND1_DISP STRUCTURE(

        CONTROL_WORD WORD,
        REAL_TIME_ROUT_PTR POINTER,
        GLOBAL_ATTRIBUTES BYTE,
        NUMBER_OF_ITEMS WORD,

        WHEEL_LABEL1 (16) BYTE,
        VERT1 (6) BYTE,
        WHEEL_LABEL2 (22) BYTE,
        WHEEL_LABEL3 (22) BYTE,
        MACHINE_LABEL (10) BYTE,
        VERT2 (6) BYTE,
        VERT3 (6) BYTE,
        N1_STRING_LABEL (18) BYTE,
        N2_STRING_LABEL (18) BYTE,
        N1_STRING (13) BYTE,
```

```
        TIL_DRES (13) BYTE,
        VERT4 (6) BYTE,
        VERT5 (6) BYTE,
        VERT6 (6) BYTE,
        VERT7 (6) BYTE,
        DIVIDER (69) BYTE,
        WHEEL_NAME_STRING (13) BYTE,
        WHEEL_SPEED_STRING (10) BYTE,
        WHEEL_DIAM_STRING (13) BYTE,
        COMPONENT_LABEL (16) BYTE,
        VERT8 (6) BYTE,
        CMPT_LABEL1 (50) BYTE,
        CMPT_LABEL2 (37) BYTE,
        CMPT_LABEL3 (28) BYTE,
        CMPT_NAME_STRING (11) BYTE,
        CMPT_FD_STRING (13) BYTE,
        CMPT_SD_STRING (13) BYTE,
        CMPT_ID_STRING (13) BYTE,
        CMPT_LEN_STRING (13) BYTE,
        CMPT_MATL_STRING (11) BYTE,
        DIVIDER2 (69) BYTE,
        OPT_LABEL (13) BYTE,
        VERT9 (6) BYTE,
        OPT_LABEL2 (14) BYTE,
        OPT_LABEL3 (14) BYTE,
        OPT_LABEL4 (16) BYTE,
        OPT_LABEL5 (14) BYTE,
        OPT_LABEL6 (16) BYTE,
        OPT_LABEL7 (14) BYTE);

DECLARE GRIND1_DISP_INIT STRUCTURE(

        CONTROL_WORD WORD,
        REAL_TIME_ROUT_PTR POINTER,
        GLOBAL_ATTRIBUTES BYTE,
        NUMBER_OF_ITEMS WORD,

        WHEEL_LABEL1 (16) BYTE,
        VERT1 (6) BYTE,
        WHEEL_LABEL2 (22) BYTE,
        WHEEL_LABEL3 (22) BYTE,
        MACHINE_LABEL (10) BYTE,
        VERT2 (6) BYTE,
        VERT3 (6) BYTE,
        N1_STRING_LABEL (18) BYTE,
        N2_STRING_LABEL (18) BYTE,
        N1_STRING (13) BYTE,
        TIL_DRES (13) BYTE,
        VERT4 (6) BYTE,
        VERT5 (6) BYTE,
        VERT6 (6) BYTE,
        VERT7 (6) BYTE,
        DIVIDER (69) BYTE,
        WHEEL_NAME_STRING (13) BYTE,
        WHEEL_SPEED_STRING (10) BYTE,
        WHEEL_DIAM_STRING (13) BYTE,
        COMPONENT_LABEL (16) BYTE,
        VERT8 (6) BYTE,
        CMPT_LABEL1 (50) BYTE,
        CMPT_LABEL2 (37) BYTE,
        CMPT_LABEL3 (28) BYTE,
        CMPT_NAME_STRING (11) BYTE,
        CMPT_FD_STRING (13) BYTE,
        CMPT_SD_STRING (13) BYTE,
        CMPT_ID_STRING (13) BYTE,
        CMPT_LEN_STRING (13) BYTE,
        CMPT_MATL_STRING (11) BYTE,
        DIVIDER2 (69) BYTE,
        OPT_LABEL (13) BYTE,
        VERT9 (6) BYTE,
        OPT_LABEL2 (14) BYTE,
        OPT_LABEL3 (14) BYTE,
        OPT_LABEL4 (16) BYTE,
        OPT_LABEL5 (14) BYTE,
        OPT_LABEL6 (16) BYTE,
        OPT_LABEL7 (14) BYTE)

DATA (1,0,0FFH,39,
3,0,247,11,0,' GDG WHEEL ',
3,11,255,1,0,'|',
3,12,255,17,0,'  VMAX      DIAM  ',
4,12,255,17,0,'   m/s       (mm)  ',
3,41,247,5,0,' M/C ',
3,40,255,1,0,'|',
3,46,255,1,0,'|',
5,41,255,13,0,' CYCLE NO     ',
6,41,255,13,0,' DRESS IN     ',
5,55,255,8,0,'11111111',
6,55,255,8,0,'11111111',
4,40,255,1,0,'|',
5,40,255,1,0,'|',
6,40,255,1,0,'|',
7,40,255,1,0,'|',
8,0,247,64,0,'
                ',
6,1,255,8,0,'AAAAAAAA',
6,14,255,5,0,'11111',
6,21,255,8,0,'11111111',
        9,0,247,11,0,' COMPONENT ',
        9,11,255,1,0,'|',
        9,12,255,45,0,' FINAL    START    INTRN    LENGTH
MATERIAL',
        10,12,255,32,0,' DIAM     DIAM     DIAM     (mm)',
        11,12,255,23,0,' (mm)     (mm)     (mm)',
        13,1,255,6,0,'AAAAAA',
        13,13,255,8,0,'11111111',
        13,22,255,8,0,'11111111',
        13,31,255,8,0,'11111111',
        13,40,255,8,0,'11111111',
        13,49,255,6,0,'AAAAAA',
        15,0,247,64,0,'
',
        16,0,247,8,0,' OPTION ',
        16,8,255,1,0,'|',
        18,12,255,9,0,' 1. GRIND',
        20,12,255,9,0,' 2. EDIT ',
        22,12,255,11,0,' 3. SET BED',
        18,27,255,9,0,' 4. DRESS',
        20,27,255,11,0,' 5. NEW BAT',
        22,27,255,9,0,' 6. QUIT ');

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

DISP1:
        PROCEDURE PUBLIC;
/*-------------------------------------------------*/

SK(631) = 00H;
SK(632) = 00H;
SK(633) = 00H;
SK(634) = 00H;
SK(636) = 00H;
SK(606) = 00H;

IF DISP1_DATA_LENGTH = 0 THEN DO:
        DISP1_DATA_LENGTH = SIZE(GRIND1_DISP_INIT);
        CALL
MOVB(@GRIND1_DISP_INIT,@GRIND1_DISP,DISP1_DATA_LENGTH);

        DISP_NUM = 3;
        GRIND1_DISP.CONTROL_WORD = 8001H;
        GRIND1_DISP.REAL_TIME_ROUT_PTR = @REAL_TIME_DISP1;
        CALL DEF_DISP(DISP_NUM,@GRIND1_DISP,@STATUS_D);

        END;

IF SK(600) = 0FFH THEN

DO;

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

CALL SCONNECT(@DEVICE,@CHANNEL,@STAT);

IF STAT <> 0 THEN
        SK(631) = (SK(631) OR 01H);

IF STAT = 0 THEN DO;

/*-------------------------------------------------
        OPEN  MC DATA FILE NO.1
-------------------------------------------------*/

CALL SOPEN(CHANNEL(0),@STAT,READ,@MC_DATA1_FILE(0),RAN,0);

IF STAT <> 0 AND STAT <> -24 THEN
        SK(631) = (SK(631) OR 02H);

IF STAT = -24 THEN DO;
        CALL
SFORMAT(CHANNEL(0),@MC_DATA1_FILE(0),@MC_DATA1_FORMAT,33,1,@ST
AT);
        CALL
SOPEN(CHANNEL(0),@STAT,RE_WR,@MC_DATA1_FILE(0),RAN,0);

IF STAT <> 0 THEN
        SK(631) = (SK(631) OR 02H);

IF STAT = 0 THEN DO;

/*-------------------------------------------------
        INITIALISE DEFAULT VALUES
-------------------------------------------------*/

CALL MOVB(@MC_DATA1_DEFAULT.CMPT_NO,@MC_DATA1.CMPT_NO,2);
CALL MOVB(@MC_DATA1_DEFAULT.WHEEL_NO,@MC_DATA1.WHEEL_NO,2);
CALL R2LR3(@MC_DATA1_DEFAULT.N1,@MC_DATA1.N1);
CALL R2LR3(@MC_DATA1_DEFAULT.N2,@MC_DATA1.N2);
CALL R2LR3(@MC_DATA1_DEFAULT.N3,@MC_DATA1.N3);
CALL R2LR3(@MC_DATA1_DEFAULT.BED_LEN,@MC_DATA1.BED_LEN);
CALL
R2LR3(@MC_DATA1_DEFAULT.SAFETY_DIST,@MC_DATA1.SAFETY_DIST);
CALL R2LR3(@MC_DATA1_DEFAULT.CYCLE_DIST,@MC_DATA1.CYCLE_DIST);
CALL R2LR3(@MC_DATA1_DEFAULT.CW_DIAM,@MC_DATA1.CW_DIAM);
CALL R2LR3(@MC_DATA1_DEFAULT.DWELL,@MC_DATA1.DWELL);
CALL R2LR3(@MC_DATA1_DEFAULT.HOME,@MC_DATA1.HOME);
CALL R2LR3(@MC_DATA1_DEFAULT.SAFETY,@MC_DATA1.SAFETY);
```

```
CALL R2LR3(@MC_DATA1_DEFAULT.STRT,@MC_DATA1.STRT);
CALL R2LR3(@MC_DATA1_DEFAULT.TARGET,@MC_DATA1.TARGET);
CALL R2LR3(@MC_DATA1_DEFAULT.OFFSET,@MC_DATA1.OFFSET);
CALL R2LR3(@MC_DATA1_DEFAULT.ALLOW,@MC_DATA1.ALLOW);

CALL
SPUT(CHANNEL(0),@MC_DATA1,MC_DATA1_LEN,@STAT,@IOSTAT,00H,1,00H
);

IF STAT <> 0 THEN SK(631) = (SK(631) OR 04H);

/*-------------------------------------------------
          WARN PART-PROGRAM OF FILE INIT.
-------------------------------------------------*/

          SK(636) = (SK(636) OR 01H);

END;
END;

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

IF STAT = 0 THEN DO;
          CALL
SGET(CHANNEL(0),@MC_DATA1,MC_DATA1_LEN,@STAT,@IOSTAT,0FFH,1,0)
;

          IF STAT <> 0 THEN
                    SK(631) = (SK(631) OR 08H);
          END;

CALL SCLOSE(CHANNEL(0),@STAT);

/*-------------------------------------------------
          OPEN  MC DATA FILE NO.2
-------------------------------------------------*/

CALL SOPEN(CHANNEL(0),@STAT,READ,@MC_DATA2_FILE(0),RAN,0);

IF STAT <> 0 AND STAT <> -24 THEN
          SK(631) = (SK(631) OR 10H);

IF STAT = -24 THEN DO;

          CALL
SFORMAT(CHANNEL(0),@MC_DATA2_FILE(0),@MC_DATA2_FORMAT,17,1,@ST
AT);
          CALL
SOPEN(CHANNEL(0),@STAT,RE_WR,@MC_DATA2_FILE(0),RAN,0);

IF STAT <> 0 THEN
          SK(631) = (SK(631) OR 10H);

IF STAT = 0 THEN DO;

/*-------------------------------------------------
          INITIALISE DEFAULT VALUES
-------------------------------------------------*/

CALL R2LR3(@MC_DATA2_DEFAULT.VF_RELAX,@MC_DATA2.VF_RELAX);
CALL R2LR3(@MC_DATA2_DEFAULT.VW_RELAX,@MC_DATA2.VW_RELAX);
CALL R2LR3(@MC_DATA2_DEFAULT.MIN_REVS,@MC_DATA2.MIN_REVS);
CALL R2LR3(@MC_DATA2_DEFAULT.Q_MIN,@MC_DATA2.Q_MIN);
CALL R2LR3(@MC_DATA2_DEFAULT.P_RELAX,@MC_DATA2.P_RELAX);
CALL R2LR3(@MC_DATA2_DEFAULT.SIZE_RELAX,@MC_DATA2.SIZE_RELAX);
CALL R2LR3(@MC_DATA2_DEFAULT.TOR_RELAX,@MC_DATA2.TOR_RELAX);
CALL R2LR3(@MC_DATA2_DEFAULT.P_LIMIT_K,@MC_DATA2.P_LIMIT_K);

CALL
SPUT(CHANNEL(0),@MC_DATA2,MC_DATA2_LEN,@STAT,@IOSTAT,00H,1,00H
);

IF STAT <> 0 THEN SK(631) = (SK(631) OR 20H);

/*-------------------------------------------------
          WARN PART-PROGRAM OF FILE INIT.
-------------------------------------------------*/

          SK(636) = (SK(636) OR 02H);

END;
END;

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

IF STAT = 0 THEN DO;
          CALL
SGET(CHANNEL(0),@MC_DATA2,MC_DATA2_LEN,@STAT,@IOSTAT,0FFH,1,0)
;

          IF STAT <> 0 THEN
                    SK(631) = (SK(631) OR 40H);
          END;

CALL SCLOSE(CHANNEL(0),@STAT);

/*-------------------------------------------------
          OPEN  MC DATA FILE NO.3
-------------------------------------------------*/

CALL SOPEN(CHANNEL(0),@STAT,READ,@MC_DATA3_FILE(0),RAN,0);

IF STAT <> 0 AND STAT <> -24 THEN
          SK(634) = (SK(634) OR 01H);

IF STAT = -24 THEN DO;

          CALL
SFORMAT(CHANNEL(0),@MC_DATA3_FILE(0),@MC_DATA3_FORMAT,17,1,@ST
AT);
          CALL
SOPEN(CHANNEL(0),@STAT,RE_WR,@MC_DATA3_FILE(0),RAN,0);

IF STAT <> 0 THEN
          SK(634) = (SK(634) OR 01H);

IF STAT = 0 THEN DO;

/*-------------------------------------------------
          INITIALISE DEFAULT VALUES
-------------------------------------------------*/

CALL
MOVB(@MC_DATA3_DEFAULT.MAN_OR_SWITCH,@MC_DATA3.MAN_OR_SWITCH,6
);
CALL
MOVB(@MC_DATA3_DEFAULT.ADAPT_OR_SWITCH,@MC_DATA3.ADAPT_OR_SWIT
CH,6);
CALL
MOVB(@MC_DATA3_DEFAULT.OS_OR_SWITCH,@MC_DATA3.OS_OR_SWITCH,6);
CALL
MOVB(@MC_DATA3_DEFAULT.PECK_OR_SWITCH,@MC_DATA3.PECK_OR_SWITCH
,6);

CALL R2LR3(@MC_DATA3_DEFAULT.OR_VS,@MC_DATA3.OR_VS);
CALL R2LR3(@MC_DATA3_DEFAULT.OR_VW,@MC_DATA3.OR_VW);
CALL R2LR3(@MC_DATA3_DEFAULT.OR_VF,@MC_DATA3.OR_VF);
CALL R2LR3(@MC_DATA3_DEFAULT.OR_OS,@MC_DATA3.OR_OS);


CALL
SPUT(CHANNEL(0),@MC_DATA3,MC_DATA3_LEN,@STAT,@IOSTAT,00H,1,00H
);

IF STAT <> 0 THEN SK(634) = (SK(634) OR 02H);

/*-------------------------------------------------
          WARN PART-PROGRAM OF FILE INIT.
-------------------------------------------------*/

          SK(636) = (SK(636) OR 04H);

END;
END;

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

IF STAT = 0 THEN DO;
          CALL
SGET(CHANNEL(0),@MC_DATA3,MC_DATA3_LEN,@STAT,@IOSTAT,0FFH,1,0)
;

          IF STAT <> 0 THEN
                    SK(634) = (SK(634) OR 04H);
          END;

CALL SCLOSE(CHANNEL(0),@STAT);

/*-------------------------------------------------
          OPEN COMPONENT FILE
-------------------------------------------------*/

CALL SOPEN(CHANNEL(0),@STAT,READ,@PART_FILE(0),RAN,0);

IF STAT <> 0 AND STAT <> -24 THEN
          SK(632) = (SK(632) OR 01H);

IF STAT = -24 THEN DO;

          CALL
SFORMAT(CHANNEL(0),@PART_FILE(0),@PART_FORMAT,15,10,@STAT);
          CALL
SOPEN(CHANNEL(0),@STAT,RE_WR,@PART_FILE(0),RAN,0);

IF STAT <> 0 THEN
          SK(632) = (SK(632) OR 01H);

IF STAT = 0 THEN DO;

/*-------------------------------------------------
          INITIALISE DEFAULT VALUES
-------------------------------------------------*/

CALL MOVB(@PART_DEFAULT.NAME,@PART.NAME,12);
CALL R2LR3(@PART_DEFAULT.FINAL_DIM,@PART.FINAL_DIM);
CALL R2LR3(@PART_DEFAULT.START_DIM,@PART.START_DIM);
```

```
 CALL R2LR3(@PART_DEFAULT.INTRN_DIM,@PART.INTRN_DIM);
 CALL R2LR3(@PART_DEFAULT.CMPT_LEN,@PART.CMPT_LEN);
 CALL R2LR3(@PART_DEFAULT.TOR,@PART.TOR);
 CALL MOVB(@PART_DEFAULT.MATL_NO,@PART.MATL_NO,2);

 INDEX = 1;
 DO WHILE INDEX < 11;

 CALL
 SPUT(CHANNEL(0),@PART,PART_LEN,@STAT,@IOSTAT,00H,INDEX,00H);

 IF STAT <> 0 THEN DO;
          SR(632) = (SR(632) OR 02H);
          INDEX = 11;
          END;

          INDEX=INDEX+1;
          END;
END;

/*-------------------------------------------------
          WARN PART-PROGRAM OF FILE INIT.
-------------------------------------------------*/

          SR(636) = (SR(636) OR 08H);

END;

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

IF STAT = 0 THEN DO;
          INDEX = UNSIGN (MC_DATA1.CMPT_NO);
          CALL
SGET(CHANNEL(0),@PART,PART_LEN,@STAT,@IOSTAT,0FFH,INDEX,0);

          IF STAT <> 0 THEN
                    SR(632) = (SR(632) OR 04H);

          END;

CALL SCLOSE(CHANNEL(0),@STAT);

/*-------------------------------------------------
          OPEN MATERIAL FILE
-------------------------------------------------*/

CALL SOPEN(CHANNEL(0),@STAT,READ,@MATL_FILE(0),RAN,0);

IF STAT <> 0 AND STAT <> -24 THEN
          SR(632) = (SR(632) OR 08H);

IF STAT = -24 THEN DO;
          CALL
SFORMAT(CHANNEL(0),@MATL_FILE(0),@MATL_FORMAT,15,10,@STAT);
          CALL
SOPEN(CHANNEL(0),@STAT,RE_WR,@MATL_FILE(0),RAN,0);

IF STAT <> 0 THEN
          SR(632) = (SR(632) OR 08H);

IF STAT = 0 THEN DO;

/*-------------------------------------------------
          INITIALISE DEFAULT VALUES
-------------------------------------------------*/

CALL MOVB(@MATL_DEFAULT.NAME,@MATL.NAME,6);
CALL R2LR3(@MATL_DEFAULT.T_CRIT,@MATL.T_CRIT);
CALL R2LR3(@MATL_DEFAULT.DIFF,@MATL.DIFF);
CALL R2LR3(@MATL_DEFAULT.T_COND,@MATL.T_COND);
CALL R2LR3(@MATL_DEFAULT.VS,@MATL.VS);
CALL R2LR3(@MATL_DEFAULT.AR,@MATL.AR);
CALL R2LR3(@MATL_DEFAULT.VM,@MATL.VM);

INDEX = 1;
DO WHILE INDEX < 11;

CALL
SPUT(CHANNEL(0),@MATL,MATL_LEN,@STAT,@IOSTAT,00H,INDEX,00H);

IF STAT <> 0 THEN DO;
          SR(632) = (SR(632) OR 10H);
          INDEX = 11;
          END;

          INDEX=INDEX+1;
          END;
END;

/*-------------------------------------------------
          WARN PART-PROGRAM OF FILE INIT.
-------------------------------------------------*/
          SR(636) = (SR(636) OR 10H);

END;

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

IF STAT = 0 THEN DO;
          INDEX = UNSIGN (PART.MATL_NO);
          CALL
SGET(CHANNEL(0),@MATL,MATL_LEN,@STAT,@IOSTAT,0FFH,INDEX,0);

          IF STAT <> 0 THEN
                    SR(632) = (SR(632) OR 20H);

          END;

CALL SCLOSE(CHANNEL(0),@STAT);

/*-------------------------------------------------
          OPEN WHEEL DATA FILE
-------------------------------------------------*/

CALL SOPEN(CHANNEL(0),@STAT,READ,@WHEEL_FILE(0),RAN,0);

IF STAT <> 0 AND STAT <> -24 THEN
          SR(633) = (SR(633) OR 01H);

IF STAT = -24 THEN DO;

          CALL
SFORMAT(CHANNEL(0),@WHEEL_FILE(0),@WHEEL_FORMAT,19,3,@STAT);
          CALL
SOPEN(CHANNEL(0),@STAT,RE_WR,@WHEEL_FILE(0),RAN,0);

          IF STAT <> 0 THEN
                    SR(633) = (SR(633) OR 01H);

IF STAT = 0 THEN DO;

/*-------------------------------------------------
          INITIALISE DEFAULT VALUES
-------------------------------------------------*/

CALL MOVB(@WHEEL_DEFAULT.NAME,@WHEEL.NAME,8);
CALL R2LR3(@WHEEL_DEFAULT.MAX_SPEED,@WHEEL.MAX_SPEED);
CALL R2LR3(@WHEEL_DEFAULT.GRIT_SIZE,@WHEEL.GRIT_SIZE);
CALL R2LR3(@WHEEL_DEFAULT.INITIAL_DIAM,@WHEEL.INITIAL_DIAM);
CALL R2LR3(@WHEEL_DEFAULT.CURRENT_DIAM,@WHEEL.CURRENT_DIAM);
CALL R2LR3(@WHEEL_DEFAULT.FINAL_DIAM,@WHEEL.FINAL_DIAM);
CALL R2LR3(@WHEEL_DEFAULT.WIDTH,@WHEEL.WIDTH);
CALL R2LR3(@WHEEL_DEFAULT.DIFF,@WHEEL.DIFF);
CALL R2LR3(@WHEEL_DEFAULT.COND,@WHEEL.COND);

INDEX = 1;
DO WHILE INDEX < 4;

CALL
SPUT(CHANNEL(0),@WHEEL,WHEEL_LEN,@STAT,@IOSTAT,00H,INDEX,00H);

IF STAT <> 0 THEN DO;
          SR(633) = (SR(633) OR 02H);
          INDEX = 10;
          END;

          INDEX=INDEX+1;
          END;
END;

/*-------------------------------------------------
          WARN PART-PROGRAM OF FILE INIT.
-------------------------------------------------*/
          SR(636) = (SR(636) OR 20H);

END;

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

IF STAT = 0 THEN DO;
          INDEX = UNSIGN (MC_DATA1.WHEEL_NO);
          CALL
SGET(CHANNEL(0),@WHEEL,WHEEL_LEN,@STAT,@IOSTAT,0FFH,INDEX,0);

          IF STAT <> 0 THEN
                    SR(633) = (SR(633) OR 04H);

          END;

CALL SCLOSE(CHANNEL(0),@STAT);

/*-------------------------------------------------
          OPEN PASS  WORD DATA FILE
          AND STORE IN NEW VARIABLE 'PASS'
-------------------------------------------------*/

CALL SOPEN(CHANNEL(0),@STAT,READ,@PASS_FILE(0),RAN,0);

IF STAT <> 0 AND STAT <> -24 THEN
          SR(633) = (SR(633) OR 08H);

IF STAT = -24 THEN DO;
          CALL
SFORMAT(CHANNEL(0),@PASS_FILE(0),@PASS_WORD_FORMAT,3,3,@STAT);
          CALL
SOPEN(CHANNEL(0),@STAT,RE_WR,@PASS_FILE(0),RAN,0);

IF STAT <> 0 THEN
```

```
            SR(633) = (SR(633) OR 08H);

 IF STAT = 0 THEN DO;

 INDEX = 1;
 DO WHILE INDEX < 4;

 CALL MOVB(@INIT_PASS(8*(INDEX-1)),@PASS_WORD.LEVEL,8);
 CALL
 SPUT(CHANNEL(0),@PASS_WORD,PASS_LEN,@STAT,@IOSTAT,00H,INDEX,00
 H);

 IF STAT <> 0 THEN DO;
            SR(633) = (SR(633) OR 10H);
            INDEX = 10;
            END;

            INDEX=INDEX+1;
            END;
 END;

 /*-----------------------------------------------
            WARN PART-PROGRAM OF FILE INIT.
 -----------------------------------------------*/
            SR(636) = (SR(636) OR 40H);

 END;

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

 IF STAT = 0 THEN DO;

 INDEX=1;
 DO WHILE INDEX < 4;
            CALL
 SGET(CHANNEL(0),@PASS_WORD,PASS_LEN,@STAT,@IOSTAT,0FFH,INDEX,0
 );

            IF STAT <> 0 THEN
                        SR(633) = (SR(633) OR 020H);

            IF STAT = 0 THEN DO;
                        CALL
 MOVB(@PASS_WORD.LEVEL,@PASS(10*(INDEX)),8);
                        CALL MOVB(@BLANKS,@PASS(10*(INDEX)+8),2);
                        END;

            INDEX = INDEX+1;
            END;

            END;

 CALL SCLOSE(CHANNEL(0),@STAT);

 /*-----------------------------------------------
            OPEN CYCLE DATA FILE
 -----------------------------------------------*/

 CALL SOPEN(CHANNEL(0),@STAT,READ,@CYCLE_FILE(0),RAN,0);

 IF STAT <> 0 AND STAT <> -24 THEN
            SR(633) = (SR(633) OR 01H);

 IF STAT = -24 THEN DO;
            CALL
 SFORMAT(CHANNEL(0),@CYCLE_FILE(0),@CYCLE_FORMAT,17,10,@STAT);
            CALL
 SOPEN(CHANNEL(0),@STAT,RE_WR,@CYCLE_FILE(0),RAN,0);

            IF STAT <> 0 THEN
                        SR(633) = (SR(633) OR 01H);

 IF STAT = 0 THEN DO;

 /*-----------------------------------------------
            INITIALISE DEFAULT VALUES
 -----------------------------------------------*/

 INDEX = 0;
 DO WHILE INDEX < 3;

 CALL R2LR3(@CYCLE_DEFAULT(INDEX).ALLOW,@CYCLE(INDEX).ALLOW);
 CALL R2LR3(@CYCLE_DEFAULT(INDEX).TOR,@CYCLE(INDEX).TOR);

 INDEX=INDEX+1;
 END;

 INDEX = 1;
 DO WHILE INDEX < 11;

 CALL
 SPUT(CHANNEL(0),@CYCLE,CYCLE_LEN,@STAT,@IOSTAT,00H,INDEX,00H);

 IF STAT <> 0 THEN DO;
            SR(633) = (SR(633) OR 02H);
            INDEX = 10;
            END;

            INDEX=INDEX+1;
            END;
 END;

 /*-----------------------------------------------
            WARN PART-PROGRAM OF FILE INIT.
 -----------------------------------------------*/
            SR(636) = (SR(636) OR 80H);

 END;

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

 IF STAT = 0 THEN DO;
            INDEX = UNSIGN (MC_DATA1.CMPT_NO);
            CALL
 SGET(CHANNEL(0),@CYCLE,CYCLE_LEN,@STAT,@IOSTAT,0FFH,INDEX,0);

            IF STAT <> 0 THEN
                        SR(633) = (SR(633) OR 04H);

            END;

 CALL SCLOSE(CHANNEL(0),@STAT);

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

 END;

 CALL SDCONNECT (CHANNEL(0),@STAT);

 /*-----------------------------------------------
 SR626=1 ALLOWS RESETTING OF PARAMS WITHOUT DISPLAY
 -----------------------------------------------*/
 IF SR(626)=0 THEN CALL SET_DISP(DISP_NUM,WAIT_EXE,@STATUS_R);

 IF SR(631) = 0 AND SR(632) = 0 AND SR(633) = 0 AND SR(634) = 0
 THEN SR(606) = OFFH;

 END;

 END DISP1;

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

 REAL_TIME_DISP1:
            PROCEDURE PUBLIC;

 DECLARE LR_TEMP (8) BYTE;

 DO;
            CALL MOVB(@BLANKS,@ASC_BUFFER,20);
            CALL MQLRASC(@MC_DATA1.W1,@ASC_BUFFER,@STAT4,20,0);
            CALL MOVB(@BLANKS,@GRIND1_DISP.W1_STRING(5),8);
            CALL
 MOVB(@ASC_BUFFER(1),@GRIND1_DISP.W1_STRING(5),8);
            CALL
 CHANGE_ITEM(DISP_NUM,WAIT_EXE,@GRIND1_DISP.W1_STRING,@STATUS_C
 );

            CALL LR_SUB(@MC_DATA1.W3,@MC_DATA1.W2,@LR_TEMP);

            CALL MOVB(@BLANKS,@ASC_BUFFER,20);
            CALL MQLRASC(@LR_TEMP,@ASC_BUFFER,@STAT4,20,0);
            CALL MOVB(@BLANKS,@GRIND1_DISP.TIL_DRES(5),8);
            CALL
 MOVB(@ASC_BUFFER(1),@GRIND1_DISP.TIL_DRES(5),8);
            CALL
 CHANGE_ITEM(DISP_NUM,WAIT_EXE,@GRIND1_DISP.TIL_DRES,@STATUS_C)
 ;


            CALL MOVB(@BLANKS,@ASC_BUFFER,20);
            CALL
 MQLRASC(@WHEEL.CURRENT_DIAM,@ASC_BUFFER,@STAT4,20,3);
            CALL
 MOVB(@BLANKS,@GRIND1_DISP.WHEEL_DIAM_STRING(5),8);
            CALL
 MOVB(@ASC_BUFFER(1),@GRIND1_DISP.WHEEL_DIAM_STRING(5),8);
            CALL
 CHANGE_ITEM(DISP_NUM,WAIT_EXE,@GRIND1_DISP.WHEEL_DIAM_STRING,@
 STATUS_C);

            CALL
 MOVB(@PART.NAME,@GRIND1_DISP.CMPT_NAME_STRING(5),6);
            CALL
 CHANGE_ITEM(DISP_NUM,WAIT_EXE,@GRIND1_DISP.CMPT_NAME_STRING,@S
 TATUS_C);

            CALL MOVB(@BLANKS,@ASC_BUFFER,20);
            CALL
 MQLRASC(@PART.FINAL_DIM,@ASC_BUFFER,@STAT4,20,3);
            CALL MOVB(@BLANKS,@GRIND1_DISP.CMPT_FD_STRING(5),8);
            CALL
 MOVB(@ASC_BUFFER(1),@GRIND1_DISP.CMPT_FD_STRING(5),8);
```

```
            CALL
CHANGE_ITEM(DISP_NUM,WAIT_EXE,@GRIND1_DISP.CMPT_FD_STRING,@STA
TUS_C);

            CALL MOVB(@BLANKS,@ASC_BUFFER,20);
            CALL
MQLRASC(@PART.START_DIM,@ASC_BUFFER,@STAT4,20,3);
            CALL MOVB(@BLANKS,@GRIND1_DISP.CMPT_SD_STRING(5),8);
            CALL
MOVB(@ASC_BUFFER(1),@GRIND1_DISP.CMPT_SD_STRING(5),8);
            CALL
CHANGE_ITEM(DISP_NUM,WAIT_EXE,@GRIND1_DISP.CMPT_SD_STRING,@STA
TUS_C);

            CALL MOVB(@BLANKS,@ASC_BUFFER,20);
            CALL
MQLRASC(@PART.INTRM_DIM,@ASC_BUFFER,@STAT4,20,3);
            CALL MOVB(@BLANKS,@GRIND1_DISP.CMPT_ID_STRING(5),8);
            CALL
MOVB(@ASC_BUFFER(1),@GRIND1_DISP.CMPT_ID_STRING(5),8);
            CALL
CHANGE_ITEM(DISP_NUM,WAIT_EXE,@GRIND1_DISP.CMPT_ID_STRING,@STA
TUS_C);

            CALL MOVB(@BLANKS,@ASC_BUFFER,20);
            CALL
MQLRASC(@PART.CMPT_LEN,@ASC_BUFFER,@STAT4,20,3);
            CALL
MOVB(@BLANKS,@GRIND1_DISP.CMPT_LEN_STRING(5),8);
            CALL
MOVB(@ASC_BUFFER(1),@GRIND1_DISP.CMPT_LEN_STRING(5),8);
            CALL
CHANGE_ITEM(DISP_NUM,WAIT_EXE,@GRIND1_DISP.CMPT_LEN_STRING,@ST
ATUS_C);

            CALL
MOVB(@MATL.NAME,@GRIND1_DISP.CMPT_MATL_STRING(5),6);
            CALL
CHANGE_ITEM(DISP_NUM,WAIT_EXE,@GRIND1_DISP.CMPT_MATL_STRING,@S
TATUS_C);

            CALL
MOVB(@WHEEL.NAME,@GRIND1_DISP.WHEEL_NAME_STRING(5),8);
            CALL
CHANGE_ITEM(DISP_NUM,WAIT_EXE,@GRIND1_DISP.WHEEL_NAME_STRING,@
STATUS_C);

            CALL MOVB(@BLANKS,@ASC_BUFFER,20);
            CALL
MQLRASC(@WHEEL.MAX_SPEED,@ASC_BUFFER,@STAT4,20,0);
            CALL
MOVB(@BLANKS,@GRIND1_DISP.WHEEL_SPEED_STRING(5),5);
            CALL
MOVB(@ASC_BUFFER(1),@GRIND1_DISP.WHEEL_SPEED_STRING(5),5);
            CALL
CHANGE_ITEM(DISP_NUM,WAIT_EXE,@GRIND1_DISP.WHEEL_SPEED_STRING,
@STATUS_C);

END;

END REAL_TIME_DISP1;

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

            READS COMPONENT DATA BASE AND
            CREATES NEW IF NON-EXISTANT

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

READ_CMPT_FILE:
            PROCEDURE PUBLIC;

SR(604) = (SR(604) AND 0H);
SR(605) = (SR(605) AND 0H);
SR(606) = (SR(606) AND 0H);

/*-----------------------------------------------
            CONNECT CHANNEL
-----------------------------------------------*/
CALL SCONNECT(@DEVICE,@CHANNEL,@STAT);

IF STAT <> 0 THEN
            SR(604) = (SR(604) OR 01H);

IF STAT = 0 THEN

/*-----------------------------------------------
            OPEN FILE
-----------------------------------------------*/
DO;

REC_COUNT = 0;

DO WHILE REC_COUNT < 10;

/*-----------------------------------------------*/
CALL SOPEN(CHANNEL(0),@STAT,READ,@PART_FILE(0),RAN,0);

IF STAT = -24 THEN
            SR(604) = (SR(604) OR 02H);

IF STAT <> 0 THEN
            SR(604) = (SR(604) OR 04H);

IF STAT = 0 THEN
            CALL
SGET(CHANNEL(0),@PART,PART_LEN,@STAT,@IOSTAT,0FFH,REC_COUNT+1,
0);

IF STAT <> 0 THEN DO;
            SR(604) = (SR(604) OR 08H);
            REC_COUNT = 10;
            END;

CALL SCLOSE(CHANNEL(0),@STAT);

/*-----------------------------------------------*/
CALL SOPEN(CHANNEL(0),@STAT,READ,@MATL_FILE(0),RAN,0);

IF STAT <> 0 THEN
            SR(604) = (SR(604) OR 04H);

INDEX = UNSIGN (PART.MATL_NO);

IF STAT = 0 THEN
            CALL
SGET(CHANNEL(0),@MATL,MATL_LEN,@STAT,@IOSTAT,0FFH,INDEX,0);

IF STAT <> 0 THEN DO;
            SR(604) = (SR(604) OR 08H);
            REC_COUNT = 10;
            END;

CALL SCLOSE(CHANNEL(0),@STAT);

/*-----------------------------------------------*/
IF STAT = 0 THEN DO;
            CALL MOVB(@PART.NAME,@SYVAR(6*REC_COUNT),6);
            CALL
MOVB(@PART.FINAL_DIM,@E_PAR(30*REC_COUNT).VAL,8);
            CALL
MOVB(@PART.START_DIM,@E_PAR(40*REC_COUNT).VAL,8);
            CALL
MOVB(@PART.INTRM_DIM,@E_PAR(50*REC_COUNT).VAL,8);
            CALL
MOVB(@PART.CMPT_LEN,@E_PAR(60*REC_COUNT).VAL,8);
            CALL MOVB(@PART.TOR,@E_PAR(70*REC_COUNT).VAL,8);
            CALL MOVB(@MATL.NAME,@SYVAR(60+6*REC_COUNT),6);
            CALL MOVB(@PART.MATL_NO,@SYVAR(100+2*REC_COUNT),2);
            REC_COUNT = REC_COUNT + 1;
            END;

/*-----------------------------------------------*/
END;

CALL SDCONNECT (CHANNEL(0),@STAT);

END;

IF SR(604) = 0 AND SR(605) = 0 THEN SR(606) = 255;

END READ_CMPT_FILE;

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

            PROCEDURE TO WRITE ALL THE APPROPRIATE
            E'S AND SYVAR'S TO THE DATA BASE

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

SAVE_CMPT_FILE:
            PROCEDURE PUBLIC;

SR(604) = (SR(604) AND 00H);
SR(605) = (SR(605) AND 00H);
SR(606) = (SR(606) AND 00H);

/*-----------------------------------------------
            CONNECT CHANNEL
-----------------------------------------------*/

CALL SCONNECT(@DEVICE,@CHANNEL,@STAT);

IF STAT <> 0 THEN
            SR(604) = (SR(604) OR 01H);

IF STAT = 0 THEN DO;

/*-----------------------------------------------
            OPEN FILE
-----------------------------------------------*/

CALL SOPEN(CHANNEL(0),@STAT,RE_WR,@PART_FILE(0),RAN,0);

IF STAT = -24 THEN
            SR(604) = (SR(604) OR 02H);
```

```
IF STAT <> 0 THEN
          SK(604) = (SK(604) OR 04H);

IF STAT = 0 THEN DO;

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

REC_COUNT = 0;

DO WHILE REC_COUNT < 10;

          CALL
SGET(CHANNEL(0),@PART,PART_LEN,@STAT,@IOSTAT,0FFH,REC_COUNT+1,
0);

          CALL MOVB(@SYVAR(6*REC_COUNT),@PART.NAME,6);
          CALL
MOVB(@E_PAR(30+REC_COUNT).VAL,@PART.FINAL_DIM,8);
          CALL
MOVB(@E_PAR(40+REC_COUNT).VAL,@PART.START_DIM,8);
          CALL
MOVB(@E_PAR(50+REC_COUNT).VAL,@PART.INTRN_DIM,8);
          CALL
MOVB(@E_PAR(60+REC_COUNT).VAL,@PART.CMPT_LEN,8);
          CALL MOVB(@E_PAR(70+REC_COUNT).VAL,@PART.TOR,8);
          CALL MOVB(@SYVAR(180+2*REC_COUNT),@PART.MATL_NO,2);

          CALL SUPDATE(CHANNEL(0),@PART,PART_LEN,@STAT);

          IF STAT <> 0 THEN DO;
                    SK(604) = (SK(604) OR 10H);
                    REC_COUNT = 10;
                    END;

          IF STAT = 0 THEN
                    REC_COUNT = REC_COUNT + 1;

          END;

END;

CALL SCLOSE(CHANNEL(0),@STAT);

END;

CALL SDCONNECT (CHANNEL(0),@STAT);

IF SK(604) = 0 AND SK(605) = 0 THEN SK(606) = 255;

END SAVE_CMPT_FILE;

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

          PROCCY TO READ THE MATERIALS DATA BASE

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

READ_MATL_FILE:
          PROCEDURE PUBLIC;

SK(604) = (SK(604) AND 0H);
SK(605) = (SK(605) AND 0H);
SK(606) = (SK(606) AND 0H);

/*-------------------------------------------------
          CONNECT CHANNEL
-------------------------------------------------*/
CALL SCONNECT(@DEVICE,@CHANNEL,@STAT);

IF STAT <> 0 THEN
          SK(604) = (SK(604) OR 01H);

IF STAT = 0 THEN

/*-------------------------------------------------
          OPEN FILE
-------------------------------------------------*/
DO;
          CALL
SOPEN(CHANNEL(0),@STAT,READ,@MATL_FILE(0),RAN,0);

          IF STAT = -24 THEN
                    SK(604) = (SK(604) OR 02H);

          IF STAT <> 0 THEN
                    SK(604) = (SK(604) OR 04H);

/*-------------------------------------------------
          LOAD INFO
-------------------------------------------------*/
IF STAT = 0 THEN
DO;

REC_COUNT = 0;

DO WHILE REC_COUNT < 10;

CALL
SGET(CHANNEL(0),@MATL,MATL_LEN,@STAT,@IOSTAT,0FFH,REC_COUNT+1,
0);

IF STAT <> 0 THEN DO;
          SK(604) = (SK(604) OR 08H);
          REC_COUNT = 10;
          END;

IF STAT = 0 THEN DO;
          CALL MOVB(@MATL.NAME,@SYVAR(120+6*REC_COUNT),6);
          CALL MOVB(@MATL.T_CRIT,@E_PAR(80+REC_COUNT).VAL,8);
          CALL MOVB(@MATL.DIFF,@E_PAR(90+REC_COUNT).VAL,8);
          CALL MOVB(@MATL.T_COND,@E_PAR(100+REC_COUNT).VAL,8);
          CALL MOVB(@MATL.VS,@E_PAR(110+REC_COUNT).VAL,8);
          CALL MOVB(@MATL.AR,@E_PAR(120+REC_COUNT).VAL,8);
          CALL MOVB(@MATL.VM,@E_PAR(130+REC_COUNT).VAL,8);

          REC_COUNT = REC_COUNT + 1;

END;

END;

/*-------------------------------------------------
          CLOSE FILE
-------------------------------------------------*/

END;

          CALL SCLOSE(CHANNEL(0),@STAT);
          CALL SDCONNECT (CHANNEL(0),@STAT);

END;

IF SK(604) = 0 AND SK(605) = 0 THEN SK(606) = 255;

END READ_MATL_FILE;

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

          AND SAVE MATERIALS DATA BASE

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

SAVE_MATL_FILE:
          PROCEDURE PUBLIC;

/*-------------------------------------------------
          CONNECT CHANNEL
-------------------------------------------------*/

SK(604) = (SK(604) AND 00H);
SK(605) = (SK(605) AND 00H);
SK(606) = (SK(606) AND 00H);

CALL SCONNECT(@DEVICE,@CHANNEL,@STAT);

IF STAT <> 0 THEN
          SK(604) = (SK(604) OR 01H);

IF STAT = 0 THEN DO;

/*-------------------------------------------------
          OPEN FILE
-------------------------------------------------*/
CALL SOPEN(CHANNEL(0),@STAT,RE_WR,@MATL_FILE(0),RAN,0);

IF STAT = -24 THEN
          SK(604) = (SK(604) OR 02H);

IF STAT <> 0 THEN
          SK(604) = (SK(604) OR 04H);

IF STAT = 0 THEN DO;

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

REC_COUNT = 0;

DO WHILE REC_COUNT < 10;

          CALL
SGET(CHANNEL(0),@MATL,MATL_LEN,@STAT,@IOSTAT,0FFH,REC_COUNT+1,
0);

          CALL MOVB(@SYVAR(120+6*REC_COUNT),@MATL.NAME,6);
          CALL MOVB(@E_PAR(80+REC_COUNT).VAL,@MATL.T_CRIT,8);
          CALL MOVB(@E_PAR(90+REC_COUNT).VAL,@MATL.DIFF,8);
          CALL MOVB(@E_PAR(100+REC_COUNT).VAL,@MATL.T_COND,8);
          CALL MOVB(@E_PAR(110+REC_COUNT).VAL,@MATL.VS,8);
          CALL MOVB(@E_PAR(120+REC_COUNT).VAL,@MATL.AR,8);
          CALL MOVB(@E_PAR(130+REC_COUNT).VAL,@MATL.VM,8);

          CALL SUPDATE(CHANNEL(0),@MATL,MATL_LEN,@STAT);

          IF STAT <> 0 THEN DO;
                    SK(604) = (SK(604) OR 10H);
                    REC_COUNT = 10;
                    END;
```

```
                REC_COUNT = REC_COUNT + 1;
        END;

/*-------------------------------------------------
        CLOSE FILE
-------------------------------------------------*/

END;
        CALL SCLOSE(CHANNEL(0),@STAT);

END;
        CALL SDCONNECT (CHANNEL(0),@STAT);

IF SK(604) = 0 AND SK(605) = 0 THEN SK(606) = 255;

END SAVE_MATL_FILE;

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

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

READ_WHEEL_FILE:
        PROCEDURE PUBLIC;


SK(604) = (SK(604) AND 0H);
SK(605) = (SK(605) AND 0H);
SK(606) = (SK(606) AND 0H);

/*-------------------------------------------------
        CONNECT CHANNEL
-------------------------------------------------*/
CALL SCONNECT(@DEVICE,@CHANNEL,@STAT);

IF STAT <> 0 THEN
        SK(604) = (SK(604) OR 01H);

IF STAT = 0 THEN

/*-------------------------------------------------
        OPEN FILE
-------------------------------------------------*/
DO;
        CALL
SOPEN(CHANNEL(0),@STAT,READ,@WHEEL_FILE(0),RAN,0);

        IF STAT = -24 THEN
                SK(604) = (SK(604) OR 02H);

        IF STAT <> 0 THEN
                SK(604) = (SK(604) OR 04H);

IF STAT = 0 THEN
DO;

REC_COUNT = 0;

DO WHILE REC_COUNT < 3;
CALL
SGET(CHANNEL(0),@WHEEL,WHEEL_LEN,@STAT,@IOSTAT,0FFH,REC_COUNT+
1,0);

IF STAT <> 0 THEN DO;
        SK(604) = (SK(604) OR 08H);
        REC_COUNT = 10;
        END;

IF STAT = 0 THEN DO;
        CALL MOVB(@WHEEL.NAME,@SYVAR(8*REC_COUNT),8);
        CALL
MOVB(@WHEEL.MAX_SPEED,@E_PAR(50+REC_COUNT).VAL,8);
        CALL
MOVB(@WHEEL.GRIT_SIZE,@E_PAR(53+REC_COUNT).VAL,8);
        CALL
MOVB(@WHEEL.INITIAL_DIAM,@E_PAR(56+REC_COUNT).VAL,8);
        CALL
MOVB(@WHEEL.CURRENT_DIAM,@E_PAR(59+REC_COUNT).VAL,8);
        CALL
MOVB(@WHEEL.FINAL_DIAM,@E_PAR(62+REC_COUNT).VAL,8);
        CALL MOVB(@WHEEL.WIDTH,@E_PAR(65+REC_COUNT).VAL,8);
        CALL MOVB(@WHEEL.DIFF,@E_PAR(68+REC_COUNT).VAL,8);
        CALL MOVB(@WHEEL.COND,@E_PAR(71+REC_COUNT).VAL,8);
        REC_COUNT = REC_COUNT + 1;
        END;

END;

/*-------------------------------------------------
        CLOSE FILE
-------------------------------------------------*/

END;

        CALL SCLOSE(CHANNEL(0),@STAT);
        CALL SDCONNECT (CHANNEL(0),@STAT);

END;

IF SK(604) = 0 AND SK(605) = 0 THEN SK(606) = 255;

END READ_WHEEL_FILE;

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

        PROCEDURE TO WRITE ALL THE APPROPRIATE
        E'S AND SYVAR'S TO THE DATA BASE

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

SAVE_WHEEL_FILE:
        PROCEDURE PUBLIC;


SK(604) = (SK(604) AND 00H);
SK(605) = (SK(605) AND 00H);
SK(606) = (SK(606) AND 00H);

/*-------------------------------------------------
        CONNECT CHANNEL
-------------------------------------------------*/

CALL SCONNECT(@DEVICE,@CHANNEL,@STAT);

IF STAT <> 0 THEN
        SK(604) = (SK(604) OR 01H);

IF STAT = 0 THEN DO;

/*-------------------------------------------------
        OPEN FILE
-------------------------------------------------*/

CALL SOPEN(CHANNEL(0),@STAT,RE_WR,@WHEEL_FILE(0),RAN,0);

IF STAT = -24 THEN
        SK(604) = (SK(604) OR 02H);

IF STAT <> 0 THEN
        SK(604) = (SK(604) OR 04H);

IF STAT = 0 THEN DO;

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

REC_COUNT = 0;

DO WHILE REC_COUNT < 3;

        CALL
SGET(CHANNEL(0),@WHEEL,WHEEL_LEN,@STAT,@IOSTAT,0FFH,REC_COUNT+
1,0);

        CALL MOVB(@SYVAR(8*REC_COUNT),@WHEEL.NAME,8);
        CALL
MOVB(@E_PAR(50+REC_COUNT).VAL,@WHEEL.MAX_SPEED,8);
        CALL
MOVB(@E_PAR(53+REC_COUNT).VAL,@WHEEL.GRIT_SIZE,8);
        CALL
MOVB(@E_PAR(56+REC_COUNT).VAL,@WHEEL.INITIAL_DIAM,8);
        CALL
MOVB(@E_PAR(59+REC_COUNT).VAL,@WHEEL.CURRENT_DIAM,8);
        CALL
MOVB(@E_PAR(62+REC_COUNT).VAL,@WHEEL.FINAL_DIAM,8);
        CALL MOVB(@E_PAR(65+REC_COUNT).VAL,@WHEEL.WIDTH,8);
        CALL MOVB(@E_PAR(68+REC_COUNT).VAL,@WHEEL.DIFF,8);
        CALL MOVB(@E_PAR(71+REC_COUNT).VAL,@WHEEL.COND,8);

        CALL SUPDATE(CHANNEL(0),@WHEEL,WHEEL_LEN,@STAT);

        IF STAT <> 0 THEN DO;
                SK(604) = (SK(604) OR 10H);
                REC_COUNT = 10;
                END;

        IF STAT = 0 THEN
                REC_COUNT = REC_COUNT + 1;

        END;

/*-------------------------------------------------
        CLOSE FILE
-------------------------------------------------*/

END;
        CALL SCLOSE(CHANNEL(0),@STAT);

END;
        CALL SDCONNECT (CHANNEL(0),@STAT);

IF SK(604) = 0 AND SK(605) = 0 THEN SK(606) = 255;

END SAVE_WHEEL_FILE;

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

READ_CYCLE_FILE:
```

```
          PROCEDURE PUBLIC;


SK(604) = (SK(604) AND 0H);
SK(605) = (SK(605) AND 0H);
SK(606) = (SK(606) AND 0H);

/*-------------------------------------------------
          CONNECT CHANNEL
-------------------------------------------------*/
CALL SCONNECT(@DEVICE,@CHANNEL,@STAT);

IF STAT <> 0 THEN
          SK(604) = (SK(604) OR 01H);

IF STAT = 0 THEN

/*-------------------------------------------------
          OPEN FILE
-------------------------------------------------*/
DO;
          CALL
SOPEN(CHANNEL(0),@STAT,READ,@CYCLE_FILE(0),RAN,0);

          IF STAT = -24 THEN
                    SK(604) = (SK(604) OR 02H);

          IF STAT <> 0 THEN
                    SK(604) = (SK(604) OR 04H);

IF STAT = 0 THEN
/*-------------------------------------------------
          GET CYCLE FOR CURRENT COMPONENT
-------------------------------------------------*/
DO;

INDEX = UNSIGN (MC_DATA1.CMPT_NO);
CALL
SGET(CHANNEL(0),@CYCLE,CYCLE_LEN,@STAT,@IOSTAT,0FFH,INDEX,0);

IF STAT <> 0 THEN
          SK(604) = (SK(604) OR 08H);

IF STAT = 0 THEN DO;
          REC_COUNT = 0;

          DO WHILE REC_COUNT < 4;
                    CALL
MOVB(@CYCLE(REC_COUNT).ALLOW,@E_PAR(130+2*REC_COUNT).VAL,0);
                    CALL
MOVB(@CYCLE(REC_COUNT).TOR,@E_PAR(131+2*REC_COUNT).VAL,0);
                    REC_COUNT = REC_COUNT + 1;
                    END;

          END;

/*-------------------------------------------------
          CLOSE FILE
-------------------------------------------------*/

END;

          CALL SCLOSE(CHANNEL(0),@STAT);
          CALL SDCONNECT (CHANNEL(0),@STAT);
END;

IF SK(604) = 0 AND SK(605) = 0 THEN SK(606) = 255;

END READ_CYCLE_FILE;

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

          PROCEDURE TO WRITE ALL THE APPROPRIATE
          E'S AND SYVAR'S TO THE DATA BASE

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

SAVE_CYCLE_FILE:
          PROCEDURE PUBLIC;


SK(604) = (SK(604) AND 00H);
SK(605) = (SK(605) AND 00H);
SK(606) = (SK(606) AND 00H);

/*-------------------------------------------------
          CONNECT CHANNEL
-------------------------------------------------*/

CALL SCONNECT(@DEVICE,@CHANNEL,@STAT);

IF STAT <> 0 THEN
          SK(604) = (SK(604) OR 01H);

IF STAT = 0 THEN DO;
/*-------------------------------------------------
          OPEN FILE
-------------------------------------------------*/

CALL SOPEN(CHANNEL(0),@STAT,RE_WR,@CYCLE_FILE(0),RAN,0);

IF STAT = -24 THEN
          SK(604) = (SK(604) OR 02H);

IF STAT <> 0 THEN
          SK(604) = (SK(604) OR 04H);

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

IF STAT = 0 THEN DO;

INDEX = UNSIGN (MC_DATA1.CMPT_NO);
CALL
SGET(CHANNEL(0),@CYCLE,CYCLE_LEN,@STAT,@IOSTAT,0FFH,INDEX,0);

IF STAT <> 0 THEN
          SK(604) = (SK(604) OR 08H);

IF STAT = 0 THEN DO;

          REC_COUNT = 0;

          DO WHILE REC_COUNT < 4;
                    CALL
MOVB(@E_PAR(130+2*REC_COUNT).VAL,@CYCLE(REC_COUNT).ALLOW,0);
                    CALL
MOVB(@E_PAR(131+2*REC_COUNT).VAL,@CYCLE(REC_COUNT).TOR,0);
                    REC_COUNT = REC_COUNT + 1;
                    END;

          END;

          CALL SUPDATE(CHANNEL(0),@CYCLE,CYCLE_LEN,@STAT);

          IF STAT <> 0 THEN
                    SK(604) = (SK(604) OR 10H);

/*-------------------------------------------------
          CLOSE FILE
-------------------------------------------------*/

END;
          CALL SCLOSE(CHANNEL(0),@STAT);

END;
          CALL SDCONNECT (CHANNEL(0),@STAT);

IF SK(604) = 0 AND SK(605) = 0 THEN SK(606) = 255;

END SAVE_CYCLE_FILE;

/*-------------------------------------------------*/
READ_MC_DATA1_FILE:
          PROCEDURE PUBLIC;

SK(604) = (SK(604) AND 0H);
SK(605) = (SK(605) AND 0H);
SK(606) = (SK(606) AND 0H);

CALL SCONNECT(@DEVICE,@CHANNEL,@STAT);

IF STAT <> 0 THEN
          SK(604) = (SK(604) OR 01H);

IF STAT = 0 THEN
DO;
          CALL
SOPEN(CHANNEL(0),@STAT,READ,@MC_DATA1_FILE(0),RAN,0);

          IF STAT = -24 THEN
                    SK(604) = (SK(604) OR 02H);

          IF STAT <> 0 THEN
                    SK(604) = (SK(604) OR 04H);

IF STAT = 00H THEN
DO;
          CALL
SGET(CHANNEL(0),@MC_DATA1,MC_DATA1_LEN,@STAT,@IOSTAT,0FFH,1,0)
;

          IF STAT <> 0 THEN
                    SK(604) = (SK(604) OR 08H);

          CALL SCLOSE(CHANNEL(0),@STAT);
END;

IF STAT = 0 THEN
DO;
          IF (SK(613) AND 02H) = 02H THEN DO;
                    CALL MOVB(@MC_DATA1.W1,@E_PAR(30).VAL,0);
                    CALL MOVB(@MC_DATA1.W2,@E_PAR(31).VAL,0);
                    CALL MOVB(@MC_DATA1.W3,@E_PAR(32).VAL,0);
                    END;
```

```
        IF (SK(611) AND 02H) = 02H THEN DO;
                CALL MOVB(@MC_DATA1.N1,@E_PAR(30).VAL,8);
                CALL MOVB(@MC_DATA1.N2,@E_PAR(31).VAL,8);
                CALL MOVB(@MC_DATA1.N3,@E_PAR(32).VAL,8);
                CALL
MOVB(@MC_DATA1.BED_LEN,@E_PAR(33).VAL,8);
                CALL
MOVB(@MC_DATA1.SAFETY_DIST,@E_PAR(34).VAL,8);
                CALL
MOVB(@MC_DATA1.CYCLE_DIST,@E_PAR(35).VAL,8);
                CALL
MOVB(@MC_DATA1.CW_DIAM,@E_PAR(36).VAL,8);
                CALL
MOVB(@MC_DATA1.DWELL,@E_PAR(37).VAL,8);
                CALL
MOVB(@MC_DATA1.OFFSET,@E_PAR(38).VAL,8);
                CALL
MOVB(@MC_DATA1.ALLOW,@E_PAR(39).VAL,8);
                CALL
MOVB(@MC_DATA1.CMPT_NO,@E_PAR(10).VAL,2);
                CALL
MOVB(@MC_DATA1.WHEEL_NO,@E_PAR(11).VAL,2);
                END;
END;
END;
        CALL SDCONNECT (CHANNEL(0),@STAT);

IF SK(604) = 0 AND SK(605) = 0 THEN SK(606) = 255;

END READ_MC_DATA1_FILE;

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

SAVE_MC_DATA1_FILE:
        PROCEDURE PUBLIC;

SK(604) = (SK(604) AND 0H);
SK(605) = (SK(605) AND 0H);
SK(606) = (SK(606) AND 0H);

CALL SCONNECT(@DEVICE,@CHANNEL,@STAT);

IF STAT <> 0 THEN
        SK(604) = (SK(604) OR 01H);

IF STAT = 0 THEN DO;

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

CALL SOPEN(CHANNEL(0),@STAT,RE_WR,@MC_DATA1_FILE(0),RAN,0);

IF STAT = -24 THEN
        SK(604) = (SK(604) OR 02H);

IF STAT <> 0 THEN
        SK(604) = (SK(604) OR 04H);

IF STAT = 0 THEN DO;
        CALL
SGET(CHANNEL(0),@MC_DATA1,MC_DATA1_LEN,@STAT,@IOSTAT,0FFH,1,0)
;
        IF (SK(613) AND 04H) = 04H THEN DO;
                CALL MOVB(@E_PAR(30).VAL,@MC_DATA1.N1,8);
                CALL MOVB(@E_PAR(31).VAL,@MC_DATA1.N2,8);
                CALL MOVB(@E_PAR(32).VAL,@MC_DATA1.N3,8);
                END;
        IF (SK(611) AND 04H) = 04H THEN DO;
                CALL
MOVB(@E_PAR(10).VAL,@MC_DATA1.CMPT_NO,2);
                CALL
MOVB(@E_PAR(11).VAL,@MC_DATA1.WHEEL_NO,2);
                CALL MOVB(@E_PAR(30).VAL,@MC_DATA1.N1,8);
                CALL MOVB(@E_PAR(31).VAL,@MC_DATA1.N2,8);
                CALL MOVB(@E_PAR(32).VAL,@MC_DATA1.N3,8);
                CALL
MOVB(@E_PAR(33).VAL,@MC_DATA1.BED_LEN,8);
                CALL
MOVB(@E_PAR(34).VAL,@MC_DATA1.SAFETY_DIST,8);
                CALL
MOVB(@E_PAR(35).VAL,@MC_DATA1.CYCLE_DIST,8);
                CALL
MOVB(@E_PAR(36).VAL,@MC_DATA1.CW_DIAM,8);
                CALL
MOVB(@E_PAR(37).VAL,@MC_DATA1.DWELL,8);
                CALL
MOVB(@E_PAR(38).VAL,@MC_DATA1.OFFSET,8);
                CALL
MOVB(@E_PAR(39).VAL,@MC_DATA1.ALLOW,8);
                END;

        CALL
SUPDATE(CHANNEL(0),@MC_DATA1,MC_DATA1_LEN,@STAT);

        IF STAT <> 0 THEN
                SK(604) = (SK(604) OR 10H);

        END;
        CALL SCLOSE(CHANNEL(0),@STAT);

END;
        CALL SDCONNECT (CHANNEL(0),@STAT);

IF SK(604) = 0 AND SK(605) = 0 THEN SK(606) = 255;

END SAVE_MC_DATA1_FILE;

/*-------------------------------------------------*/
READ_MC_DATA2_FILE:
        PROCEDURE PUBLIC;

SK(604) = (SK(604) AND 0H);
SK(605) = (SK(605) AND 0H);
SK(606) = (SK(606) AND 0H);

CALL SCONNECT(@DEVICE,@CHANNEL,@STAT);

IF STAT <> 0 THEN
        SK(604) = (SK(604) OR 01H);

IF STAT = 0 THEN

DO;
        CALL
SOPEN(CHANNEL(0),@STAT,READ,@MC_DATA2_FILE(0),RAN,0);

        IF STAT = -24 THEN
                SK(604) = (SK(604) OR 02H);

        IF STAT <> 0 THEN
                SK(604) = (SK(604) OR 04H);

IF STAT = 0 THEN
DO;
        CALL
SGET(CHANNEL(0),@MC_DATA2,MC_DATA2_LEN,@STAT,@IOSTAT,0FFH,1,0)
;

IF STAT = 00H THEN
DO;
        CALL
SGET(CHANNEL(0),@MC_DATA2,MC_DATA2_LEN,@STAT,@IOSTAT,0FFH,1,0)
;

        IF STAT <> 0 THEN
                SK(604) = (SK(604) OR 08H);

        CALL SCLOSE(CHANNEL(0),@STAT);
END;

IF STAT = 0 THEN
DO;
        CALL MOVB(@MC_DATA2.VF_RELAX,@E_PAR(30).VAL,8);
        CALL MOVB(@MC_DATA2.VW_RELAX,@E_PAR(31).VAL,8);
        CALL MOVB(@MC_DATA2.MIN_REVS,@E_PAR(32).VAL,8);
        CALL MOVB(@MC_DATA2.Q_MIN,@E_PAR(33).VAL,8);
        CALL MOVB(@MC_DATA2.P_RELAX,@E_PAR(34).VAL,8);
        CALL MOVB(@MC_DATA2.SIZE_RELAX,@E_PAR(35).VAL,8);
        CALL MOVB(@MC_DATA2.TOR_RELAX,@E_PAR(36).VAL,8);
        CALL MOVB(@MC_DATA2.P_LIMIT_K,@E_PAR(37).VAL,8);

END;
END;
        CALL SDCONNECT (CHANNEL(0),@STAT);

IF SK(604) = 0 AND SK(605) = 0 THEN SK(606) = 255;

END READ_MC_DATA2_FILE;

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

SAVE_MC_DATA2_FILE:
        PROCEDURE PUBLIC;

SK(604) = (SK(604) AND 0H);
SK(605) = (SK(605) AND 0H);
SK(606) = (SK(606) AND 0H);

CALL SCONNECT(@DEVICE,@CHANNEL,@STAT);

IF STAT <> 0 THEN
        SK(604) = (SK(604) OR 01H);

IF STAT = 0 THEN DO;

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

CALL SOPEN(CHANNEL(0),@STAT,RE_WR,@MC_DATA2_FILE(0),RAN,0);

IF STAT = -24 THEN
        SK(604) = (SK(604) OR 02H);

IF STAT <> 0 THEN
        SK(604) = (SK(604) OR 04H);

IF STAT = 0 THEN DO;
        CALL
SGET(CHANNEL(0),@MC_DATA2,MC_DATA2_LEN,@STAT,@IOSTAT,0FFH,1,0)
;

/*-------------------------------------------------*/
```

```
            CALL MOVB(@E_PAR(30).VAL,@MC_DATA2.VF_RELAX,8);
            CALL MOVB(@E_PAR(31).VAL,@MC_DATA2.VW_RELAX,8);
            CALL MOVB(@E_PAR(32).VAL,@MC_DATA2.MIN_REVS,8);
            CALL MOVB(@E_PAR(33).VAL,@MC_DATA2.Q_MIN,8);
            CALL MOVB(@E_PAR(34).VAL,@MC_DATA2.P_RELAX,8);
            CALL MOVB(@E_PAR(35).VAL,@MC_DATA2.SIZE_RELAX,8);
            CALL MOVB(@E_PAR(36).VAL,@MC_DATA2.TOR_RELAX,8);
            CALL MOVB(@E_PAR(37).VAL,@MC_DATA2.P_LIMIT_K,8);

            CALL
SUPDATE(CHANNEL(0),@MC_DATA2,MC_DATA2_LEN,@STAT);

            IF STAT <> 0 THEN
                        SK(604) = (SK(604) OR 10H);

            END;

            CALL SCLOSE(CHANNEL(0),@STAT);

END;
            CALL SDCONNECT (CHANNEL(0),@STAT);

IF SK(604) = 0 AND SK(605) = 0 THEN SK(606) = 255;

END SAVE_MC_DATA2_FILE;

/*-------------------------------------------------*/
READ_MC_DATA3_FILE:
            PROCEDURE PUBLIC;

SK(604) = (SK(604) AND 0H);
SK(605) = (SK(605) AND 0H);
SK(606) = (SK(606) AND 0H);

CALL SCONNECT(@DEVICE,@CHANNEL,@STAT);

IF STAT <> 0 THEN
            SK(604) = (SK(604) OR 01H);

IF STAT = 0 THEN

DO;
            CALL
SOPEN(CHANNEL(0),@STAT,READ,@MC_DATA3_FILE(0),RAN,0);

            IF STAT = -24 THEN
                        SK(604) = (SK(604) OR 02H);

            IF STAT <> 0 THEN
                        SK(604) = (SK(604) OR 04H);

IF STAT = 00H THEN
DO;
            CALL
SGET(CHANNEL(0),@MC_DATA3,MC_DATA3_LEN,@STAT,@IOSTAT,0FFH,1,0)
;

            IF STAT <> 0 THEN
                        SK(604) = (SK(604) OR 08H);

            CALL SCLOSE(CHANNEL(0),@STAT);
END;

IF STAT = 0 THEN
DO;
            CALL MOVB(@MC_DATA3.MAN_OR_SWITCH,@SYVAR(0),6);
            CALL MOVB(@MC_DATA3.ADAPT_OR_SWITCH,@SYVAR(6),6);
            CALL MOVB(@MC_DATA3.OS_OR_SWITCH,@SYVAR(12),6);
            CALL MOVB(@MC_DATA3.PECK_OR_SWITCH,@SYVAR(18),6);

            CALL MOVB(@MC_DATA3.OR_VS,@E_PAR(30).VAL,8);
            CALL MOVB(@MC_DATA3.OR_VW,@E_PAR(31).VAL,8);
            CALL MOVB(@MC_DATA3.OR_VF,@E_PAR(32).VAL,8);
            CALL MOVB(@MC_DATA3.OR_OS,@E_PAR(33).VAL,8);
END;

END;
            CALL SDCONNECT (CHANNEL(0),@STAT);

IF SK(604) = 0 AND SK(605) = 0 THEN SK(606) = 255;

END READ_MC_DATA3_FILE;

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

SAVE_MC_DATA3_FILE:
            PROCEDURE PUBLIC;

SK(604) = (SK(604) AND 0H);
SK(605) = (SK(605) AND 0H);
SK(606) = (SK(606) AND 0H);

CALL SCONNECT(@DEVICE,@CHANNEL,@STAT);

IF STAT <> 0 THEN
            SK(604) = (SK(604) OR 01H);

IF STAT = 0 THEN DO;

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

CALL SOPEN(CHANNEL(0),@STAT,RE_WR,@MC_DATA3_FILE(0),RAN,0);

IF STAT = -24 THEN
            SK(604) = (SK(604) OR 02H);

IF STAT <> 0 THEN
            SK(604) = (SK(604) OR 04H);

IF STAT = 0 THEN DO;
            CALL
SGET(CHANNEL(0),@MC_DATA3,MC_DATA3_LEN,@STAT,@IOSTAT,0FFH,1,0)
;

            CALL MOVB(@SYVAR(0),@MC_DATA3.MAN_OR_SWITCH,6);
            CALL MOVB(@SYVAR(6),@MC_DATA3.ADAPT_OR_SWITCH,6);
            CALL MOVB(@SYVAR(12),@MC_DATA3.OS_OR_SWITCH,6);
            CALL MOVB(@SYVAR(18),@MC_DATA3.PECK_OR_SWITCH,6);

            CALL MOVB(@E_PAR(30).VAL,@MC_DATA3.OR_VS,8);
            CALL MOVB(@E_PAR(31).VAL,@MC_DATA3.OR_VW,8);
            CALL MOVB(@E_PAR(32).VAL,@MC_DATA3.OR_VF,8);
            CALL MOVB(@E_PAR(33).VAL,@MC_DATA3.OR_OS,8);

            CALL
SUPDATE(CHANNEL(0),@MC_DATA3,MC_DATA3_LEN,@STAT);

            IF STAT <> 0 THEN
                        SK(604) = (SK(604) OR 10H);

            END;

            CALL SCLOSE(CHANNEL(0),@STAT);
END;

            CALL SDCONNECT (CHANNEL(0),@STAT);

IF SK(604) = 0 AND SK(605) = 0 THEN SK(606) = 255;

END SAVE_MC_DATA3_FILE;

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

FORMAT_TABLE:
            PROCEDURE
(INIT_STRUC_PTR,STRUC_LEN,FILE_NAME_PTR,FOR_STRUC_PTR,
                        FOR_LEN,FOR_REC) PUBLIC;


            DECLARE INIT_STRUC_PTR POINTER;
            DECLARE FILE_NAME_PTR POINTER;
            DECLARE FOR_STRUC_PTR POINTER;
            DECLARE STRUC_LEN WORD;
            DECLARE FOR_LEN WORD;
            DECLARE FOR_REC WORD;
/*-------------------------------------------------
            CONNECT CHANNEL
-------------------------------------------------*/

            SK(634) = 0;
DO;
            CALL
SFORMAT(CHANNEL(0),FILE_NAME_PTR,FOR_STRUC_PTR,FOR_LEN,FOR_REC
,@STAT);
            IF STAT = -16 THEN
                        SK(634) = (SK(634) OR 02H);     /*EXISTS*/
            IF STAT = -15 THEN
                        SK(634) = (SK(634) OR 04H);     /*NO
SECTORS*/
            IF STAT <> 0 THEN
                        SK(634) = (SK(634) OR 01H);      /*ERROR*/

IF SK(634) = 0 THEN

/*-------------------------------------------------
            OPEN FILE
-------------------------------------------------*/
DO;
            CALL
SOPEN(CHANNEL(0),@STAT,WRITE,FILE_NAME_PTR,RAN,0);

            IF STAT = -24 THEN
                        SK(634) = (SK(634) OR 08H);

            IF STAT <> 0 THEN
                        SK(634) = (SK(634) OR 10H);

            IF STAT = 0 THEN

REC_COUNT = 0;

DO WHILE REC_COUNT < FOR_REC;
```

```
CALL
SPUT(CHANNEL(0),INIT_STRUC_PTR,STRUC_LEN,@STAT,@IOSTAT,0,REC_C
OUNT+1,0);

IF STAT <> 0 THEN DO;
          SK(634) = (SK(634) OR 20H);
          REC_COUNT = FOR_REC + 1;
          END;

REC_COUNT = REC_COUNT + 1;

END;

/*-------------------------------------------------
          CLOSE FILE
-------------------------------------------------*/

END;
          CALL SCLOSE(CHANNEL(0),@STAT);
END;

END FORMAT_TABLE;

/*-------------------------------------------------
          PROCEDURE TO WRITE FILTERED TOR
          TO CYCLE DATA BASE -DEPENDS ON PECK NO.

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

LEARN_TOR:
          PROCEDURE PUBLIC;
DECLARE TEMP_RE REAL;

DO;

CALL SCONNECT(@DEVICE,@CHANNEL,@STAT);

IF STAT <> 0 THEN
          SK(631) = (SK(631) OR 01H);

IF STAT = 0 THEN
DO;

/*-------------------------------------------------
          OPEN  MC DATA FILE NO.1
-------------------------------------------------*/

CALL SOPEN(CHANNEL(0),@STAT,READ,@MC_DATA1_FILE(0),RAN,0);

IF STAT <> 0 AND STAT <> -24 THEN
          SK(631) = (SK(631) OR 02H);

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

IF STAT = 0 THEN DO;
          CALL
SGET(CHANNEL(0),@MC_DATA1,MC_DATA1_LEN,@STAT,@IOSTAT,0FFH,1,0)
;
          IF STAT <> 0 THEN
                    SK(631) = (SK(631) OR 08H);
          END;

CALL SCLOSE(CHANNEL(0),@STAT);

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

          OPEN CYCLE FILE FOR READ/WRITE
-------------------------------------------------*/

CALL SOPEN(CHANNEL(0),@STAT,RE_WR,@CYCLE_FILE(0),RAN,0);

IF STAT <> 0 AND STAT <> -24 THEN
          SK(632) = (SK(632) OR 01H);


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

IF STAT = 0 THEN DO;
          INDEX = UNSIGN (MC_DATA1.CMPT_NO);
          CALL
SGET(CHANNEL(0),@CYCLE,CYCLE_LEN,@STAT,@IOSTAT,0FFH,INDEX,0);
          IF STAT <> 0 THEN
                    SK(632) = (SK(632) OR 04H);
          END;

/*-------------------------------------------------
          CALC NEW TOR
-------------------------------------------------*/

IF STAT = 0 THEN DO;

CALL LR2R(@E_PAR(119).VAL,@TEMP_RE);
INDEX = UNSIGN(FIX(TEMP_RE))-1;

IF INDEX >= 0 AND INDEX <= 3 THEN DO;
          CALL LR2R(@TOR,@TEMP_RE);

          IF TEMP_RE >= 0.0 THEN DO;
                    CALL
FILTER(@TOR,@CYCLE(INDEX).TOR,@MC_DATA2.TOR_RELAX);
                    CALL SIG_FIG4(@CYCLE(INDEX).TOR);
                    CALL
SUPDATE(CHANNEL(0),@CYCLE,CYCLE_LEN,@STAT);
                    END;

          IF STAT <> 0 THEN
                    SK(632) = (SK(632) OR 02H);

                    CALL
MOVB(@MC_DATA2.TOR_RELAX,@E_PAR(110).VAL,8);
                    CALL
MOVB(@CYCLE(INDEX).TOR,@E_PAR(111).VAL,8);
                    END;
                    END;

          CALL SCLOSE(CHANNEL(0),@STAT);

          END;

CALL SDCONNECT (CHANNEL(0),@STAT);

END;

END LEARN_TOR;

END DISP1_PROG;
```

# DISP2.PLM

## Real time screen display for grinding cycle.

---

```
DISP2_PROG: DO;
/*-------------------------------------------------

        A slow CSI proggy to generate a screen
        display for the grind

        Started 27/3/1987    (c) Sean Kelly

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

$INCLUDE(CSI.PLM)
$INCLUDE(MATHS.PLM)
$INCLUDE(TABLES.PLM)
$INCLUDE(FILES.PLM)
/*-------------------------------------------------*/

DECLARE CHANNEL (300) WORD;
DECLARE STAT INTEGER;
DECLARE IOSTAT POINTER;

DECLARE PROC_P_MAX REAL EXTERNAL;
DECLARE US_MAX REAL EXTERNAL;
DECLARE P_NO_LOAD REAL EXTERNAL;
DECLARE EC (8) BYTE EXTERNAL;
DECLARE TOR (8) BYTE EXTERNAL;

DECLARE SIZE1 (8) BYTE EXTERNAL;
DECLARE SIZE2 (8) BYTE EXTERNAL;

DECLARE READ LITERALLY '1';
DECLARE RAM LITERALLY '0FFH';
DECLARE DEVICE (4) BYTE DATA ('/MP0');
DECLARE GRIND_MESS_FILE (6) BYTE DATA ('GDMESS');

DECLARE DISP2_DATA_LENGTH WORD EXTERNAL;


DECLARE ASC_BUFFER (20) BYTE;
DECLARE BLANKS (20) BYTE DATA ('                    ');
DECLARE MSG_BLANKS (23) BYTE DATA ('                       ');
DECLARE STAT4 (2) WORD;

DECLARE (VF_SPEED,VS_SPEED,VW_SPEED) REAL EXTERNAL;
DECLARE (VW_RPM,VS_RPM) (8) BYTE EXTERNAL;

DECLARE GRIND_POWER REAL EXTERNAL;
DECLARE P_MAX REAL EXTERNAL;

DECLARE CNT BYTE;

DECLARE STRING_RES WORD;

DECLARE    PART STRUCTURE (
           NAME (12) BYTE,
           FINAL_DIM (8) BYTE,
           START_DIM (8) BYTE,
           INTRN_DIM (8) BYTE,
           CMPT_LEN (8) BYTE,
           TOR (8) BYTE,
           MATL_NO INTEGER) EXTERNAL;

DECLARE MATL STRUCTURE (
           NAME (6) BYTE,
           T_CRIT (8) BYTE,
           DIFF (8) BYTE,
           T_COND (8) BYTE,
           VS (8) BYTE,
           AR (8) BYTE,
           VM (8) BYTE) EXTERNAL;

DECLARE WHEEL STRUCTURE (
           NAME (8) BYTE,
           MAX_SPEED (8) BYTE,
           GRIT_SIZE (8) BYTE,
           INITIAL_DIAM (8) BYTE,
           CURRENT_DIAM (8) BYTE,
           FINAL_DIAM (8) BYTE,
           WIDTH (8) BYTE,
           DIFF (8) BYTE,
           COND (8) BYTE) EXTERNAL;

DECLARE AXIS STRUCTURE (
           VS_CALC (8) BYTE,
           VS_CURR (8) BYTE,

           VW_CALC (8) BYTE,
           VW_CURR (8) BYTE,
           VW_MAX (8) BYTE,
           VW_MIN (8) BYTE,

           VF_REAL_CALC (8) BYTE,
           VF_REAL_CURR (8) BYTE,
           VF_REAL_NEW (8) BYTE,

           VF_FEED_CALC (8) BYTE,
           VF_FEED_CURR (8) BYTE,
           VF_FEED_NEW (8) BYTE) EXTERNAL;

DECLARE MC_DATA1 STRUCTURE (
           CMPT_NO INTEGER,
           WHEEL_NO INTEGER,
           N1 (8) BYTE,
           N2 (8) BYTE,
           N3 (8) BYTE,
           BED_LEN (8) BYTE,
           SAFETY_DIST (8) BYTE,
           CYCLE_DIST (8) BYTE,
           CW_DIAM (8) BYTE,
           DWELL (8) BYTE,
           HOME (8) BYTE,
           SAFETY (8) BYTE,
           STRT (8) BYTE,
           TARGET (8) BYTE,
           OFFSET (8) BYTE,
           ALLOW (8) BYTE) EXTERNAL;

DECLARE MC_DATA3 STRUCTURE (
           MAN_OR_SWITCH (6) BYTE,
           ADAPT_OR_SWITCH (6) BYTE,
           OS_OR_SWITCH (6) BYTE,
           PECK_OR_SWITCH (6) BYTE,
           OR_VS (8) BYTE,
           OR_VW (8) BYTE,
           OR_VF (8) BYTE,
           OR_OS (8) BYTE) EXTERNAL;

DECLARE GRIND_DISP_INIT STRUCTURE(

           CONTROL_WORD WORD,
           REAL_TIME_ROUT_PTR POINTER,
           GLOBAL_ATTRIBUTES BYTE,
           NUMBER_OF_ITEMS WORD,

           VS_ACT     (13) BYTE,
           VS_PROG (13) BYTE,
           VW_ACT  (13) BYTE,
           VW_PROG (13) BYTE,
           VF_ACT  (15) BYTE,
           VF_PROG (15) BYTE,
           FP_ACT (15) BYTE,
           FP_PROG (15) BYTE,
           P_ACT (11) BYTE,
           P_PROG (11) BYTE,
           TOT_GRIN (13) BYTE,
           TIL_DRES (13) BYTE,
           LAB1 (11) BYTE,
           V1 (6) BYTE,
           LAB2 (26) BYTE,
           LAB3 (20) BYTE,
           LAB4 (20) BYTE,
           LAB5 (20) BYTE,
           LAB6 (20) BYTE,
           LAB7 (20) BYTE,
           LAB8 (10) BYTE,
           LAB9 (18) BYTE,
           LAB10 (18) BYTE,
           LAB12 (16) BYTE,
           LAB13 (6) BYTE,
           LAB14 (50) BYTE,
           LAB15 (37) BYTE,
           LAB16 (28) BYTE,
           PT_NAME (11) BYTE,
           PT_FINAL (13) BYTE,
           PT_START (13) BYTE,
           PT_INTRN (13) BYTE,
           PT_LEN (13) BYTE,
           PT_MATL (11) BYTE,
           DIV1 (69) BYTE,
           DIV2 (46) BYTE,
           VTOT (72) BYTE,
           MESSAG1 (28) BYTE,
           MESSAG2 (28) BYTE,
           MESSAG3 (28) BYTE,
           MESSAG4 (28) BYTE,
           MESSAG5 (28) BYTE,
           MESSAG6 (28) BYTE,
           MESSAG7 (28) BYTE,
           DIV3 (69) BYTE,
           LAB17 (11) BYTE,
           LAB18 (11) BYTE,
           SIZE1 (13) BYTE,
```

```
SIZE2 (13) BYTE,
VTOT2 (12) BYTE,
ADAPT (10) BYTE,
OSHOT (10) BYTE,
DBASE (10) BYTE,
PECKR (10) BYTE,
LIMIT (10) BYTE)

DATA (1,0,0FFH,67,
5,18,38H,8,0,'1111.111',
5,29,38H,8,0,'1111.111',
7,18,38H,8,0,'1111.111',
7,29,38H,8,0,'1111.111',
10,18,38H,10,0,'+1111.1111',
10,29,38H,10,0,'+1111.1111',
12,17,38H,10,0,'+1111.1111',
12,28,38H,10,0,'+1111.1111',
14,18,38H,6,0,'111.11',
14,29,38H,6,0,'111.11',
5,55,38H,8,0,'11111111',
6,55,38H,8,0,'11111111',
3,0,247,6,0,' AXES ',
3,6,38H,1,0,'|',
3,16,38H,21,0,'  actual      program',
5,1,38H,15,0,'GDG WHEEL  rpm ',
7,1,38H,15,0,'CTRL WHEEL rpm ',
10,1,38H,15,0,'FEED RATE mm/s ',
12,1,38H,15,0,'FEED POSN   mm ',
14,1,38H,15,0,'MOTOR POWER kW ',
3,41,247,5,0,' M/C ',
5,41,38H,13,0,' CYCLE NO     ',
6,41,38H,13,0,' DRESS IN     ',
16,0,247,11,0,' COMPONENT ',
16,11,38H,1,0,'|',
16,12,38H,45,0,' FINAL    START    INTRN    LENGTH
MATERIAL',
17,12,38H,32,0,' DIAM     DIAM     DIAM     (mm)',
18,12,38H,23,0,'  (mm)    (mm)     (mm)',
20,1,38H,6,0,'AAAAAA',
20,13,38H,8,0,'11111111',
20,22,38H,8,0,'11111111',
20,31,38H,8,0,'11111111',
20,40,38H,8,0,'11111111',
20,49,38H,6,0,'AAAAAA',
8,0,247,64,0,'
|                                                  ',
15,0,247,41,0,'
|',
3,40,38H,1,0,'|',
3,46,38H,1,0,'|',
4,40,38H,1,0,'|',
5,40,38H,1,0,'|',
6,40,38H,1,0,'|',
7,40,38H,1,0,'|',
10,40,38H,1,0,'|',
11,40,38H,1,0,'|',
12,40,38H,1,0,'|',
13,40,38H,1,0,'|',
14,40,38H,1,0,'|',
9,40,38H,1,0,'|',
9,41,247,23,0,'                    ',
10,41,247,23,0,'                    ',
11,41,247,23,0,'                    ',
12,41,215,23,0,'                    ',
13,41,247,23,0,' EC                 ',
14,41,247,23,0,' Us        T        ',
15,41,247,23,0,' kW        NL       ',
21,0,247,64,0,'
',
22,1,38H,6,0,'SIZE 1',
23,1,38H,6,0,'SIZE 2',
22,13,38H,8,0,'11111111',
23,13,38H,8,0,'000.000 ',
22,40,38H,1,0,'|',
23,40,38H,1,0,'|',
22,42,38H,5,0,'ADAPT',
22,49,38H,5,0,'OSHOT',
23,42,38H,5,0,'DBASE',
23,49,38H,5,0,'PECKR',
22,56,38H,5,0,'LIMIT');


DECLARE GRIND_DISP STRUCTURE(

        CONTROL_WORD WORD,
        REAL_TIME_ROUT_PTR POINTER,
        GLOBAL_ATTRIBUTES BYTE,
        NUMBER_OF_ITEMS WORD,

        VS_ACT    (13) BYTE,
        VS_PROG (13) BYTE,
        VW_ACT  (13) BYTE,
        VW_PROG (13) BYTE,
        VF_ACT  (15) BYTE,
        VF_PROG (15) BYTE,
        FP_ACT (15) BYTE,
        FP_PROG (15) BYTE,
        P_ACT (11) BYTE,
        P_PROG (11) BYTE,
        TOT_GRIN (13) BYTE,
        TIL_DRES (13) BYTE,
        LAB1 (11) BYTE,
        V1 (6) BYTE,
        LAB2 (26) BYTE,
        LAB3 (20) BYTE,
        LAB4 (20) BYTE,
        LAB5 (20) BYTE,
        LAB6 (20) BYTE,
        LAB7 (20) BYTE,
        LAB8 (10) BYTE,
        LAB9 (18) BYTE,
        LAB10 (18) BYTE,
        LAB12 (16) BYTE,
        LAB13 (6) BYTE,
        LAB14 (50) BYTE,
        LAB15 (37) BYTE,
        LAB16 (28) BYTE,
        PT_NAME (11) BYTE,
        PT_FINAL (13) BYTE,
        PT_START (13) BYTE,
        PT_INTRN (13) BYTE,
        PT_LEN (13) BYTE,
        PT_MATL (11) BYTE,
        DIV1 (69) BYTE,
        DIV2 (46) BYTE,
        VTOT (72) BYTE,
        MESSAG1 (28) BYTE,
        MESSAG2 (28) BYTE,
        MESSAG3 (28) BYTE,
        MESSAG4 (28) BYTE,
        MESSAG5 (28) BYTE,
        MESSAG6 (28) BYTE,
        MESSAG7 (28) BYTE,
        DIV3 (69) BYTE,
        LAB17 (11) BYTE,
        LAB18 (11) BYTE,
        SIZE1 (13) BYTE,
        SIZE2 (13) BYTE,
        VTOT2 (12) BYTE,
        ADAPT (10) BYTE,
        OSHOT (10) BYTE,
        DBASE (10) BYTE,
        PECKR (10) BYTE,
        LIMIT (10) BYTE) PUBLIC;

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

DISP2:
        PROCEDURE PUBLIC;

        IF DISP2_DATA_LENGTH = 0 THEN DO;
                DISP2_DATA_LENGTH =
SIZE(GRIND_DISP_INIT);
                CALL
MOVB(@GRIND_DISP_INIT,@GRIND_DISP,DISP2_DATA_LENGTH);

                DISP_NUM = 4;
                GRIND_DISP.CONTROL_WORD = 8001H;
                GRIND_DISP.REAL_TIME_ROUT_PTR =
@REAL_TIME_PROC;
                CALL
DEF_DISP(DISP_NUM,@GRIND_DISP,@STATUS_D);

                END;

                CALL
SET_DISP(DISP_NUM,WAIT_EXE,@STATUS_K);

END DISP2;

REAL_TIME_PROC:
        PROCEDURE PUBLIC;

DECLARE LR_TEMP (8) BYTE;

DO;

/*-------------------------------------------------
Update axis position:
-------------------------------------------------*/

        DISP_NUM = 4;

        CALL MOVB(@PART.NAME,@GRIND_DISP.PT_NAME(5),6);
        CALL
CHANGE_ITEM(DISP_NUM,WAIT_EXE,@GRIND_DISP.PT_NAME,@STATUS_C);

        CALL MOVB(@MATL.NAME,@GRIND_DISP.PT_MATL(5),6);
        CALL
CHANGE_ITEM(DISP_NUM,WAIT_EXE,@GRIND_DISP.PT_MATL,@STATUS_C);

        CALL MOVB(@BLANKS,@ASC_BUFFER,20);
        CALL
MQLRASC(@PART.FINAL_DIM,@ASC_BUFFER,@STAT4,20,3);
        CALL MOVB(@BLANKS,@GRIND_DISP.PT_FINAL(5),8);
        CALL MOVB(@ASC_BUFFER(1),@GRIND_DISP.PT_FINAL(5),8);
```

```
                                                    CALL
CHANGE_ITEM(DISP_NUM,WAIT_EXE,@GRIND_DISP.PT_FINAL,$STATUS_C);
                                                    CALL
CALL MOVB(@BLANKS,@ASC_BUFFER,20);
MOLRASC(@PART.START_DIM,@ASC_BUFFER,$STAT4,20,3);
CALL MOVB(@BLANKS,@GRIND_DISP.PT_START(5),8);
CALL MOVB(@ASC_BUFFER(1),@GRIND_DISP.PT_START(5),8);
                                                    CALL
CHANGE_ITEM(DISP_NUM,WAIT_EXE,@GRIND_DISP.PT_START,$STATUS_C);
                                                    CALL
CALL MOVB(@BLANKS,@ASC_BUFFER,20);
                                                    CALL
CHANGE_ITEM(DISP_NUM,WAIT_EXE,@GRIND_DISP.VF_ACT,$STATUS_C);
                                                    CALL
CALL MOVB(@BLANKS,@GRIND_DISP.VF_ACT(5),10);
CALL MOVB(@ASC_BUFFER(1),@GRIND_DISP.VF_ACT(5),10);
CALL MOREASC(@VF_SPEED,@ASC_BUFFER,$STAT4,20,4);
CALL MOVB(@BLANKS,@GRIND_DISP.VF_ACT(5),10);
CALL MOVB(@ASC_BUFFER(1),@GRIND_DISP.VF_ACT(5),10);
                                                    CALL
CHANGE_ITEM(DISP_NUM,WAIT_EXE,@GRIND_DISP.VP_ACT,$STATUS_C);
                                                    CALL
CALL MOVB(@BLANKS,@ASC_BUFFER,20);
MOLRASC(@MC_DATA1.N1,@ASC_BUFFER,$STAT4,20,0);
CALL MOVB(@BLANKS,@GRIND_DISP.TOT_GRIN(5),8);
CALL MOVB(@ASC_BUFFER(1),@GRIND_DISP.TOT_GRIN(5),8);
                                                    CALL
CHANGE_ITEM(DISP_NUM,WAIT_EXE,@GRIND_DISP.TOT_GRIN,$STATUS_C);
                                                    CALL
CALL LR_SUB(@MC_DATA1.N3,@MC_DATA1.N2,@LR_TEMP);
                                                    CALL
CALL MOVB(@BLANKS,@ASC_BUFFER,20);
CALL MOLRASC(@LR_TEMP,@ASC_BUFFER,$STAT4,20,0);
CALL MOVB(@BLANKS,@GRIND_DISP.TIL_DRES(5),8);
CALL MOVB(@ASC_BUFFER(1),@GRIND_DISP.TIL_DRES(5),8);
                                                    CALL
CHANGE_ITEM(DISP_NUM,WAIT_EXE,@GRIND_DISP.TIL_DRES,$STATUS_C);
                                                    CALL
CALL MOREASC(@P_MAX,@ASC_BUFFER,$STAT4,20,2);
CALL MOVB(@BLANKS,@GRIND_DISP.P_PROG(5),6);
CALL MOVB(@ASC_BUFFER(1),@GRIND_DISP.P_PROG(5),6);
                                                    CALL
CHANGE_ITEM(DISP_NUM,WAIT_EXE,@GRIND_DISP.P_PROG,$STATUS_C);
                                                    CALL
CALL MOVB(@BLANKS,@ASC_BUFFER,20);
CALL MOLRASC(@VS_RPM,@ASC_BUFFER,$STAT4,20,3);
CALL MOVB(@BLANKS,@GRIND_DISP.VS_PROG(5),8);
CALL MOVB(@ASC_BUFFER(1),@GRIND_DISP.VS_PROG(5),8);
                                                    CALL
CHANGE_ITEM(DISP_NUM,WAIT_EXE,@GRIND_DISP.VS_PROG,$STATUS_C);
                                                    CALL
CALL MOVB(@BLANKS,@ASC_BUFFER,20);
CALL MOLRASC(@VW_RPM,@ASC_BUFFER,$STAT4,20,3);
CALL MOVB(@BLANKS,@GRIND_DISP.VW_PROG(5),8);
CALL MOVB(@ASC_BUFFER(1),@GRIND_DISP.VW_PROG(5),8);
                                                    CALL
CHANGE_ITEM(DISP_NUM,WAIT_EXE,@GRIND_DISP.VW_PROG,$STATUS_C);
                                                    CALL
CALL SOPEN(CHANNEL(0),$STAT,READ,@GRIND_MESS_FILE(0),RAN,0);
IF STAT = 0 THEN DO;

/*-----------------------------------------------------------*/
                    ERROR MESSAGES
-------------------------------------------------------------*/
CALL SCONNECT(@DEVICE,@CHANNEL,$STAT);
IF STAT = 0 THEN
DO;
                                                    CALL
CHANGE_ITEM(DISP_NUM,WAIT_EXE,@GRIND_DISP.SIZE2,$STATUS_C);
                                                    CALL
CALL MOVB(@BLANKS,@ASC_BUFFER,20);
CALL MOLRASC(@SIZE2,@ASC_BUFFER,$STAT4,20,3);
CALL MOVB(@BLANKS,@GRIND_DISP.SIZE2(5),8);
CALL MOVB(@ASC_BUFFER(1),@GRIND_DISP.SIZE2(5),8);
                                                    CALL
CHANGE_ITEM(DISP_NUM,WAIT_EXE,@GRIND_DISP.SIZE1,$STATUS_C);
                                                    CALL
CALL MOVB(@BLANKS,@ASC_BUFFER,20);
MOLRASC(@PART.CMPT_LEN,@ASC_BUFFER,$STAT4,20,3);
                                                    CALL
CALL MOVB(@BLANKS,@GRIND_DISP.PT_LEN(5),8);
CALL MOVB(@ASC_BUFFER(1),@GRIND_DISP.PT_LEN(5),8);
                                                    CALL
CHANGE_ITEM(DISP_NUM,WAIT_EXE,@GRIND_DISP.PT_LEN,$STATUS_C);
                                                    CALL
CALL MOVB(@BLANKS,@ASC_BUFFER,20);
CALL MOREASC(@POS_PATT,@ASC_BUFFER,$STAT4,20,2);
CALL MOVB(@BLANKS,@GRIND_DISP.F_ACT(5),6);
CALL MOVB(@ASC_BUFFER(1),@GRIND_DISP.F_ACT(5),6);
                                                    CALL
CHANGE_ITEM(DISP_NUM,WAIT_EXE,@GRIND_DISP.F_ACT,$STATUS_C);
                                                    CALL
CALL MOVB(@BLANKS,@ASC_BUFFER,20);
```

```
CALL MOREASC(@NO_LOAD,@ASC_BUFFER,$STAT4,20,2);
CALL MOVB(@BLANKS,@GRIND_DISP.MESSAG7(19),09);
CALL MOVB(@ASC_BUFFER(1),@GRIND_DISP.MESSAG7(20),08);
                                                    CALL
CHANGE_ITEM(DISP_NUM,WAIT_EXE,@GRIND_DISP.MESSAG7,$STATUS_C);
                                                    CALL
CALL MOVB(@BLANKS,@ASC_BUFFER,20);
CALL MOREASC(@TOR,@ASC_BUFFER,$STAT4,20,4);
CALL MOVB(@BLANKS,@GRIND_DISP.MESSAG6(19),09);
CALL MOVB(@ASC_BUFFER(1),@GRIND_DISP.MESSAG6(20),08);
                                                    CALL
CHANGE_ITEM(DISP_NUM,WAIT_EXE,@GRIND_DISP.MESSAG6,$STATUS_C);
                                                    CALL
CALL MOVB(@BLANKS,@ASC_BUFFER,20);
CALL MOREASC(@US_MAX,@ASC_BUFFER,$STAT4,20,4);
CALL MOVB(@BLANKS,@GRIND_DISP.MESSAG6(8),09);
CALL MOVB(@ASC_BUFFER(1),@GRIND_DISP.MESSAG6(9),08);
                                                    CALL
CHANGE_ITEM(DISP_NUM,WAIT_EXE,@GRIND_DISP.MESSAG6,$STATUS_C);
                                                    CALL
CALL MOVB(@BLANKS,@ASC_BUFFER,20);
CALL MOLRASC(@SIZE1,@ASC_BUFFER,$STAT4,20,3);
CALL MOVB(@BLANKS,@GRIND_DISP.SIZE1(5),8);
CALL MOVB(@ASC_BUFFER(1),@GRIND_DISP.SIZE1(5),8);
                                                    CALL
CHANGE_ITEM(DISP_NUM,WAIT_EXE,@GRIND_DISP.SIZE1,$STATUS_C);
                                                    CALL
CHANGE(CHANNEL(0),@GRIND_DISP.MESSAG2(5),23,$STAT,@IOSTAT,OFFH,S
K(621)+8,0);
ELSE CALL MOVB(@MSG_BLANKS,@GRIND_DISP.MESSAG2(5),23);
CALL
CHANGE_ITEM(DISP_NUM,WAIT_EXE,@GRIND_DISP.MESSAG2,$STATUS_C);
IF SK(622) > 0 THEN
CALL
SGET(CHANNEL(0),@GRIND_DISP.MESSAG2(5),23,$STAT,@IOSTAT,OFFH,S
K(621)+8,0);
ELSE CALL MOVB(@MSG_BLANKS,@GRIND_DISP.MESSAG2(5),23);
CALL
CHANGE_ITEM(DISP_NUM,WAIT_EXE,@GRIND_DISP.MESSAG1,$STATUS_C);
IF SK(621) > 0 THEN
CALL
SGET(CHANNEL(0),@GRIND_DISP.MESSAG1(5),23,$STAT,@IOSTAT,OFFH,S
K(620),0);
ELSE CALL MOVB(@MSG_BLANKS,@GRIND_DISP.MESSAG1(5),23);
CALL
CHANGE_ITEM(DISP_NUM,WAIT_EXE,@GRIND_DISP.VW_ACT,$STATUS_C);
                                                    CALL
CALL MOVB(@BLANKS,@ASC_BUFFER,20);
CALL MOREASC(@VW_SPEED,@ASC_BUFFER,$STAT4,20,3);
CALL MOVB(@BLANKS,@GRIND_DISP.VW_ACT(5),8);
CALL MOVB(@ASC_BUFFER(1),@GRIND_DISP.VW_ACT(5),8);
                                                    CALL
CHANGE_ITEM(DISP_NUM,WAIT_EXE,@GRIND_DISP.FP_PROG,$STATUS_C);
                                                    CALL
CALL MOVB(@BLANKS,@ASC_BUFFER,20);
CALL MOLRASC(@END_POINT,@ASC_BUFFER,$STAT4,20,4);
CALL MOVB(@BLANKS,@GRIND_DISP.FP_PROG(5),10);
CALL MOVB(@ASC_BUFFER,@GRIND_DISP.FP_PROG(5),10);
                                                    CALL
CHANGE_ITEM(DISP_NUM,WAIT_EXE,@GRIND_DISP.FP_ACT,$STATUS_C);
                                                    CALL
CALL MOVB(@BLANKS,@ASC_BUFFER,20);
CALL MOLRASC(@AX_POS_PATT,@ASC_BUFFER,$STAT4,20,4);
CALL MOVB(@BLANKS,@GRIND_DISP.PT_FINAL(5),8);
CALL MOVB(@ASC_BUFFER(1),@GRIND_DISP.PT_FINAL(5),8);
                                                    CALL
CHANGE_ITEM(DISP_NUM,WAIT_EXE,@GRIND_DISP.PT_FINAL,$STATUS_C);
                                                    CALL
CALL MOVB(@BLANKS,@ASC_BUFFER,20);
CALL MOREASC(@SPEED_PATT,@ASC_BUFFER,$STAT4,20,3);
CALL MOVB(@BLANKS,@GRIND_DISP.PT_INTRN(5),8);
CALL MOVB(@ASC_BUFFER(1),@GRIND_DISP.PT_INTRN(5),8);
                                                    CALL
CHANGE_ITEM(DISP_NUM,WAIT_EXE,@GRIND_DISP.PT_INTRN,$STATUS_C);
                                                    CALL
CALL MOVB(@BLANKS,@ASC_BUFFER,20);
MOLRASC(@PART.INTRN_DIM,@ASC_BUFFER,$STAT4,20,3);
                                                    CALL
CALL MOVB(@BLANKS,@GRIND_DISP.VP_ACT(5),8);
CALL MOVB(@ASC_BUFFER(1),@GRIND_DISP.VP_ACT(5),8);
                                                    CALL
CHANGE_ITEM(DISP_NUM,WAIT_EXE,@GRIND_DISP.VP_PROG,$STATUS_C);
                                                    CALL
CALL MOVB(@BLANKS,@ASC_BUFFER,20);
CALL MOREASC(@VP_SPEED,@ASC_BUFFER,$STAT4,20,4);
CALL MOVB(@BLANKS,@GRIND_DISP.VP_PROG(5),10);
CALL MOVB(@ASC_BUFFER(1),@GRIND_DISP.VP_PROG(5),10);
                                                    CALL
CHANGE_ITEM(DISP_NUM,WAIT_EXE,@GRIND_DISP.VP_FEED_CURR,$STATUS_C);
                                                    CALL
CALL MOVB(@BLANKS,@ASC_BUFFER,20);
MOLRASC(@AXIS_VP_FEED_CURR,@ASC_BUFFER,$STAT4,20,4);
                                                    CALL
CALL MOVB(@BLANKS,@GRIND_DISP.VP_PROG(5),10);
CALL MOVB(@ASC_BUFFER(1),@GRIND_DISP.VP_PROG(5),10);
                                                    CALL
CHANGE_ITEM(DISP_NUM,WAIT_EXE,@GRIND_DISP.VW_PROG,$STATUS_C);
                                                    CALL
CALL MOVB(@BLANKS,@ASC_BUFFER,20);
```

```
CALL
SGET(CHANNEL(0),@GRIND_DISP.MESSAG3(5),23,@STAT,@IOSTAT,0FFH,S
K(622)+16,0);
ELSE CALL MOVB(@MSG_BLANKS,@GRIND_DISP.MESSAG3(5),23);
CALL
CHANGE_ITEM(DISP_NUM,WAIT_EXE,@GRIND_DISP.MESSAG3,@STATUS_C);

IF SK(623) > 0 THEN
CALL
SGET(CHANNEL(0),@GRIND_DISP.MESSAG4(5),23,@STAT,@IOSTAT,0FFH,S
K(623)+24,0);
ELSE CALL MOVB(@MSG_BLANKS,@GRIND_DISP.MESSAG4(5),23);
CALL
CHANGE_ITEM(DISP_NUM,WAIT_EXE,@GRIND_DISP.MESSAG4,@STATUS_C);

/*IF SK(624) > 0 THEN
CALL
SGET(CHANNEL(0),@GRIND_DISP.MESSAG5(5),23,@STAT,@IOSTAT,0FFH,S
K(624)+32,0);
ELSE CALL MOVB(@MSG_BLANKS,@GRIND_DISP.MESSAG5(5),23);
CALL
CHANGE_ITEM(DISP_NUM,WAIT_EXE,@GRIND_DISP.MESSAG5,@STATUS_C);

IF SK(625) > 0 THEN
CALL
SGET(CHANNEL(0),@GRIND_DISP.MESSAG6(5),23,@STAT,@IOSTAT,0FFH,S
K(625)+40,0);
ELSE CALL MOVB(@MSG_BLANKS,@GRIND_DISP.MESSAG6(5),23);
CALL
CHANGE_ITEM(DISP_NUM,WAIT_EXE,@GRIND_DISP.MESSAG6,@STATUS_C);

IF SK(626) > 0 THEN
CALL
SGET(CHANNEL(0),@GRIND_DISP.MESSAG7(5),23,@STAT,@IOSTAT,0FFH,S
K(626)+48,0);
ELSE CALL MOVB(@MSG_BLANKS,@GRIND_DISP.MESSAG7(5),23);
CALL
CHANGE_ITEM(DISP_NUM,WAIT_EXE,@GRIND_DISP.MESSAG7,@STATUS_C);*
/

END;

CALL SCLOSE(CHANNEL(0),@STAT);
END;

CALL SDCONNECT (CHANNEL(0),@STAT);


/*-------------------------------------------------
          OSHOT,ADAPT,PECKR AND DBASE FLAGS
-------------------------------------------------*/


STRING_RES = CMPB(@MC_DATA3.MAN_OR_SWITCH,@('OFF    '),6);

IF STRING_RES = 0FFFFH THEN DO;

CNT = 0;
DO WHILE CNT < 5;
          GRIND_DISP.DBASE(5+CNT) = GRIND_DISP.DBASE(5+CNT) OR
128;
          CNT = CNT + 1;
          END;
CALL
CHANGE_ITEM(DISP_NUM,WAIT_EXE,@GRIND_DISP.DBASE,@STATUS_C);
END;


STRING_RES = CMPB(@MC_DATA3.MAN_OR_SWITCH,@('ON     '),6);

IF STRING_RES = 0FFFFH THEN DO;

CNT = 0;
DO WHILE CNT < 5;
          GRIND_DISP.DBASE(5+CNT) = GRIND_DISP.DBASE(5+CNT)
AND 127;
          CNT = CNT + 1;
          END;
CALL
CHANGE_ITEM(DISP_NUM,WAIT_EXE,@GRIND_DISP.DBASE,@STATUS_C);

END;

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

STRING_RES = CMPB(@MC_DATA3.OS_OR_SWITCH,@('OFF    '),6);

IF STRING_RES = 0FFFFH THEN DO;

CNT = 0;
DO WHILE CNT < 5;
          GRIND_DISP.OSHOT(5+CNT) = GRIND_DISP.OSHOT(5+CNT) OR
128;
          CNT = CNT + 1;
          END;
CALL
CHANGE_ITEM(DISP_NUM,WAIT_EXE,@GRIND_DISP.OSHOT,@STATUS_C);

END;


STRING_RES = CMPB(@MC_DATA3.OS_OR_SWITCH,@('ON     '),6);

IF STRING_RES = 0FFFFH THEN DO;

CNT = 0;
DO WHILE CNT < 5;
          GRIND_DISP.OSHOT(5+CNT) = GRIND_DISP.OSHOT(5+CNT)
AND 127;
          CNT = CNT + 1;
          END;
CALL
CHANGE_ITEM(DISP_NUM,WAIT_EXE,@GRIND_DISP.OSHOT,@STATUS_C);

END;

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

STRING_RES = CMPB(@MC_DATA3.ADAPT_OR_SWITCH,@('OFF    '),6);

IF STRING_RES = 0FFFFH THEN DO;

CNT = 0;
DO WHILE CNT < 5;
          GRIND_DISP.ADAPT(5+CNT) = GRIND_DISP.ADAPT(5+CNT) OR
128;
          CNT = CNT + 1;
          END;
CALL
CHANGE_ITEM(DISP_NUM,WAIT_EXE,@GRIND_DISP.ADAPT,@STATUS_C);

END;


STRING_RES = CMPB(@MC_DATA3.ADAPT_OR_SWITCH,@('ON     '),6);

IF STRING_RES = 0FFFFH THEN DO;

CNT = 0;
DO WHILE CNT < 5;
          GRIND_DISP.ADAPT(5+CNT) = GRIND_DISP.ADAPT(5+CNT)
AND 127;
          CNT = CNT + 1;
          END;
CALL
CHANGE_ITEM(DISP_NUM,WAIT_EXE,@GRIND_DISP.ADAPT,@STATUS_C);

END;

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

STRING_RES = CMPB(@MC_DATA3.PECK_OR_SWITCH,@('OFF    '),6);

IF STRING_RES = 0FFFFH THEN DO;

CNT = 0;
DO WHILE CNT < 5;
          GRIND_DISP.PECKR(5+CNT) = GRIND_DISP.PECKR(5+CNT) OR
128;
          CNT = CNT + 1;
          END;
CALL
CHANGE_ITEM(DISP_NUM,WAIT_EXE,@GRIND_DISP.PECKR,@STATUS_C);
END;


STRING_RES = CMPB(@MC_DATA3.PECK_OR_SWITCH,@('ON     '),6);

IF STRING_RES = 0FFFFH THEN DO;

CNT = 0;
DO WHILE CNT < 5;
          GRIND_DISP.PECKR(5+CNT) = GRIND_DISP.PECKR(5+CNT)
AND 127;
          CNT = CNT + 1;
          END;
CALL
CHANGE_ITEM(DISP_NUM,WAIT_EXE,@GRIND_DISP.PECKR,@STATUS_C);

END;

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

IF GRIND_POWER > P_MAX AND SK(616) = 2 THEN DO;

CNT = 0;
DO WHILE CNT < 5;
          GRIND_DISP.LIMIT(5+CNT) = GRIND_DISP.LIMIT(5+CNT) OR
128;
          GRIND_DISP.LIMIT(2) = GRIND_DISP.LIMIT(2) AND 0DFH;
          CNT = CNT + 1;
          END;

CALL
CHANGE_ITEM(DISP_NUM,WAIT_EXE,@GRIND_DISP.LIMIT,@STATUS_C);

END;
```

```
/*-------------------------------------------------*/

IF GRIND_POWER < P_MAX OR SK(616) = 0 THEN DO;

CNT = 0;
DO WHILE CNT < 5;
          GRIND_DISP.LIMIT(5+CNT) = GRIND_DISP.LIMIT(5+CNT)
AND 0127;
          GRIND_DISP.LIMIT(2) = GRIND_DISP.LIMIT(2) OR 0FFH;
          CNT = CNT + 1;
          END;

CALL
CHANGE_ITEM(DISP_NUM,WAIT_EXE,@GRIND_DISP.LIMIT,@STATUS_C);

END;

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

END;

END REAL_TIME_PROC;

END DISP2_PROG;
```

# DISP3.PLM

## Large character real time screen display for grinding cycle.

---

```
DISP3_PROG: DO;
/*-------------------------------------------------

        A slow CSI proggy to generate a screen
        display for the grind

        Started 27/3/1987     (c) Sean Kelly

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

$INCLUDE(CSI.PLM)
$INCLUDE(MATHS.PLM)
$INCLUDE(TABLES.PLM)
$INCLUDE(FILES.PLM)

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

DECLARE DISP3_DATA_LENGTH WORD EXTERNAL;

DECLARE CHANNEL (300) WORD;
DECLARE STAT INTEGER;
DECLARE IOSTAT POINTER;

DECLARE ASC_BUFFER (20) BYTE;
DECLARE BLANKS (20) BYTE DATA ('                    ');
DECLARE STAT4 (2) WORD;
DECLARE GRIND_POWER REAL EXTERNAL;
DECLARE P_MAX REAL EXTERNAL;
DECLARE (VF_SPEED,VS_SPEED,VW_SPEED) REAL EXTERNAL;
DECLARE SIZE1 (8) BYTE EXTERNAL;
DECLARE SIZE2 (8) BYTE EXTERNAL;

DECLARE READ LITERALLY '1';
DECLARE RAW LITERALLY '0FFH';
DECLARE DEVICE (4) BYTE DATA ('/MP0');
DECLARE GRIND_MESS_FILE (6) BYTE DATA ('GDMESS');
DECLARE GD_MESS_BUFF (23) BYTE;

DECLARE    PART STRUCTURE (
           NAME (12) BYTE,
           FINAL_DIM (8) BYTE,
           START_DIM (8) BYTE,
           INTRM_DIM (8) BYTE,
           CMPT_LEN (8) BYTE,
           TOR (8) BYTE,
           MATL_NO INTEGER) EXTERNAL;

DECLARE MATL STRUCTURE (
           NAME (6) BYTE,
           T_CRIT (8) BYTE,
           DIFF (8) BYTE,
           T_COND (8) BYTE,
           VS (8) BYTE,
           AR (8) BYTE,
           VM (8) BYTE) EXTERNAL;

DECLARE WHEEL STRUCTURE (
           NAME (8) BYTE,
           MAX_SPEED (8) BYTE,
           GRIT_SIZE (8) BYTE,
           INITIAL_DIAM (8) BYTE,
           CURRENT_DIAM (8) BYTE,
           FINAL_DIAM (8) BYTE,
           WIDTH (8) BYTE,
           DIFF (8) BYTE,
           COND (8) BYTE) EXTERNAL;

DECLARE AXIS STRUCTURE (
           VS_CALC (8) BYTE,
           VS_CURR (8) BYTE,

           VW_CALC (8) BYTE,
           VW_CURR (8) BYTE,
           VW_MAX (8) BYTE,
           VW_MIN (8) BYTE,

           VF_REAL_CALC (8) BYTE,
           VF_REAL_CURR (8) BYTE,
           VF_REAL_NEW (8) BYTE,

           VF_FEED_CALC (8) BYTE,
           VF_FEED_CURR (8) BYTE,
           VF_FEED_NEW (8) BYTE) EXTERNAL;

DECLARE MC_DATA1 STRUCTURE (
           CMPT_NO INTEGER,
           WHEEL_NO INTEGER,
           N1 (8) BYTE,
           N2 (8) BYTE,
           N3 (8) BYTE,
           BED_LEN (8) BYTE,
           SAFETY_DIST (8) BYTE,
           CYCLE_DIST (8) BYTE,
           CW_DIAM (8) BYTE,
           DWELL (8) BYTE,
           HOME (8) BYTE,
           SAFETY (8) BYTE,
           STRT (8) BYTE,
           TARGET (8) BYTE,
           OFFSET (8) BYTE,
           ALLOW (8) BYTE) EXTERNAL;

/*-------------------------------------------------*/
           DECLARE INIT_DISP3 STRUCTURE(
                      CONTROL_WORD WORD,
                      REAL_TIME_ROUT_PTR POINTER,
                      GLOBAL_ATTRIBUTES BYTE,
                      NUMBER_OF_ITEMS WORD,

                      LABEL1 (12) BYTE,
                      VS_SPEED (13) BYTE,
                      LABEL2 (12) BYTE,
                      VW_SPEED (13) BYTE,
                      LABEL3 (12) BYTE,
                      FEED_POS (15) BYTE,
                      LABEL4 (12) BYTE,
                      FEED_RATE (14) BYTE,
                      LABEL5 (12) BYTE,
                      SIZE1 (13) BYTE,
                      LABEL6 (12) BYTE,
                      SIZE2 (13) BYTE,
                      LABEL7 (10) BYTE,
                      POWER (10) BYTE,
                      LABEL8 (10) BYTE,
                      P_MAX (10) BYTE,
                      LABEL9 (10) BYTE,
                      CY_NO (9) BYTE,
                      LABEL10 (10) BYTE,
                      DR_IN (9) BYTE,
                      V1 (6) BYTE,
                      V2 (6) BYTE,
                      V3 (37) BYTE,
                      V4 (6) BYTE,
                      V5 (6) BYTE,
                      V6 (37) BYTE,
                      V7 (6) BYTE,
                      V8 (6) BYTE,
                      V9 (6) BYTE,
                      MESSAG1 (18) BYTE,
                      MESSAG2 (18) BYTE,
                      MESSAG3 (18) BYTE)
           DATA (0,0,0FFH,32,
           3,0,38H,7,0,'GW  rpm',
           3,9,38H,8,0,'    0.000',

           4,0,38H,7,0,'CW  rpm',
           4,9,38H,8,0,'    0.000',

           6,0,38H,7,0,'FP    mm',
           6,8,38H,10,0,'    0.0000',

           7,0,38H,7,0,'FR mm/s',
           7,9,38H,9,0,'    0.0000',

           9,0,38H,7,0,'SIZ1 mm',
           9,9,38H,8,0,'    0.000',

           10,0,38H,7,0,'SIZ2 mm',
           10,9,38H,8,0,'    0.000',

           6,20,38H,5,0,'P  kW',
           6,26,38H,5,0,' 0.00',

           7,20,38H,5,0,'P MAX',
           7,26,38H,5,0,' 0.00',

           3,20,38H,5,0,'CY NO',
           3,26,38H,4,0,'   0',

           4,20,38H,5,0,'DR IN',
           4,26,38H,4,0,'   0',

           3,18,38H,1,0,'|',
           4,18,38H,1,0,'|',
           5,0,30H,32,0,'

           6,18,38H,1,0,'|',
           7,18,38H,1,0,'|',
```

```
/*-------------------------------------------------------------*/
DISP3:
        PROCEDURE PUBLIC;

DO:
        IF DISP3_DATA_LENGTH = 0 THEN DO;
                DISP3_DATA_LENGTH = SIZE(INIT_DISP3);
                CALL
MOVB(@INIT_DISP3,@GRIND_DISP3_DATA_LENGTH);

        DISP_NUM = 5;
        GRIND_DISP3.CONTROL_WORD = 8000H;
        GRIND_DISP3.REAL_TIME_ROOT_PTR =
@REAL_TIME_PROC;

DECLARE GRIND_DISP3 STRUCTURE(
        CONTROL_WORD WORD,
        REAL_TIME_ROOT_PTR POINTER,
        GLOBAL_ATTRIBUTES BYTE,
        NUMBER_OF_ITEMS WORD,

        LABEL1 (12) BYTE,
        V5_SPEED (13) BYTE,
        LABEL2 (12) BYTE,
        VW_SPEED (13) BYTE,
        LABEL3 (12) BYTE,
        FEED_POS (15) BYTE,
        LABEL4 (12) BYTE,
        FEED_RATE (14) BYTE,
        LABEL5 (12) BYTE,
        SIZE1 (13) BYTE,
        LABEL6 (12) BYTE,
        SIZE2 (13) BYTE,
        LABEL7 (10) BYTE,
        POWER (10) BYTE,
        LABEL8 (10) BYTE,
        P_MAX (10) BYTE,
        LABEL9 (10) BYTE,
        CY_NO (9) BYTE,
        LABEL10 (10) BYTE,
        DR_IN (9) BYTE,
        V1 (6) BYTE,
        V2 (6) BYTE,
        V3 (37) BYTE,
        V4 (6) BYTE,
        V5 (6) BYTE,
        V6 (37) BYTE,
        V7 (6) BYTE,
        V8 (6) BYTE,
        V9 (6) BYTE,
        MESSAG1 (18) BYTE,
        MESSAG2 (18) BYTE,
        MESSAG3 (18) BYTE) PUBLIC;

/*-------------------------------------------------------------
                        LARGE CHARACTER DISPLAY.

                        V1.0 Sean Kelly 7/10/88
-------------------------------------------------------------*/

END DISP3;

DEF_DISP(DISP_NUM,@GRIND_DISP3,@STATUS_D);
                                        CALL
END;

SET_DISP(DISP_NUM,WAIT_EXE,@STATUS_X);
                                        CALL
END;

REAL_TIME_PROC3:
        PROCEDURE PUBLIC;

DECLARE LR_TEMP (8) BYTE;

DO;

        DISP_NUM = 5;
        CALL_MOVB(@BLANKS,@ASC_BUFFER,20);
        CALL MOLRASC(@AX_POS.PATT,@ASC_BUFFER,@STAT4,20,4);
        CALL MOVB(@BLANKS,@GRIND_DISP3.FEED_POS(5),10);
        CALL MOVB(@ASC_BUFFER,@GRIND_DISP3.FEED_POS(5),10);
CHANGE_ITEM2(DISP_NUM,WAIT_EXE,@GRIND_DISP3.FEED_POS,@STATUS_C
);

        CALL MOVB(@BLANKS,@ASC_BUFFER,20);
        CALL MOLRASC(@SIZE1,@ASC_BUFFER,@STAT4,20,3);
        CALL MOVB(@BLANKS,@GRIND_DISP3.SIZE1(5),8);
        CALL MOVB(@ASC_BUFFER(1),@GRIND_DISP3.SIZE1(5),8);
CHANGE_ITEM2(DISP_NUM,WAIT_EXE,@GRIND_DISP3.SIZE1,@STATUS_C
);

        CALL MOVB(@BLANKS,@ASC_BUFFER,20);
        CALL MOREASC(@VW_SPEED,@ASC_BUFFER,@STAT4,20,3);
        CALL MOVB(@BLANKS,@GRIND_DISP3.VW_SPEED(5),8);
        CALL
MOVB(@ASC_BUFFER(1),@GRIND_DISP3.VW_SPEED(5),8);
CHANGE_ITEM2(DISP_NUM,WAIT_EXE,@GRIND_DISP3.VW_SPEED,@STATUS_C
);

        CALL MOVB(@BLANKS,@ASC_BUFFER,20);
        CALL MOREASC(@V5_SPEED,@ASC_BUFFER,@STAT4,20,3);
        CALL MOVB(@BLANKS,@GRIND_DISP3.V5_SPEED(5),8);
        CALL
MOVB(@ASC_BUFFER(1),@GRIND_DISP3.V5_SPEED(5),8);
CHANGE_ITEM2(DISP_NUM,WAIT_EXE,@GRIND_DISP3.V5_SPEED,@STATUS_C
);

        CALL MOVB(@BLANKS,@ASC_BUFFER,20);
        CALL MOREASC(@P_MAX,@ASC_BUFFER,@STAT4,20,2);
        CALL MOVB(@BLANKS,@GRIND_DISP3.P_MAX(5),5);
        CALL MOVB(@ASC_BUFFER(1),@GRIND_DISP3.P_MAX(5),5);
CHANGE_ITEM2(DISP_NUM,WAIT_EXE,@GRIND_DISP3.P_MAX,@STATUS_C);

        CALL MOVB(@BLANKS,@ASC_BUFFER,20);
        CALL MOLRASC(@MC_DATA1.N1,@ASC_BUFFER,@STAT4,20,0);
        CALL MOVB(@BLANKS,@GRIND_DISP3.CY_NO(5),4);
        CALL MOVB(@ASC_BUFFER(1),@GRIND_DISP3.CY_NO(5),4);
CHANGE_ITEM2(DISP_NUM,WAIT_EXE,@GRIND_DISP3.CY_NO,@STATUS_C);
        CALL LR_SUB(@MC_DATA1.N3,@MC_DATA1.N2,@LR_TEMP);

        CALL MOVB(@BLANKS,@ASC_BUFFER,20);
        CALL MOLRASC(@LR_TEMP,@ASC_BUFFER,@STAT4,20,0);
        CALL MOVB(@BLANKS,@GRIND_DISP3.DR_IN(5),4);
        CALL MOVB(@ASC_BUFFER(1),@GRIND_DISP3.DR_IN(5),4);
CHANGE_ITEM2(DISP_NUM,WAIT_EXE,@GRIND_DISP3.DR_IN,@STATUS_C);

/*-------------------------------------------------------------
                        ERROR MESSAGES
-------------------------------------------------------------*/
CALL CONNECT(@DEVICE,@CHANNEL,@STAT);
IF STAT = 0 THEN
DO;
CALL SOPEN(CHANNEL(0),@STAT,READ,@GRIND_MESS_FILE(0),RAN,0);
IF STAT = 0 THEN DO;

/*-------------------------------------------------------------*/
IF SK(620) > 0 THEN DO;
                CALL
SCET(CHANNEL(0),@GD_MESS_BUFF,23,@STAT,@IOSTAT,OFFH,SK(620),0)
;
        CALL MOVB(@GD_MESS_BUFF,@GRIND_DISP3.MESSAG1(5),13);
        END;
ELSE CALL MOVB(@BLANKS,@GRIND_DISP3.MESSAG1(5),13);

CALL
CHANGE_ITEM2(DISP_NUM,WAIT_EXE,@GRIND_DISP3.MESSAG1,@STATUS_C)
;
```

```
/*-------------------------------------------------*/

IF SK(621) > 0 THEN DO;
          CALL
SGET(CHANNEL(0),@GD_MESS_BUFF,23,@STAT,@IOSTAT,0FFH,SK(621)+8,
0);
          CALL MOVB(@GD_MESS_BUFF,@GRIND_DISP3.MESSAG2(5),13);
          END;

ELSE CALL MOVB(@BLANKS,@GRIND_DISP3.MESSAG2(5),13);

CALL
CHANGE_ITEM2(DISP_NUM,WAIT_EXE,@GRIND_DISP3.MESSAG2,@STATUS_C)
;


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

IF SK(623) > 0 THEN DO;
          CALL
SGET(CHANNEL(0),@GD_MESS_BUFF,23,@STAT,@IOSTAT,0FFH,SK(623)+24
,0);
          CALL MOVB(@GD_MESS_BUFF,@GRIND_DISP3.MESSAG3(5),13);
          END;

ELSE CALL MOVB(@BLANKS,@GRIND_DISP3.MESSAG3(5),13);

CALL
CHANGE_ITEM2(DISP_NUM,WAIT_EXE,@GRIND_DISP3.MESSAG3,@STATUS_C)
;
/*-------------------------------------------------*/

END;

CALL SCLOSE(CHANNEL(0),@STAT);
END;

CALL SDCONNECT (CHANNEL(0),@STAT);


END;

END REAL_TIME_PROC3;

END DISP3_PROG;
```

# CALC.PLM

## CSI module for calculation of process model parameters.

---

```
CALC: DO;

/*-----------------------------------------------------
          A program to prform the necessary calculations
          to determine grinding variables.

          (c) Sean Kelly March 1988


          Uses the contents of structures generated
          by DISP.PLM to calculate:

                    Vf,Vs,Vw,Ecrit,
                    also axis positions.

          Uses long real conversion routines from
          MATHS.ASM

-----------------------------------------------------*/
$INCLUDE(TABLES.PLM)
$INCLUDE(MATHS.PLM)

/*-----------------------------------------------------
          PROCEDURE TO CALCULATE VF

          VF = 2*VM*VS/(PI*DW*LD*LD)

-----------------------------------------------------*/
CALC_VF:
          PROCEDURE(CHIP_VOL_PTR,VS_PTR,DIAM_PTR,GRIT_SIZE_PTR
,VF_PTR)
          EXTERNAL;

          DECLARE
(CHIP_VOL_PTR,VS_PTR,DIAM_PTR,GRIT_SIZE_PTR,VF_PTR) POINTER;

END CALC_VF;

/*-----------------------------------------------------
          PROCEDURE TO CALCULATE REAL VF

-----------------------------------------------------*/
CALC_VF_REAL:
          PROCEDURE(VF_FEED_PTR,T_INFEED_PTR,TOR_PTR,VF_REAL_P
TR)
          EXTERNAL;

          DECLARE
(VF_FEED_PTR,T_INFEED_PTR,TOR_PTR,VF_REAL_PTR) POINTER;

END CALC_VF_REAL;


/*-----------------------------------------------------
          PROCEDURE TO CALCULATE VW

          VW = DE*VS/(LD*AR)

-----------------------------------------------------*/
CALC_VW:
          PROCEDURE(DE_PTR,VS_PTR,GRIT_SIZE_PTR,CHIP_RATIO_PTR
,VW_PTR)
          EXTERNAL;

          DECLARE
(DE_PTR,VS_PTR,GRIT_SIZE_PTR,CHIP_RATIO_PTR,VW_PTR) POINTER;

END CALC_VW;


/*-----------------------------------------------------
          PROCEDURE TO CALCULATE MIN VW

          VW MIN = PI*DW*n (n=no. of cmpt revs/s)

-----------------------------------------------------*/
CALC_VW_MIN:
          PROCEDURE(DIAM_PTR,MIN_REVS_PTR,VW_MIN_PTR)
          EXTERNAL;

          DECLARE (DIAM_PTR,MIN_REVS_PTR,VW_MIN_PTR) POINTER;

END CALC_VW_MIN;


/*-----------------------------------------------------
          CALCULATE EFFECTIVE DIAM

          DE = DS*DW/(DS+DW)

-----------------------------------------------------*/
CALC_DE:
          PROCEDURE(DW_PTR,DS_PTR,DE_PTR)
          EXTERNAL;

          DECLARE (DW_PTR,DS_PTR,DE_PTR) POINTER;

END CALC_DE;


/*-----------------------------------------------------
          PROCEDURE TO CALCULATE CHIP LENGTH
          AND DEPTH OF CUT

          A = 0.5*PI*DW*VF/(VW*1000)
          LG = SQRT(A*DE)

-----------------------------------------------------*/
CALC_LG:
          PROCEDURE(DIAM_PTR,VF_PTR,VW_PTR,DE_PTR,A_PTR,LG_PTR
)
          EXTERNAL;

          DECLARE (DIAM_PTR,VF_PTR,VW_PTR,DE_PTR,A_PTR,LG_PTR)
POINTER;

END CALC_LG;

/*-----------------------------------------------------
          CALCULATE LE

          LE = LG*(4.95*((VS/VW)**-0.216)*EXP(-
0.0205*((VS/VW)**1/3)*LN(A)))

-----------------------------------------------------*/
CALC_LE:
          PROCEDURE(VS_PTR,VW_PTR,CUT_DEPTH_PTR,LG_PTR,LE_PTR)
          EXTERNAL;

          DECLARE (VS_PTR,VW_PTR,CUT_DEPTH_PTR,LG_PTR,LE_PTR)
POINTER;

END CALC_LE;

/*-----------------------------------------------------
          CALCULATE AXIS CYCLE POSITIONS

          POS = -1*(BED_LEN-DIAM-DIST)

-----------------------------------------------------*/
CALC_POS:
          PROCEDURE(BED_LEN_PTR,DIAM_PTR,DIST_PTR,POS_PTR)
          EXTERNAL;

          DECLARE (BED_LEN_PTR,DIAM_PTR,DIST_PTR,POS_PTR)
POINTER;

END CALC_POS;

/*-----------------------------------------------------
          CALCULATE RATIO R

          1/R = 1 +
SQRT((TM_DIFF*VS)/(TS_DIFF*VW))*TS_COND/TM_COND

-----------------------------------------------------*/
CALC_R:
          PROCEDURE(TM_DIF_PTR,TS_DIF_PTR,VW_PTR,VS_PTR,TM_CON
_PTR,
                    TS_CON_PTR,R_PTR)
          EXTERNAL;

          DECLARE
(TM_DIF_PTR,TS_DIF_PTR,VW_PTR,VS_PTR,TM_CON_PTR,
                    TS_CON_PTR,R_PTR) POINTER;

END CALC_R;

/*-----------------------------------------------------
          CALCULATE CRITICAL ENERGY

          EC =
TM_CRIT*SQRT(LE/(VW*TM_DIFF))*TM_COND/(0.667*R*A)

-----------------------------------------------------*/
CALC_EC:
```

PAGE
MISSING
IN
ORIGINAL

```
            IF RESULT <= 0.0 THEN
                        CALL MOVB(@WHEEL.MAX_SPEED,@VS,8);

            CALL MSRPM(@VS,@WHEEL.CURRENT_DIAM,@VS_RPM);
            CALL LR2R(@VS_RPM,@TEMP_RE);

            IF TEMP_RE > 2000.0 THEN DO;
                        TEMP_RE = 2000.0;
                        CALL R2LR(@TEMP_RE,@VS_RPM);
                        SK(635) = SK(635) OR 01H;
                        END;

            ELSE IF TEMP_RE <= 1000.0 THEN DO;
                        TEMP_RE = 1000.0;
                        CALL R2LR(@TEMP_RE,@VS_RPM);
                        SK(635) = SK(635) OR 02H;
                        END;

            CALL MOVB(@VS,@AXIS.VS_CALC,8);
            CALL
RPMMS(@VS_RPM,@WHEEL.CURRENT_DIAM,@AXIS.VS_CURR);
            CALL LR2R(@VS_RPM,@E_PAR(25).VAL);

/*-------------------------------------------------
            CALC VF IN MM/S
-------------------------------------------------*/
CALL
LR_SUB(@AXIS.VF_REAL_CALC,@AXIS.VF_REAL_CALC,@AXIS.VF_REAL_CAL
C);
CALL
LR_SUB(@AXIS.VF_REAL_CURR,@AXIS.VF_REAL_CURR,@AXIS.VF_REAL_CUR
R);
CALL
LR_SUB(@AXIS.VF_REAL_NEW,@AXIS.VF_REAL_NEW,@AXIS.VF_REAL_NEW);

CALL
LR_SUB(@AXIS.VF_FEED_CALC,@AXIS.VF_FEED_CALC,@AXIS.VF_FEED_CAL
C);
CALL
LR_SUB(@AXIS.VF_FEED_CURR,@AXIS.VF_FEED_CURR,@AXIS.VF_FEED_CUR
R);
CALL
LR_SUB(@AXIS.VF_FEED_NEW,@AXIS.VF_FEED_NEW,@AXIS.VF_FEED_NEW);

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

CALL
CALC_VF(@MATL.VM,@AXIS.VS_CURR,@PART.FINAL_DIM,@WHEEL.GRIT_SIZ
E,
                                    @AXIS.VF_FEED_CALC);

CALL MOVB(@AXIS.VF_FEED_CALC,@AXIS.VF_FEED_CURR,8);

CALL MOVB(@AXIS.VF_FEED_CALC,@AXIS.VF_REAL_CALC,8);
CALL MOVB(@AXIS.VF_FEED_CALC,@AXIS.VF_REAL_CURR,8);

CALL LR2R(@AXIS.VF_FEED_CURR,@TEMP_RE);
            IF TEMP_RE > 7.0 THEN DO;
                        TEMP_RE = 7.0;
                        CALL R2LR(@TEMP_RE,@AXIS.VF_FEED_CURR);
                        SK(635) = SK(635) OR 04H;
                        END;

            CALL LR2R(@AXIS.VF_FEED_CURR,@E_PAR(26).VAL);

/*-------------------------------------------------
            CALC DE THEN VW IN M/S
-------------------------------------------------*/
            CALL
CALC_DE(@PART.FINAL_DIM,@WHEEL.CURRENT_DIAM,@DE);
            CALL
CALC_VW(@DE,@AXIS.VS_CURR,@WHEEL.GRIT_SIZE,@MATL.AR,@VW);

/*-------------------------------------------------
            AND STORE IN STRUCTURE AXIS
-------------------------------------------------*/
            CALL MOVB(@VW,@AXIS.VW_CALC,8);
            CALL MOVB(@VW,@AXIS.VW_CURR,8);

/*-------------------------------------------------
            CALC VW MAX AND VW MIN:
            VW MAX = VS/Q_MIN
-------------------------------------------------*/
            CALL
LR_DIV(@AXIS.VS_CURR,@MC_DATA2.Q_MIN,@AXIS.VW_MAX);
            CALL
CALC_VW_MIN(@PART.FINAL_DIM,@MC_DATA2.MIN_REVS,@AXIS.VW_MIN);


/*-------------------------------------------------
            CHECK VW WITHIN LIMITS:
-------------------------------------------------*/
            CALL LR_CMP(@AXIS.VW_MAX,@AXIS.VW_MIN,@RESULT);

            IF RESULT > 0.0 THEN DO;
                                    CALL
LR_CMP(@AXIS.VW_CURR,@AXIS.VW_MIN,@RESULT);
                                    IF RESULT < 0.0 THEN DO;
                                                CALL
MOVB(@AXIS.VW_MIN,@AXIS.VW_CURR,8);
                                                SK(635) = SK(635) OR 08H;
                                                END;

                                    CALL
LR_CMP(@AXIS.VW_CURR,@AXIS.VW_MAX,@RESULT);
                                    IF RESULT > 0.0 THEN DO;
                                                CALL
MOVB(@AXIS.VW_MAX,@AXIS.VW_CURR,8);
                                                SK(635) = SK(635) OR 010H;
                                                END;
                                    END;

                        CALL MSRPM(@AXIS.VW_CURR,@MC_DATA1.CW_DIAM,@VW_RPM);
                        CALL MOVB(@VW_RPM,@E_PAR(56).VAL,8);

END;        /*END OF NO OVER-RIDE LOOP*/

/*-------------------------------------------------
            IF MANUAL OVER-RIDE ON THEN USE STORED VF,VW,VS
-------------------------------------------------*/

            STRING_RES = CMPB(@MC_DATA3.MAN_OR_SWITCH,@('ON
'),6);

IF STRING_RES = 0FFFFH THEN DO;

/*-------------------------------------------------
            LOAD NEW VS STRAIGHT INTO E25
            -AND VSRPM FOR DISPLAY
            -FORGET AXIS.CALC/CURR ETC
-------------------------------------------------*/
            CALL MOVB(@MC_DATA3.OR_VS,@VS_RPM,8);
            CALL LR2R(@VS_RPM,@E_PAR(25).VAL);

/*-------------------------------------------------
            LOAD STORED VW INTO E56
            -DISPLAY IN VW_RPM
            -CONVERT TO M/S
            -SAVE IN CALC AND CURR FOR ADAPTIVE CONTROL
-------------------------------------------------*/
            CALL MOVB(@MC_DATA3.OR_VW,@VW_RPM,8);
            CALL MOVB(@VW_RPM,@E_PAR(56).VAL,8);
            CALL RPMMS(@VW_RPM,@MC_DATA1.CW_DIAM,@AXIS.VW_CURR);
            CALL MOVB(@AXIS.VW_CURR,@AXIS.VW_CALC,8);

/*-------------------------------------------------
            STORE OVER-RIDE VALUE OF VF IN E26
            -SAVE IN CALC AND CURR FOR ADAPTIVE
            -DISPLAY VIA CURR
-------------------------------------------------*/

            CALL MOVB(@MC_DATA3.OR_VF,@AXIS.VF_REAL_CURR,8);
            CALL MOVB(@MC_DATA3.OR_VF,@AXIS.VF_REAL_CALC,8);

            CALL MOVB(@MC_DATA3.OR_VF,@AXIS.VF_FEED_CURR,8);
            CALL MOVB(@MC_DATA3.OR_VF,@AXIS.VF_FEED_CALC,8);

            CALL LR2R(@MC_DATA3.OR_VF,@E_PAR(26).VAL);

END;

/*-------------------------------------------------
            CALC TOTAL POWER AVAILABLE
-------------------------------------------------*/
            CALL LR2R(@VS_RPM,@TEMP_RE);
/*          P_MAX = 75.00*TEMP_RE/2000.0;*/

            P_MAX = 50.0;


/*-------------------------------------------------
            CALC HOME POSITION
-------------------------------------------------*/
            CALL CALC_POS(@MC_DATA1.BED_LEN,@PART.START_DIM,

            @MC_DATA1.CYCLE_DIST,@MC_DATA1.HOME);
            CALL MOVB(@MC_DATA1.HOME,@E_PAR(50).VAL,8);


/*-------------------------------------------------
            SAFETY POSITION
-------------------------------------------------*/
            CALL CALC_POS(@MC_DATA1.BED_LEN,@PART.START_DIM,

            @MC_DATA1.SAFETY_DIST,@MC_DATA1.SAFETY);
            CALL MOVB(@MC_DATA1.SAFETY,@E_PAR(51).VAL,8);

/*-------------------------------------------------
            SAFETY DISTANCE
-------------------------------------------------*/

            CALL MOVB(@MC_DATA1.SAFETY_DIST,@E_PAR(100).VAL,8);

/*-------------------------------------------------
            START GRIND
-------------------------------------------------*/
```

```
          CALL LR_SUB(@TEMP_LR1,@TEMP_LR1,@TEMP_LR1);
          CALL CALC_POS(@MC_DATA1.BED_LEN,@PART.START_DIM,
                        @TEMP_LR1,@MC_DATA1.STRT);
          CALL MOVB(@MC_DATA1.STRT,@E_PAR(52).VAL,8);

/*-------------------------------------------------
          TARGET POSITION
-------------------------------------------------*/
          CALL LR_SUB(@TEMP_LR1,@TEMP_LR1,@TEMP_LR1);
          CALL CALC_POS(@MC_DATA1.BED_LEN,@PART.FINAL_DIM,
                        @TEMP_LR1,@MC_DATA1.TARGET);
          CALL MOVB(@MC_DATA1.TARGET,@E_PAR(53).VAL,8);

/*-------------------------------------------------
          TRANSFER DWELL IF USED
-------------------------------------------------*/
          CALL MOVB(@MC_DATA1.DWELL,@E_PAR(54).VAL,8);

/*-------------------------------------------------
          PASS OFFSET TO E68
-------------------------------------------------*/
          CALL MOVB(@MC_DATA1.OFFSET,@E_PAR(68).VAL,8);

/*-------------------------------------------------
          PASS COMPONENT TOR TO E67
          CALL MOVB(@PART.TOR,@E_PAR(67).VAL,8);
-------------------------------------------------*/

/*-------------------------------------------------
          PASS COMPONENT ALLOWANCE TO E69
-------------------------------------------------*/
          CALL MOVB(@MC_DATA1.ALLOW,@E_PAR(69).VAL,8);

/*-------------------------------------------------
          PASS COMPONENT DIAMETER TO E87
-------------------------------------------------*/
          CALL MOVB(@PART.FINAL_DIM,@E_PAR(87).VAL,8);

/*-------------------------------------------------
          PASS CYCLE ALLOWANCE PARAMETERS TO E130,132,134,136
          TOR PARAMETERS TO E131,133,135,137
-------------------------------------------------*/
          INDEX = 0;

          DO WHILE INDEX < 4;
                    CALL
MOVB(@CYCLE(INDEX).ALLOW,@E_PAR(130+2*INDEX).VAL,8);
                    CALL
MOVB(@CYCLE(INDEX).TOR,@E_PAR(131+2*INDEX).VAL,8);
                    INDEX = INDEX + 1;
                    END;

/*-------------------------------------------------*/
END;

END CALC_VARS;

/*-------------------------------------------------
          PROCEDURE TO CALCULATE CRITICAL ENERGY
-------------------------------------------------*/
CALC_CRIT_ENERGY:
          PROCEDURE PUBLIC;

DO;

/*-------------------------------------------------
          CALC REAL VF
-------------------------------------------------*/
/*-------------------------------------------------
          IF OVER SHOOT OVER-RIDE OFF THEN CALC VF_REAL
-------------------------------------------------*/

STRING_RES = CMPB(@MC_DATA3.OS_OR_SWITCH,@('OFF    '),6);

IF STRING_RES = 0FFFFH THEN DO;
          CALL CALC_VF_REAL(@AXIS.VF_FEED_CURR,@T_INFEED,@TOR,
                            @AXIS.VF_REAL_CURR);
          END;

/*-------------------------------------------------
          IF OVER SHOOT OVER-RIDE ON THEN USE VF_FEED
-------------------------------------------------*/

STRING_RES = CMPB(@MC_DATA3.OS_OR_SWITCH,@('ON    '),6);

IF STRING_RES = 0FFFFH THEN
          CALL MOVB(@AXIS.VF_FEED_CURR,@AXIS.VF_REAL_CURR,8);

/*-------------------------------------------------
          DEPTH OF CUT AND HENCE LG
-------------------------------------------------*/
          CALL
CALC_LG(@PART.FINAL_DIM,@AXIS.VF_REAL_CURR,@AXIS.VW_CURR,
                    @DE,@CUT_DEPTH,@LG);

/*-------------------------------------------------
          CALCULATE LE
-------------------------------------------------*/
          CALL
CALC_LE(@AXIS.VS_CURR,@AXIS.VW_CURR,@CUT_DEPTH,@LG,@LE);

/*-------------------------------------------------
          CALCULATE RATIO R
-------------------------------------------------*/
          CALL  CALC_R(@MATL.DIFF,@WHEEL.DIFF,@AXIS.VW_CURR,

          @AXIS.VS_CURR,@MATL.T_COND,@WHEEL.COND,@RRR);

/*-------------------------------------------------
          CALCULATE EC
-------------------------------------------------*/
          CALL
CALC_EC(@MATL.T_CRIT,@LE,@MATL.DIFF,@AXIS.VW_CURR,
                    @MATL.T_COND,@RRR,@CUT_DEPTH,@EC);

/*-------------------------------------------------
          TRANSFER TO E PARAMETERS
-------------------------------------------------*/
          CALL MOVB(@LE,@E_PAR(75).VAL,8);
          CALL MOVB(@LG,@E_PAR(76).VAL,8);
          CALL MOVB(@RRR,@E_PAR(77).VAL,8);
          CALL MOVB(@EC,@E_PAR(64).VAL,8);
          CALL MOVB(@AXIS.VF_REAL_CURR,@E_PAR(112).VAL,8);

/*-------------------------------------------------*/
END;

END CALC_CRIT_ENERGY;

/*-------------------------------------------------
          CALCULATE UPDATED VF
-------------------------------------------------*/
VF_UPDATE:
          PROCEDURE PUBLIC;

/*-------------------------------------------------
          IF ADAPTIVE OVER-RIDE OFF THEN UPDATE VF
-------------------------------------------------*/

STRING_RES = CMPB(@MC_DATA3.ADAPT_OR_SWITCH,@('OFF    '),6);

IF STRING_RES = 0FFFFH THEN DO;

IF PROC_P_MAX > 1.5*P_NO_LOAD THEN DO;

          CALL LR2R(@AXIS.VF_FEED_CURR,@VF_CUR);
          CALL LR2R(@MC_DATA2.VF_RELAX,@VFK);

          VF_CUR = VF_CUR*(1.0+VFK*(((P_MAX-
P_NO_LOAD)/(PROC_P_MAX-P_NO_LOAD))
                    -1.0)/100.0);

          IF VF_CUR > 7.0 THEN
                    VF_CUR = 7.0;

          CALL R2LR(@VF_CUR,@AXIS.VF_FEED_CURR);
          CALL MOVB(@VF_CUR,@E_PAR(26).VAL,4);
          END;

END;

END VF_UPDATE;

/*-------------------------------------------------
          CALCULATE UPDATED VW
-------------------------------------------------*/

VW_UPDATE:
          PROCEDURE PUBLIC;
DO;

/*-------------------------------------------------
          IF ADAPTIVE OVER-RIDE OFF THEN UPDATE VW
-------------------------------------------------*/

STRING_RES = CMPB(@MC_DATA3.ADAPT_OR_SWITCH,@('OFF    '),6);

IF STRING_RES = 0FFFFH THEN DO;

/*-------------------------------------------------*/
          CALL LR2R(@EC,@EC_RE);

          IF US_MAX > EC_RE THEN DO;
                    CALL
LR_SUB(@AXIS.VW_MAX,@AXIS.VW_CURR,@TEMP_LR1);
                    CALL LR2R(@MC_DATA2.VW_RELAX,@TEMP_RE);
                    TEMP_RE = TEMP_RE/100.0;
```

```
                    CALL
LR_MUL_RE(@TEMP_LR1,@TEMP_RE,@TEMP_LR1);
                    CALL
LR_ADD(@AXIS.VW_CURR,@TEMP_LR1,@AXIS.VW_CURR);
                    END;

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

          IF US_MAX < EC_RE THEN DO;
                    CALL
LR_SUB(@AXIS.VW_CURR,@AXIS.VW_MIN,@TEMP_LR1);
                    CALL LR2R(@MC_DATA2.VW_RELAX,@TEMP_RE);
                    TEMP_RE = TEMP_RE/100.0;
                    CALL
LR_MUL_RE(@TEMP_LR1,@TEMP_RE,@TEMP_LR1);
                    CALL
LR_ADD(@AXIS.VW_MIN,@TEMP_LR1,@AXIS.VW_CURR);
                    END;

          CALL MSRPM(@AXIS.VW_CURR,@MC_DATA1.CW_DIAM,@VW_RPM);
          CALL MOVB(@VW_RPM,@E_PAR(56).VAL,8);

END;
END;

END VW_UPDATE;

/*-------------------------------------------------
          ITERATIVE PROCEDURE TO CALCULATE OVERSHOOT
-------------------------------------------------*/

CALC_OS:
          PROCEDURE (ERROR_PTR) PUBLIC;

DECLARE ERROR_PTR POINTER;

DO;

/*-------------------------------------------------
          IF OVER SHOOT OVER-RIDE OFF THEN CALC OS
-------------------------------------------------*/

STRING_RES = CMPB(@MC_DATA3.OS_OR_SWITCH,@('OFF    '),6);

IF STRING_RES = 0FFFFH THEN DO;

/*-------------------------------------------------*/
CALL MOVB(@E_PAR(70).VAL,@XT,8);
CALL LR2R(@E_PAR(119).VAL,@TEMP_RE);
INDEX = UNSIGN(FIX(TEMP_RE))-1;

IF INDEX < 0 OR INDEX > 3 THEN INDEX = 3;

/*-------------------------------------------------
          ON FIRST PECK SET TOR_RATIO = 1
-------------------------------------------------*/

IF INDEX = 0 THEN DO;
          TEMP_RE = 1.0;
          CALL R2LR(@TEMP_RE,@TOR_RATIO);
          END;

CALL MOVB(@CYCLE(INDEX).TOR,@TOR,8);
CALL LR_MUL(@TOR,@TOR_RATIO,@TOR);

CALL MOVB(@TOR_RATIO,@E_PAR(92).VAL,8);

CALL MOVB(@CYCLE(INDEX).TOR,@E_PAR(67).VAL,8);

CALL
CALC_T_INFEED(@MC_DATA1.DWELL,@TOR,@AXIS.VF_FEED_CURR,@XT,
                              @T_INFEED,@OS_DIST,ERROR_PTR);

CALL MOVB(@T_INFEED,@E_PAR(90).VAL,8);
CALL MOVB(@OS_DIST,@E_PAR(91).VAL,8);

END;

/*-------------------------------------------------
          IF OS OVER-RIDE ON THEN USE STORED OS
-------------------------------------------------*/

STRING_RES = CMPB(@MC_DATA3.OS_OR_SWITCH,@('ON     '),6);

IF STRING_RES = 0FFFFH THEN DO;

CALL MOVB(@MC_DATA3.OR_OS,@E_PAR(91).VAL,8);

END;
/*-------------------------------------------------*/
END;

END CALC_OS;

/*-------------------------------------------------
          ITERATIVE PROCEDURE TO CALCULATE COMPONENT TOR
          XT = TOTAL REQUIRED TO BE REMOVED -PREVIOUSLY STORED
          XA = TOTAL ACTUALLY REMOVED
          TOR = RECALCULATED TOR -PREVIOUSLY LOADED
-------------------------------------------------*/

CALC_TOR:
          PROCEDURE (ERROR_PTR) PUBLIC;

DECLARE ERROR_PTR POINTER;

DO;
/*-------------------------------------------------*/
CALL MOVB(@E_PAR(73).VAL,@XA,8);

CALL
CALC_CMPT_TOR(@MC_DATA1.DWELL,@TOR,@AXIS.VF_FEED_CURR,@XT,@XA,
@T_INFEED,
                              ERROR_PTR);

CALL MOVB(@TOR,@E_PAR(74).VAL,8);

/*-------------------------------------------------
          CALC TOR_RATIO = TOR/CYCLE(INDEX).TOR
-------------------------------------------------*/

CALL LR2R(@E_PAR(119).VAL,@TEMP_RE);
INDEX = UNSIGN(FIX(TEMP_RE))-1;

IF INDEX < 0 OR INDEX > 3 THEN INDEX = 3;

CALL LR_DIV(@TOR,@CYCLE(INDEX).TOR,@TOR_RATIO);

/*-------------------------------------------------
          CHECK TOR_RATIO > 0 AND < 2
-------------------------------------------------*/

CALL LR2R(@TOR_RATIO,@TEMP_RE);

IF TEMP_RE > 2.0 THEN TEMP_RE = 2.0;
IF TEMP_RE < 0.0 THEN TEMP_RE = 0.0;

CALL R2LR(@TEMP_RE,@TOR_RATIO);

END;

END CALC_TOR;

/*-------------------------------------------------
          ITERATIVE PROCEDURE TO CALCULATE OVERSHOOT
          BASED ON RECALCULATED TIME CONSTANT
          XT = TOTAL REQUIRED TO BE REMOVED
          TOR = RECALCULATED TOR -PREVIOUSLY LOADED
-------------------------------------------------*/

CALC_LEARNED_OS:
          PROCEDURE (ERROR_PTR) PUBLIC;

DECLARE ERROR_PTR POINTER;

DO;

/*-------------------------------------------------
          IF OVER SHOOT OVER-RIDE OFF THEN CALC OS
-------------------------------------------------*/

STRING_RES = CMPB(@MC_DATA3.OS_OR_SWITCH,@('OFF    '),6);
IF STRING_RES = 0FFFFH THEN DO;

/*-------------------------------------------------*/
CALL MOVB(@E_PAR(70).VAL,@XT,8);

CALL
CALC_T_INFEED(@MC_DATA1.DWELL,@TOR,@AXIS.VF_FEED_CURR,@XT,@T_I
NFEED,@OS_DIST,
                              ERROR_PTR);

CALL MOVB(@T_INFEED,@E_PAR(90).VAL,8);
CALL MOVB(@OS_DIST,@E_PAR(91).VAL,8);

END;

/*-------------------------------------------------
          IF OS OVER-RIDE ON THEN USE STORED OS
-------------------------------------------------*/

STRING_RES = CMPB(@MC_DATA3.OS_OR_SWITCH,@('ON     '),6);

IF STRING_RES = 0FFFFH THEN DO;

CALL MOVB(@MC_DATA3.OR_OS,@E_PAR(91).VAL,8);

END;
/*-------------------------------------------------*/
END;

END CALC_LEARNED_OS;

END CALC;
```

# MATHS.ASM

## Library of assembler functions to handle long real floating point calculations.

---

```
NAME LR_MATHS

;-------------------------------------------------
;         An assembler routine to handle
;         floating point calculations on
;         the 8087.
;
;         (c) Sean Kelly March 1987
;
;-------------------------------------------------


PUBLIC LR_ADD,LR_SUB,LR_MUL,LR_DIV,LR_CMP,
&
CALC_VF,CALC_VF_REAL,CALC_DE,CALC_VW,MSRPM,RPMMS,CALC_LG,CALC_
LE,
& LR2R,R2LR,R21LR3,R2W,DW2LR,CALC_US,LR_EXP,LR_POWER_RE,
& LR_MUL_RE,CALC_R,CALC_EC,CALC_VW_MIN,
& CALC_POS,CALC_SPEED,CALC_VF_ACT,FILTER,
& CALC_T_INFEED,CALC_CMPT_TOR,SIG_FIG4

EXTRN INITFP: FAR
EXTRN INIT87: FAR

LR_DATA SEGMENT PUBLIC 'DATA'

IT_COUNT  DW ?
XI                DQ ?
XD                DQ ?
XT                DQ ?
XA                DQ ?
XTIT              DQ ?
TIME1             DQ ?
TMIN              DQ ?
TMAX              DQ ?
WIND              DQ ?
T_DWELL           DQ ?
VF                DQ ?
TOR               DQ ?
MAX_ERROR  DQ ?
XT_ERROR   DQ ?

TEMP_LR1   DQ ?
TEMP_LR2   DQ ?
TEMP_LR3   DQ ?
TEMP_INT   DW ?
STATUS_WORD DW ?
CONTROL_WORD DW ?
COUNTER DW ?

LR_DATA ENDS

LR_MATHS SEGMENT PUBLIC 'CODE'

ASSUME CS:LR_MATHS,DS:LR_DATA

BIT15      DD 8000H
INT10000 DW 10000
INT1000 DW 1000
INT100     DW 100
INT20      DW 20
INT10      DW 10
INT3       DW 3
INT2       DW 2
EXP        DQ 2.71828
K0         DQ 0.33333
K1         DQ -0.0205
K2         DQ -0.216
K3         DQ 4.95
K5         DQ 0.887
SECS       DW 60
ROOT2      DQ 1.4142136

;-------------------------------------------------
;         CALCULATE SPECIFIC ENERGY FROM
;
;         US=P/(0.5*PI*DW*LEN*VF)
;
;-------------------------------------------------
CALC_US             PROC FAR

PUSH       BP
PUSH       ES
PUSH       SI
MOV        BP,SP
PUSH       DS
MOV        AX,LR_DATA
MOV        DS,AX
;
;-------------------------------------------------
LES        SI,DWORD PTR [BP+26]          ;
FLD        ES:DWORD PTR [SI]    ; LOAD POWER
LES        SI,DWORD PTR [BP+22]          ;
FLD        ES:QWORD PTR [SI]    ; LOAD LENGTH
FDIV                                     ; AND DIVIDE

LES        SI,DWORD PTR [BP+18]          ;
FLD        ES:QWORD PTR [SI]    ;
FDIV                                     ; DIVIDE BY DIAMETER

LES        SI,DWORD PTR [BP+14]          ;
FLD        ES:QWORD PTR [SI]    ;
FDIV                                     ; DIVIDE BY REAL VF

FLDPI                                    ;
FDIV                                     ; DIVIDE BY PI

FADD       ST,ST                         ; DIVIDE BY 1/2
FIMUL      INT1000                       ; AND SCALE

LES        SI,DWORD PTR [BP+10]          ;
FSTP       ES:DWORD PTR [SI]    ; POP OFF INTO US

;-------------------------------------------------
POP        DS
POP        SI
POP        ES
POP        BP
RET        20

CALC_US ENDP

;-------------------------------------------------
;         CALCULATE AND STORE EFFECTIVE DIAM
;         DEPTH OF CUT AND LG FROM
;
;         A = 0.5*PI*DW*VF/(VW*1000)
;         LG = SQRT(A*DE)
;
;-------------------------------------------------
CALC_LG             PROC FAR

PUSH       BP
PUSH       ES
PUSH       SI
MOV        BP,SP
PUSH       DS
MOV        AX,LR_DATA
MOV        DS,AX

;-------------------------------------------------
LES        SI,DWORD PTR [BP+30]          ;
FLD        ES:QWORD PTR [SI]    ; LOAD PART DIAMETER
LES        SI,DWORD PTR [BP+26]          ;
FLD        ES:QWORD PTR [SI]    ; LOAD CURRENT FEED RATE
FMUL                                     ; AND MULTIPLY

FLDPI                                    ;
FMUL                                     ; * BY PI

LES        SI,DWORD PTR [BP+22]          ;
FLD        ES:QWORD PTR [SI]    ; DIVIDE BY VW
FDIV

FIDIV      INT2                          ; DIVIDE BY TWO
FIDIV      INT1000                       ; SCALE

LES        SI,DWORD PTR [BP+14]          ;
FST        ES:QWORD PTR [SI]    ; AND STORE AS DEPTH OF CUT

;-------------------------------------------------
LES        SI,DWORD PTR [BP+18]          ;
FLD        ES:QWORD PTR [SI]    ; LOAD EFFECTIVE DIAM (DE)

FMUL                                     ; * BY STACK TOP
FSQRT                                    ; ROOT IT

LES        SI,DWORD PTR [BP+10]          ;
FSTP       ES:QWORD PTR [SI]    ;AND SAVE AS LG

;-------------------------------------------------
POP        DS
POP        SI
POP        ES
POP        BP
RET        24

CALC_LG ENDP
```

```
;-------------------------------------------------
;       CALCULATE LE FROM THE HORRID EQUATION
;
;       LE = LG*(4.95*((VS/VW)**-0.216)*EXP(-
0.0205*((VS/VW)**1/3)*LN(A)))
;
;-------------------------------------------------
CALC_LE             PROC FAR

PUSH    BP
PUSH    ES
PUSH    SI
MOV     BP,SP
PUSH    DS
MOV     AX,LR_DATA
MOV     DS,AX

;-------------------------------------------------
LES     SI,DWORD PTR [BP+26]        ;
FLD     ES:QWORD PTR [SI]   ; LOAD VS
LES     SI,DWORD PTR [BP+22]        ;
FLD     ES:QWORD PTR [SI]   ; LOAD VW
FDIV                                ; CALC VS/VW

;-------------------------------------------------
FST     TEMP_LR1            ; STORE VS/VW IN TEMP1 BUT
LEAVE ON STACK
FLD     K0                          ; LOAD UP 1/3
FSTP    TEMP_LR2            ; STORE 1/3 IN TEMP2
CALL    LR_POWER_LR                 ; CALCULATE
(VS/VW)**1/3
FLD     TEMP_LR3            ; TEMP3 = RESULT ON TO STACK

;-------------------------------------------------
FLD1                                ; LOAD 1
LES     SI,DWORD PTR [BP+18]        ;
FLD     ES:QWORD PTR [SI]   ; LOAD DEPTH OF CUT = A
FYL2X                               ; CALC 1*LOG2(A)
FLDL2E                              ; LOAD LOG2(e) ONTO
STACK
FDIV                                ; DIVIDE TO GIVE
LOGe(A)

;-------------------------------------------------
FMUL                                ; (VS/VW)**3 *LN(CUT
DEPTH)
FMUL    K1                          ; * -0.0205

;-------------------------------------------------
FSTP    TEMP_LR2            ; SAVE ALL IN TEMP2
FLD     EXP                         ; LOAD e = 2.7128
FSTP    TEMP_LR1            ; STORE IN TEMP1
CALL    LR_POWER_LR                 ; CALCULATE e**TEMP2
                                    ; RESULT IN TEMP3
;-------------------------------------------------
FSTP    TEMP_LR1            ; POP OF VS/VW
FLD     TEMP_LR3            ; SAVE PREVIOUS RESULT ON STACK
FLD     K2                          ; LOAD -0.216
FSTP    TEMP_LR2            ; STORE IN TEMP2
CALL    LR_POWER_LR                 ; (VS/VW)**-0.216
FLD     TEMP_LR3            ; RESULT ON TO STACK

;-------------------------------------------------
FMUL                                ; * BOTH RESULTS SO
FAR
```

```
FMUL    K3                          ; * BY 4.95

LES     SI,DWORD PTR [BP+14]
FLD     ES:QWORD PTR [SI]   ; LOAD LG
FMUL                                ; * BY LG

;-------------------------------------------------
LES     SI,DWORD PTR [BP+10]
FSTP    ES:QWORD PTR [SI]   ; POP OFF INTO LE...

;-------------------------------------------------

POP     DS
POP     SI
POP     ES
POP     BP
RET     20

CALC_LE ENDP


;-------------------------------------------------
;       CALCULATE CRITICAL SPECIFIC ENERGY FROM
;
;       EC =
TM_CRIT*SQRT(LE/(VW*TM_DIFF))*TM_COND/(0.887*R*A)
;       YUK!!!
;
;-------------------------------------------------
CALC_EC             PROC FAR

PUSH    BP
PUSH    ES
PUSH    SI
MOV     BP,SP
PUSH    DS
MOV     AX,LR_DATA
MOV     DS,AX

;-------------------------------------------------
LES     SI,DWORD PTR [BP+34]        ;
FLD     ES:QWORD PTR [SI]   ; LOAD UP LE
LES     SI,DWORD PTR [BP+30]        ;
FLD     ES:QWORD PTR [SI]   ; LOAD THERMAL DIFFUSIVITY
FDIV                                ; AND DIVIDE...

LES     SI,DWORD PTR [BP+26]        ; DIVIDE BY CURRENT
VW
FLD     ES:QWORD PTR [SI]   ;
FDIV

FIDIV   INT1000                     ; CONVERT TO mm
FSQRT                               ; AND ROOT THE WHOLE
SHEBANG

;-------------------------------------------------
LES     SI,DWORD PTR [BP+22]        ;
FLD     ES:QWORD PTR [SI]   ; LOAD THERMAL CONDUCTIVITY
FMUL                                ; * BY ST

FIDIV   INT1000                     ; CONVERT TO mm

;-------------------------------------------------
LES     SI,DWORD PTR [BP+18]        ;
FLD     ES:QWORD PTR [SI]   ; LOAD RATIO R
FDIV                                ; AND DIVIDE
```

```
LES     SI,DWORD PTR [BP+14]        ;
FLD     ES:QWORD PTR [SI]   ; LOAD CUT DEPTH
FDIV                                ; AND DIVIDE

FDIV    K5                          ; DIVIDE BY 0.887

LES     SI,DWORD PTR [BP+38]        ;
FLD     ES:QWORD PTR [SI]   ; LOAD CRITICAL TEMP
FMUL                                ; * BY THE REST

LES     SI,DWORD PTR [BP+10]        ;
FSTP    ES:QWORD PTR [SI]   ; AND HEY PRESTO EC*

;-------------------------------------------------
POP     DS
POP     SI
POP     ES
POP     BP
RET     32

CALC_EC ENDP


;-------------------------------------------------
;       CALCULATE THE RATIO R AS OF THE MESS -
;
;       1/R = 1 +
SQRT((TM_DIFF*VS)/(TS_DIFF*VW))*TS_COND/TM_COND
;
;-------------------------------------------------
CALC_R              PROC FAR

PUSH    BP
PUSH    ES
PUSH    SI
MOV     BP,SP
PUSH    DS
MOV     AX,LR_DATA
MOV     DS,AX

;-------------------------------------------------
LES     SI,DWORD PTR [BP+34]        ;
FLD     ES:QWORD PTR [SI]   ; LOAD COMPONENT DIFFUSIVITY
LES     SI,DWORD PTR [BP+30]        ;
FLD     ES:QWORD PTR [SI]   ; LOAD WHEEL DIFF.
FDIV                                ; AND DIVIDE

LES     SI,DWORD PTR [BP+22]        ;
FLD     ES:QWORD PTR [SI]   ; LOAD CURRENT VS
FMUL                                ; AND MULTIPLY

LES     SI,DWORD PTR [BP+26]        ;
FLD     ES:QWORD PTR [SI]   ; LOAD CURRENT VW
FDIV                                ; AND DIVIDE

FSQRT                               ; SQRT THE LOT SO FAR

;-------------------------------------------------
LES     SI,DWORD PTR [BP+14]        ;
FLD     ES:QWORD PTR [SI]   ; LOAD WHEEL CONDUCTIVITY
FMUL                                ; AND MULT

LES     SI,DWORD PTR [BP+18]        ;
FLD     ES:QWORD PTR [SI]   ; LOAD COMPONENT COND
FDIV                                ; AND DIV
```

```
;-------------------------------------------------
FLD1                                    ;
FADD                                    ; ADD 1
FLD1                                    ;
FDIVR                                   ; INVERT

LES     SI,DWORD PTR [BP+10]            ;
FSTP    ES:QWORD PTR [SI]    ; AND STORE R

;-------------------------------------------------
POP     DS
POP     SI
POP     ES
POP     BP
RET     28

CALC_R ENDP

;-------------------------------------------------
;       CALCULATE TIME TO GRIND XT OFF COMPONENT
;
;-------------------------------------------------
CALC_T_INFEED PROC FAR

PUSH    BP
PUSH    ES
PUSH    SI
MOV     BP,SP
PUSH    DS
MOV     AX,LR_DATA
MOV     DS,AX

;-------------------------------------------------
LES     SI,DWORD PTR [BP+22]            ;
FLD     ES:QWORD PTR [SI]    ; LOAD XT
FSTP    XT

LES     SI,DWORD PTR [BP+26]            ;
FLD     ES:QWORD PTR [SI]    ; LOAD VF
FSTP    VF

LES     SI,DWORD PTR [BP+30]            ;
FLD     ES:QWORD PTR [SI]    ; LOAD TOR
FSTP    TOR

LES     SI,DWORD PTR [BP+34]            ;
FLD     ES:QWORD PTR [SI]    ; DWELL
FSTP    T_DWELL

;-------------------------------------------------
FLD     TOR                             ; LOAD TOR ON STACK
FLDZ                                    ; LOAD ZERO
FCOMPP                                  ; COMPARE

;-------------------------------------------------
FSTSW   STATUS_WORD
FWAIT
MOV     AH,BYTE PTR STATUS_WORD+1
SAHF                                    ; LOAD FLAGS

JBE     TOR_OK1                         ; IF 0 =< TOR THEN
TOR OK
JMP     IT_FAIL

;-------------------------------------------------
```

```
TOR_OK1:

FLD     XT                              ; LOAD XT ON STACK
FLDZ                                    ; LOAD ZERO
FCOMPP                                  ; COMPARE

;-------------------------------------------------
FSTSW   STATUS_WORD
FWAIT
MOV     AH,BYTE PTR STATUS_WORD+1
SAHF                                    ; LOAD FLAGS

JBE     XT_OK1                          ; IF 0 =< XT THEN XT
OK
JMP     XT_FAIL

;-------------------------------------------------
XT_OK1:

FLDZ                                    ; LOAD ZERO
FSTP    TMIN                            ; SAVE IN T MIN

FLD     XT
FLD     VF
FDIV
FST     TIME1                           ; T1 = XT/VF
FIMUL   INT100                          ; 100*T1
FSTP    TMAX                            ; SAVE IN T MAX

FLD     XT
FIDIV   INT100
FSTP    MAX_ERROR           ;CALC MAX ERROR IN XT AS 1/100

;-------------------------------------------------
MOV     IT_COUNT,0

T_LOOP:
        CMP     IT_COUNT,50
        JNE     IT_AGAIN
        JMP     IT_FAIL

IT_AGAIN:
        FLD     TIME1
        FDIV    TOR                             ; T1/TOR
        FCHS                                    ; -T1/TOR
        FSTP    TEMP_LR2            ; STORE IN LR2

        FLD     EXP
        FSTP    TEMP_LR1
        CALL    LR_POWER_LR                     ; EXP (-
T1/TOR)

        FLD1
        FSUB    TEMP_LR3

        FMUL    VF
        FMUL    TOR                             ;VF*TOR*(1-
EXP(-T1/TOR)
        FSTP    WIND                            ;SAVE

;-------------------------------------------------
        FLD     VF
        FMUL    TIME1
```

```
        FSUB    WIND                            ;XI = VF*T1
- WIND
        FSTP    XI

;-------------------------------------------------
        FLD     T_DWELL
        FDIV    TOR
        FCHS                                    ;
-TDWELL/TOR
        FSTP    TEMP_LR2            ; STORE IN LR2

        FLD     EXP
        FSTP    TEMP_LR1
        CALL    LR_POWER_LR                     ; EXP (-
TDWELL/TOR)

        FLD1
        FSUB    TEMP_LR3            ;1-EXP(-TDWELL/TOR)

        FMUL    WIND
        FSTP    XD                              ;AND
XD=WIND*1-EXP(-TDWELL/TOR))

;-------------------------------------------------
        FLD     XI
        FADD    XD
        FSTP    XTIT                            ; STORE
XI+XD IN XTOTAL ITERATIVE

;-------------------------------------------------
        FLD     MAX_ERROR
        FADD    XT

        FLD     XTIT
        FCOMPP                                  ;COMPARE
ACTUAL XTIT WITH XT + ERR

;-------------------------------------------------
        FSTSW   STATUS_WORD
        FWAIT
        MOV     AH,BYTE PTR STATUS_WORD+1
        SAHF                                    ;LOAD FLAGS

        JA      DEC_T1                          ;IF XTIT >
XT + ERROR DEC T1

;-------------------------------------------------
        FLD     XT
        FSUB    MAX_ERROR

        FLD     XTIT
        FCOMPP                                  ;COMPARE
ACTUAL XTIT WITH XT - ERR

;-------------------------------------------------
        FSTSW   STATUS_WORD
        FWAIT
        MOV     AH,BYTE PTR STATUS_WORD+1
        SAHF                                    ;LOAD FLAGS

        JB      INC_T1                          ;IF XTIT <
XT - ERROR INC T1

        JMP     SAVE_T_INFEED                   ;OTHERWISE
ITERATION COMPLETE
;-------------------------------------------------
```

```
DEC_T1:
        FLD     TIME1
        FST     TMAX                            ;NEW T_MAX
= OLD T1

        FADD    TMIN
        FIDIV   INT2
        FSTP    TIME1

        INC     IT_COUNT            ;INC LOOP COUNTER
        JMP     T_LOOP                          ;ROUND
AGAIN

;-----------------------------------------------
INC_T1:
        FLD     TIME1
        FST     TMIN                            ;NEW T_MIN
= OLD T1

        FADD    TMAX
        FIDIV   INT2
        FSTP    TIME1

        INC     IT_COUNT            ;INC LOOP COUNTER
        JMP     T_LOOP                          ;ROUND
AGAIN

;-----------------------------------------------
SAVE_T_INFEED:

FLD     TIME1
LES     SI,DWORD PTR [BP+18]            ;
FSTP    ES:QWORD PTR [SI]    ; STORE IN T_INFEED

MOV     AL,00
LES     SI,DWORD PTR [BP+10]            ;
MOV     ES:[SI],AL          ; ZERO ERROR FLAG

JMP     OS_DIST

;-----------------------------------------------
IT_FAIL:

FLDZ
LES     SI,DWORD PTR [BP+18]            ;
FSTP    ES:QWORD PTR [SI]    ; STORE 0 IN T_INFEED

FLDZ
LES     SI,DWORD PTR [BP+14]            ;
FSTP    ES:QWORD PTR [SI]    ; STORE 0 IN OS_DIST

MOV     AL,255
LES     SI,DWORD PTR [BP+10]            ;
MOV     ES:[SI],AL                      ; ERROR FLAG = 255

JMP     END_T_INFEED

;-----------------------------------------------
XT_FAIL:

FLDZ
LES     SI,DWORD PTR [BP+18]            ;
FSTP    ES:QWORD PTR [SI]    ; STORE 0 IN T_INFEED

FLDZ
LES     SI,DWORD PTR [BP+14]            ;
FSTP    ES:QWORD PTR [SI]    ; STORE 0 IN OS_DIST

MOV     AL,0
LES     SI,DWORD PTR [BP+10]            ;
MOV     ES:[SI],AL                      ; ERROR FLAG = 00

JMP     END_T_INFEED

;-----------------------------------------------
OS_DIST:

FLD     VF
FLD     TIME1
FMUL
FLD     XT
FSUB                                    ;CALC OS AS VF*T1 -
XT
LES     SI,DWORD PTR [BP+14]            ;
FSTP    ES:QWORD PTR [SI]    ;

;-----------------------------------------------
END_T_INFEED:

POP     DS
POP     SI
POP     ES
POP     BP
RET     28

CALC_T_INFEED ENDP

;-----------------------------------------------
;       CALCULATE TIME CONSTANT OF COMPONENT FROM
;       ACTUAL GRINDING DATA
;-----------------------------------------------
CALC_CMPT_TOR PROC FAR

PUSH    BP
PUSH    ES
PUSH    SI
MOV     BP,SP
PUSH    DS
MOV     AX,LR_DATA
MOV     DS,AX

;-----------------------------------------------
LES     SI,DWORD PTR [BP+14]            ;
FLD     ES:QWORD PTR [SI]    ; INFEED TIME
FSTP    TIME1

LES     SI,DWORD PTR [BP+18]            ;
FLD     ES:QWORD PTR [SI]    ; LOAD XA
FSTP    XA

LES     SI,DWORD PTR [BP+22]            ;
FLD     ES:QWORD PTR [SI]    ; LOAD XT
FSTP    XT

LES     SI,DWORD PTR [BP+26]            ;
FLD     ES:QWORD PTR [SI]    ; LOAD VF
FSTP    VF

LES     SI,DWORD PTR [BP+30]            ;
FLD     ES:QWORD PTR [SI]    ; LOAD TOR
FSTP    TOR

LES     SI,DWORD PTR [BP+34]            ;
FLD     ES:QWORD PTR [SI]    ; DWELL
FSTP    T_DWELL

;-----------------------------------------------
FLD     TOR                             ; SAVE TOR ON STACK
IN CASE OF DISASTER
FLDZ                                    ; LOAD ZERO
FCOMP                                   ; COMPARE

;-----------------------------------------------
FSTSW   STATUS_WORD
FWAIT
MOV     AH,BYTE PTR STATUS_WORD+1
SAHF                                    ; LOAD FLAGS

JB      TOR_OK                          ; IF 0 < TOR THEN TOR
OK
FSTP    ST                              ; POP OFF OLD TOR
FILD    INT10                           ; LOAD 10 FOR TOR
ONTO STACK
FST     TOR                             ; STORE

;-----------------------------------------------
TOR_OK:

FLD     XT                              ; LOAD X TARGET
FLDZ                                    ; LOAD ZERO
FCOMPP                                  ; COMPARE VALUES AND
CLEAR

;-----------------------------------------------
FSTSW   STATUS_WORD
FWAIT
MOV     AH,BYTE PTR STATUS_WORD+1
SAHF                                    ; LOAD FLAGS

JB      XT_OK                           ; IF 0 < XT THEN XT
OK
JMP     IT_TOR_FAIL                     ; OTHERWISE FAIL

;-----------------------------------------------
XT_OK:

FLD     XA                              ; LOAD X ACTUAL
FLDZ                                    ; LOAD ZERO
FCOMPP                                  ; COMPARE VALUES AND
CLEAR

;-----------------------------------------------
FSTSW   STATUS_WORD
FWAIT
MOV     AH,BYTE PTR STATUS_WORD+1
SAHF                                    ; LOAD FLAGS

JB      XA_OK                           ; IF 0 < XA THEN XA
OK
JMP     IT_TOR_FAIL                     ; OTHERWISE FAIL

;-----------------------------------------------
XA_OK:

FLDZ                                    ; LOAD ZERO
FSTP    TMIN                            ; SAVE IN TOR MIN
```

```
FLD       TOR
FIMUL     INT100                    ; 100*TOR
FSTP      TMAX                      ; SAVE IN TOR MAX

FLD       XA
FIDIV     INT100
FSTP      MAX_ERROR          ;CALC MAX ERROR IN XA AS 1/100

;-------------------------------------------------
MOV       IT_COUNT,0

TOR_LOOP:
          CMP       IT_COUNT,50
          JNE       IT_TOR_AGAIN
          JMP       IT_TOR_FAIL

IT_TOR_AGAIN:
          FLD       TIME1
          FLD       TOR
          FDIV                                  ; T1/TOR
          FCHS                                  ; -T1/TOR
          FSTP      TEMP_LR2          ; STORE IN LR2

          FLD       EXP
          FSTP      TEMP_LR1
          CALL      LR_POWER_LR                 ; EXP (-
T1/TOR)

          FLD1
          FLD       TEMP_LR3                    ;1-EXP(-
          FSUB
T1/TOR)

          FLD       VF
          FMUL
          FLD       TOR
          FMUL                                  ;VF*TOR*(1-
EXP(-T1/TOR)
          FST       WIND                        ;SAVE AND
LEAVE ON STACK

;-------------------------------------------------
          FLD       VF
          FLD       TIME1
          FMUL                                  ;VF*T1
          FSUBR                                 ;XI = VF*T1
- WIND
          FSTP      XI

;-------------------------------------------------
          FLD       T_DWELL
          FLD       TOR
          FDIV                                  ;
TDWELL/TOR
          FCHS                                  ;
-TDWELL/TOR
          FSTP      TEMP_LR2          ; STORE IN LR2

          FLD       EXP
          FSTP      TEMP_LR1
          CALL      LR_POWER_LR                 ; EXP (-
TDWELL/TOR)

          FLD1
          FLD       TEMP_LR3
          FSUB                                  ;1-EXP(-
TDWELL/TOR)

          FLD       WIND
          FMUL
          FSTP      XD                          ;AND
XD=WIND*1-EXP(-TDWELL/TOR))

;-------------------------------------------------
          FLD       XI
          FLD       XD
          FADD
          FSTP      XTIT                        ; STORE
XI+XD IN X TOTAL ITERATIVE

;-------------------------------------------------
          FLD       MAX_ERROR
          FLD       XA
          FADD                                  ;LOAD XA +
ERROR

          FLD       XTIT
          FCOMPP                                ;COMPARE
ACTUAL XTIT WITH XA + ERR

;-------------------------------------------------
          FSTSW     STATUS_WORD
          FWAIT
          MOV       AH,BYTE PTR STATUS_WORD+1
          SAHF                                  ;LOAD FLAGS

          JA        INC_TOR                     ;IF XTIT >
XA + ERROR INC TOR

;-------------------------------------------------
          FLD       XA
          FLD       MAX_ERROR
          FSUB                                  ;LOAD XA -
ERROR

          FLD       XTIT
          FCOMPP                                ;COMPARE
ACTUAL XTIT WITH XA - ERR

;-------------------------------------------------
          FSTSW     STATUS_WORD
          FWAIT
          MOV       AH,BYTE PTR STATUS_WORD+1
          SAHF                                  ;LOAD FLAGS

          JB        DEC_TOR                     ;IF XTIT <
XA - ERROR DEC TOR

          JMP       SAVE_TOR            ;OTHERWISE ITERATION
COMPLETE
;-------------------------------------------------
DEC_TOR:
          FLD       TOR
          FST       TMAX                        ;NEW T_MAX
- OLD TOR

          FLD       TMIN                        ;TOR =
          FADD
(TOR+TMIN)/2
          FIDIV     INT2
          FSTP      TOR

          INC       IT_COUNT            ;INC LOOP COUNTER
          JMP       TOR_LOOP            ;ROUND AGAIN

;-------------------------------------------------
INC_TOR:
          FLD       TOR
          FST       TMIN                        ;NEW T_MIN
= OLD TOR

          FLD       TMAX
          FADD                                  ;TOR =
(TOR+TMAX)/2
          FIDIV     INT2
          FSTP      TOR

          INC       IT_COUNT            ;INC LOOP COUNTER
          JMP       TOR_LOOP            ;ROUND AGAIN

;-------------------------------------------------
SAVE_TOR:

FLD       TOR

LES       SI,DWORD PTR [BP+30]                ;
FSTP      ES:QWORD PTR [SI]     ; STORE IN TOR

MOV       AL,00
LES       SI,DWORD PTR [BP+10]                ;
MOV       ES:[SI],AL                          ; ZERO ERROR FLAG
FSTP      ST                                  ; CLEAR STACK OF
ORIGINAL TOR
JMP       END_CMPT_TOR

;-------------------------------------------------
IT_TOR_FAIL:

LES       SI,DWORD PTR [BP+30]                ;
FSTP      ES:QWORD PTR [SI]     ; STORE ORIGINAL TOR

MOV       AL,255
LES       SI,DWORD PTR [BP+10]                ;
MOV       ES:[SI],AL                          ; ERROR FLAG = 255

;-------------------------------------------------
END_CMPT_TOR:

POP       DS
POP       SI
POP       ES
POP       BP
RET       28

CALC_CMPT_TOR ENDP

;-------------------------------------------------
;         CONVERT M/S TO RPM
;
;         RPM = 60*MS/(PI*DW)
;
;-------------------------------------------------

MSRPM               PROC FAR
```

```
PUSH    BP
PUSH    ES
PUSH    SI
MOV     BP,SP
PUSH    DS
MOV     AX,LR_DATA
MOV     DS,AX

;-------------------------------------------------
LES     SI,DWORD PTR [BP+18]            ;
FLD     ES:QWORD PTR [SI]    ; LOAD M/S VALUE

FIMUL   SECS                            ; GET M/MIN

FLDPI                                   ; LOAD PI
FDIV                                    ; AND DIV

LES     SI,DWORD PTR [BP+14]            ;
FLD     ES:QWORD PTR [SI]    ; LOAD DIAMETER IN mm
FDIV                                    ; DIVIDE

FIMUL   INT1000                         ; CONVERT TO M

LES     SI,DWORD PTR [BP+10]            ;
FSTP    ES:QWORD PTR [SI]    ; STORE IN RPM

;-------------------------------------------------
POP     DS
POP     SI
POP     ES
POP     BP
RET     12

MSRPM ENDP

;-------------------------------------------------
;       CONVERT RPM TO M/S
;
;       MS = RPM*PI*DIAM/60
;
;-------------------------------------------------

RPMMS           PROC FAR

PUSH    BP
PUSH    ES
PUSH    SI
MOV     BP,SP
PUSH    DS
MOV     AX,LR_DATA
MOV     DS,AX

;-------------------------------------------------
LES     SI,DWORD PTR [BP+18]            ;
FLD     ES:QWORD PTR [SI]    ; LOAD RPM VALUE

FIDIV   SECS                            ; GET RPS

FLDPI                                   ; LOAD PI
FMUL                                    ; AND MUL

LES     SI,DWORD PTR [BP+14]            ;
FLD     ES:QWORD PTR [SI]    ; LOAD DIAMETER IN mm
FMUL                                    ; MUL

FIDIV   INT1000                         ; CONVERT TO M

LES     SI,DWORD PTR [BP+10]            ;
FSTP    ES:QWORD PTR [SI]    ; STORE IN MS

;-------------------------------------------------
POP     DS
POP     SI
POP     ES
POP     BP
RET     12

RPMMS ENDP


;-------------------------------------------------
;       CALCULATE ACTUAL INFEED SPEED FORM
;
;       VF = 1000*PROFILE_SPEED/CLOCK(ms)
;
;-------------------------------------------------
CALC_VF_ACT                     PROC FAR

PUSH    BP
PUSH    ES
PUSH    SI
MOV     BP,SP
PUSH    DS
MOV     AX,LR_DATA
MOV     DS,AX

;-------------------------------------------------
LES     SI,DWORD PTR [BP+18]            ;
FLD     ES:DWORD PTR [SI]    ; LOAD PROFILE SPEED

FIMUL   INT1000                         ; COMPENSATE FOR ms

LES     SI,DWORD PTR [BP+14]            ;
FILD    ES:WORD PTR [SI]     ; LOAD CLOCK COUNT ms
FDIV                                    ; AND DIV

LES     SI,DWORD PTR [BP+10]            ;
FSTP    ES:DWORD PTR [SI]    ; STORE IN ACTUAL FEED RATE

;-------------------------------------------------
POP     DS
POP     SI
POP     ES
POP     BP
RET     12

CALC_VF_ACT ENDP

;-------------------------------------------------
;       FILTER SIZING DEVICE READING A LA
;
;       OLD = OLD + K*(NEW-OLD)
;
;-------------------------------------------------
FILTER                  PROC FAR

PUSH    BP
PUSH    ES
PUSH    SI
MOV     BP,SP
PUSH    DS
MOV     AX,LR_DATA
MOV     DS,AX

;-------------------------------------------------
LES     SI,DWORD PTR [BP+14]            ;
FLD     ES:QWORD PTR [SI]    ; LOAD OLD READING
LES     SI,DWORD PTR [BP+18]            ;
FLD     ES:QWORD PTR [SI]    ; LOAD NEW
FSUB    ST,ST(1)                ; NEW - OLD LEAVING OLD ON
STACK

LES     SI,DWORD PTR [BP+10]            ;
FLD     ES:QWORD PTR [SI]    ; LOAD RELAXATION
FMUL                                    ; AND MULT
FIDIV   INT100                          ; CONVERT FROM
FADD                                    ; ADD TO STACK TOP

LES     SI,DWORD PTR [BP+14]            ;
FSTP    ES:QWORD PTR [SI]    ; SAVE AS OLD

;-------------------------------------------------
POP     DS
POP     SI
POP     ES
POP     BP
RET     12

FILTER ENDP

;-------------------------------------------------
;       CALCULATE EFFECTIVE DIAMETER DE
;
;       DE = DS*DW(DS+DW)
;
;-------------------------------------------------
CALC_DE PROC FAR

PUSH    BP
PUSH    ES
PUSH    SI
MOV     BP,SP
PUSH    DS
MOV     AX,LR_DATA
MOV     DS,AX

;-------------------------------------------------
LES     SI,DWORD PTR [BP+14]            ;
FLD     ES:QWORD PTR [SI]    ; LOAD DS
LES     SI,DWORD PTR [BP+18]            ;
FLD     ES:QWORD PTR [SI]    ; LOAD DW
FMUL                                    ; MULT

LES     SI,DWORD PTR [BP+14]            ;
FLD     ES:QWORD PTR [SI]    ; LOAD DS
LES     SI,DWORD PTR [BP+18]            ;
FLD     ES:QWORD PTR [SI]    ; LOAD DW
FADD                                    ; ADD

FDIV                                    ; DIVIDE THEM

LES     SI,DWORD PTR [BP+10]            ;
FSTP    ES:QWORD PTR [SI]    ; AND STORE
```

```
;-------------------------------------------------
POP     DS
POP     SI
POP     ES
POP     BP
RET     12

CALC_DE ENDP

;-------------------------------------------------
;       CALCULATE VF FROM THE EQUATION
;
;       VF = 2*VM*VS/(PI*DW*LD*LD)
;
;-------------------------------------------------
CALC_VF PROC FAR

PUSH    BP
PUSH    ES
PUSH    SI
MOV     BP,SP
PUSH    DS
MOV     AX,LR_DATA
MOV     DS,AX

;-------------------------------------------------
LES     SI,DWORD PTR [BP+26]            ;
FLD     ES:QWORD PTR [SI]    ; LOAD CHIP VOLUME (VM)
LES     SI,DWORD PTR [BP+22]            ;
FLD     ES:QWORD PTR [SI]    ; LOAD WHEEL SPEED (VS)
FMUL                                    ; AND MULTIPLY

FADD    ST,ST                           ; * BY 2
FLDPI                                   ; LOAD PI
FDIV                                    ; AND DIVIDE

LES     SI,DWORD PTR [BP+18]            ;
FLD     ES:QWORD PTR [SI]    ; LOAD COMPONENT DIAM
FDIV                                    ; AND DIV

LES     SI,DWORD PTR [BP+14]            ;
FLD     ES:QWORD PTR [SI]    ; LOAD GRIT SIZE (LD)
FMUL    ST,ST                           ; SQUARE IT
FDIV                                    ; AND DIVIDE

FIDIV   INT1000                         ; SCALE

LES     SI,DWORD PTR [BP+10]            ;
FSTP    ES:QWORD PTR [SI]    ; AND SAVE

;-------------------------------------------------
POP     DS
POP     SI
POP     ES
POP     BP
RET     20

CALC_VF ENDP

;-------------------------------------------------
;       CALCULATE VF REAL FROM THE EQUATION
;
;       VF REAL = VF FEED * (1 - EXP(-T1/TOR))
;
;-------------------------------------------------
```

```
CALC_VF_REAL    PROC FAR

PUSH    BP
PUSH    ES
PUSH    SI
MOV     BP,SP
PUSH    DS
MOV     AX,LR_DATA
MOV     DS,AX

;-------------------------------------------------
LES     SI,DWORD PTR [BP+18]            ;
FLD     ES:QWORD PTR [SI]    ; LOAD T_INFEED
FCHS

LES     SI,DWORD PTR [BP+14]            ;
FLD     ES:QWORD PTR [SI]    ; LOAD TOR
FDIV

FSTP    TEMP_LR2             ; STORE IN LR2
FLD     EXP
FSTP    TEMP_LR1
CALL    LR_POWER_LR                     ; EXP (-T1/TOR)

FLD1
FLD     TEMP_LR3
FSUB                                    ;1-EXP(-T1/TOR)

LES     SI,DWORD PTR [BP+22]            ;
FLD     ES:QWORD PTR [SI]    ; LOAD VF_FEED
FMUL

LES     SI,DWORD PTR [BP+10]            ;
FSTP    ES:QWORD PTR [SI]    ; SAVE VF_REAL

;-------------------------------------------------
POP     DS
POP     SI
POP     ES
POP     BP
RET     16

CALC_VF_REAL ENDP

;-------------------------------------------------
;       CALCULATE VW A LA
;
;       VW = DE*VS/(LD*AR)
;
;-------------------------------------------------
CALC_VW PROC FAR

PUSH    BP
PUSH    ES
PUSH    SI
MOV     BP,SP
PUSH    DS
MOV     AX,LR_DATA
MOV     DS,AX

;-------------------------------------------------
LES     SI,DWORD PTR [BP+26]            ;
FLD     ES:QWORD PTR [SI]    ; LOAD EFFECTIVE DIAM (DE)
LES     SI,DWORD PTR [BP+22]            ;
```

```
FLD     ES:QWORD PTR [SI]    ; LOAD GDG WHEEL SPEED (VS)
FMUL                                    ; AND MULT

LES     SI,DWORD PTR [BP+18]            ;
FLD     ES:QWORD PTR [SI]    ; LOAD GRIT SIZE (LD)
FDIV                                    ; AND DIV

LES     SI,DWORD PTR [BP+14]            ;
FLD     ES:QWORD PTR [SI]    ; LOAD COMPONENT DIAM
FDIV                                    ; AND DIV

LES     SI,DWORD PTR [BP+10]            ;
FSTP    ES:QWORD PTR [SI]    ; STORE

;-------------------------------------------------
POP     DS
POP     SI
POP     ES
POP     BP
RET     20

CALC_VW ENDP

;-------------------------------------------------
;       CALCULATE MINIMUM CONTROL WHEEL SPEED
;
;       VW MIN = PI *DW * N (N=MIN CMPT REVS/S)
;
;-------------------------------------------------
CALC_VW_MIN     PROC FAR

PUSH    BP
PUSH    ES
PUSH    SI
MOV     BP,SP
PUSH    DS
MOV     AX,LR_DATA
MOV     DS,AX

;-------------------------------------------------
LES     SI,DWORD PTR [BP+18]            ;
FLD     ES:QWORD PTR [SI]    ; LOAD CMPT DIAM
LES     SI,DWORD PTR [BP+14]            ;
FLD     ES:QWORD PTR [SI]    ; LOAD MIN REVS/S
FMUL                                    ; MULT
FLDPI                                   ; LOAD PI
FMUL                                    ; MULTIPLY BY PI
FIDIV   INT1000                         ; SCALE TO M

LES     SI,DWORD PTR [BP+10]            ;
FSTP    ES:QWORD PTR [SI]    ; STORE VW MIN

;-------------------------------------------------
POP     DS
POP     SI
POP     ES
POP     BP
RET     12

CALC_VW_MIN ENDP

;-------------------------------------------------
;       CALCULATE  VARIOUS AXIS POSITIONS
;       BY SUBTRACTING DISTANCES, DIAMS, ETC
```

```
;           FROM THE BED LENGTH AND THEN NEGATING
;           THE RESULT.
;
;-------------------------------------------------
CALC_POS    PROC FAR

PUSH        BP
PUSH        ES
PUSH        SI
MOV         BP,SP
PUSH        DS
MOV         AX,LR_DATA
MOV         DS,AX

;-------------------------------------------------
LES         SI,DWORD PTR [BP+22]          ;
FLD         ES:QWORD PTR [SI]     ; LOAD BED LENGTH
LES         SI,DWORD PTR [BP+18]          ;
FLD         ES:QWORD PTR [SI]     ; LOAD CMPT DIAM
FSUB                                      ; SUB THEM

LES         SI,DWORD PTR [BP+14]          ;
FLD         ES:QWORD PTR [SI]     ; LOAD DISTANCE
FSUB                                      ; SUB AGAIN
FCHS                                      ; NEGATE

LES         SI,DWORD PTR [BP+10]          ;
FSTP        ES:QWORD PTR [SI]     ; AND SAVE

;-------------------------------------------------
POP         DS
POP         SI
POP         ES
POP         BP
RET         16

CALC_POS ENDP


;-------------------------------------------------
;           CALCULATE WHEEL SPEEDS IN RPM EVERY
;           SECOND FROM ENCODER COUNTS.
;
;           RPM = (NEW_COUNT-OLD_COUNT)*60/ENC_RES
;
;-------------------------------------------------
CALC_SPEED          PROC FAR

PUSH        BP
PUSH        ES
PUSH        SI
MOV         BP,SP
PUSH        DS
MOV         AX,LR_DATA
MOV         DS,AX

;-------------------------------------------------
LES         SI,DWORD PTR [BP+22]          ;
FILD        ES:DWORD PTR [SI]     ; LOAD NEW COUNT
LES         SI,DWORD PTR [BP+18]          ;
FILD        ES:DWORD PTR [SI]     ; LOAD OLD COUNT
FSUB                                      ; SUBTRACT

FIMUL       SECS                          ; CONVERT TO MINS

LES         SI,DWORD PTR [BP+14]          ;

FLD         ES:DWORD PTR [SI]     ; LOAD ENCODER RES
FDIV                                      ; DIVIDE

LES         SI,DWORD PTR [BP+10]          ;
FSTP        ES:DWORD PTR [SI]     ; AND STORE

;-------------------------------------------------
POP         DS
POP         SI
POP         ES
POP         BP
RET         16

CALC_SPEED ENDP


;-------------------------------------------------
;           PROC TO ADD TWO LONG REALS
;
;           LR3 = LR1 + LR2
;
;-------------------------------------------------
LR_ADD      PROC FAR

PUSH        BP
PUSH        ES
PUSH        SI
MOV         BP,SP

;-------------------------------------------------
LES         SI,DWORD PTR [BP+18]          ;
FLD         ES:QWORD PTR [SI]     ; LOAD LR1
LES         SI,DWORD PTR [BP+14]          ;
FLD         ES:QWORD PTR [SI]     ; LOAD LR2
FADD                                      ; ADD

LES         SI,DWORD PTR [BP+10]          ;
FSTP        ES:QWORD PTR [SI]     ; SAVE LR3

;-------------------------------------------------
POP         SI
POP         ES
POP         BP
RET         12

LR_ADD ENDP


;-------------------------------------------------
;           PROC TO SUBTRACT TWO LONG REALS
;
;           L3 = L1 - L2
;
;-------------------------------------------------
LR_SUB      PROC FAR

PUSH        BP
PUSH        ES
PUSH        SI
MOV         BP,SP

;-------------------------------------------------
LES         SI,DWORD PTR [BP+18]          ;
FLD         ES:QWORD PTR [SI]     ; LOAD LR1
LES         SI,DWORD PTR [BP+14]          ;
FLD         ES:QWORD PTR [SI]     ; LOAD LR2

FSUB                                      ; SUB

LES         SI,DWORD PTR [BP+10]          ;
FSTP        ES:QWORD PTR [SI]     ; SAVE LR3

;-------------------------------------------------
POP         SI
POP         ES
POP         BP
RET         12

LR_SUB ENDP


;-------------------------------------------------
;           PROC TO MULTIPLY TWO LONG REALS
;
;           LR3 = LR1 * LR2
;
;-------------------------------------------------
LR_MUL      PROC FAR

PUSH        BP
PUSH        ES
PUSH        SI
MOV         BP,SP

;-------------------------------------------------
LES         SI,DWORD PTR [BP+18]          ;
FLD         ES:QWORD PTR [SI]     ; LOAD LR1
LES         SI,DWORD PTR [BP+14]          ;
FLD         ES:QWORD PTR [SI]     ; LOAD LR2
FMUL                                      ; MUL

LES         SI,DWORD PTR [BP+10]          ;
FSTP        ES:QWORD PTR [SI]     ; SAVE LR3

;-------------------------------------------------
POP         SI
POP         ES
POP         BP
RET         12

LR_MUL ENDP


;-------------------------------------------------
;           PROC TO MULTIPLY LONG REAL BY A REAL
;
;           LR2 = LR1 * RE1
;
;-------------------------------------------------
LR_MUL_RE   PROC FAR

PUSH        BP
PUSH        ES
PUSH        SI
MOV         BP,SP

;-------------------------------------------------
LES         SI,DWORD PTR [BP+18]          ;
FLD         ES:QWORD PTR [SI]     ; LOAD LR1
LES         SI,DWORD PTR [BP+14]          ;
FLD         ES:DWORD PTR [SI]     ; LOAD RE1
FMUL                                      ; MULT
```

```
LES     SI,DWORD PTR [BP+10]          ;
FSTP    ES:QWORD PTR [SI]    ; SAVE LR2
;-------------------------------------------------------
POP     SI
POP     ES
POP     BP
RET     12

LR_MUL_RE ENDP

;-------------------------------------------------------
;       PROC TO DIVIDE TWO REALS
;
;       LR3 = LR1/LR2
;
;-------------------------------------------------------
LR_DIV  PROC FAR

PUSH    BP
PUSH    ES
PUSH    SI
MOV     BP,SP

;-------------------------------------------------------
LES     SI,DWORD PTR [BP+18]          ;
FLD     ES:QWORD PTR [SI]    ; LOAD LR1
LES     SI,DWORD PTR [BP+14]          ;
FLD     ES:QWORD PTR [SI]    ; LOAD LR2
FDIV                                  ; DIVIDE

LES     SI,DWORD PTR [BP+10]          ;
FSTP    ES:QWORD PTR [SI]    ; AND STORE LR3

;-------------------------------------------------------
POP     SI
POP     ES
POP     BP
RET     12

LR_DIV ENDP

;-------------------------------------------------------
;       CONVERT LONG REAL TO REAL
;
;-------------------------------------------------------
LR2R    PROC FAR

PUSH    BP
PUSH    ES
PUSH    SI
MOV     BP,SP

;-------------------------------------------------------
LES     SI,DWORD PTR [BP+14]
FLD     ES:QWORD PTR [SI]

LES     SI,DWORD PTR [BP+10]
FSTP    ES:DWORD PTR [SI]

;-------------------------------------------------------
POP     SI
POP     ES
POP     BP
```

```
RET     8

LR2R ENDP

;-------------------------------------------------------
;       CONVERT REAL TO LONG REAL
;
;-------------------------------------------------------
R2LR    PROC FAR

PUSH    BP
PUSH    ES
PUSH    SI
MOV     BP,SP

;-------------------------------------------------------
LES     SI,DWORD PTR [BP+14]
FLD     ES:DWORD PTR [SI]

LES     SI,DWORD PTR [BP+10]
FSTP    ES:QWORD PTR [SI]

;-------------------------------------------------------
POP     SI
POP     ES
POP     BP
RET     8

R2LR ENDP

;-------------------------------------------------------
;       CONVERT REAL TO LONG REAL
;
;-------------------------------------------------------
R2LR3   PROC FAR

PUSH    BP
PUSH    ES
PUSH    SI
MOV     BP,SP
;-------------------------------------------------------
;       SET ROUNDING CONTROL TO NEAREST

FSTCW   CONTROL_WORD
FWAIT
AND     CONTROL_WORD,0F3FFH
FLDCW   CONTROL_WORD
FWAIT

;-------------------------------------------------------
LES     SI,DWORD PTR [BP+14]
FLD     ES:DWORD PTR [SI]

FIMUL   INT1000
FRNDINT
FIDIV   INT1000

LES     SI,DWORD PTR [BP+10]
FSTP    ES:QWORD PTR [SI]

;-------------------------------------------------------
POP     SI
POP     ES
POP     BP
RET     8
```

```
R2LR3 ENDP

;-------------------------------------------------------
;       CONVERT REAL TO WORD REAL
;
;-------------------------------------------------------
R2W     PROC FAR

PUSH    BP
PUSH    ES
PUSH    SI
MOV     BP,SP

;-------------------------------------------------------
LES     SI,DWORD PTR [BP+14]
FLD     ES:DWORD PTR [SI]

;-------------------------------------------------------
;       SET ROUNDING CONTROL TO NEAREST

FSTCW   CONTROL_WORD
FWAIT

AND     CONTROL_WORD,0F3FFH

FLDCW   CONTROL_WORD
FWAIT

LES     SI,DWORD PTR [BP+10]
FISTP   ES:WORD PTR [SI]

;-------------------------------------------------------
POP     SI
POP     ES
POP     BP
RET     8

R2W ENDP

;-------------------------------------------------------
;       CONVERT LONG WORD TO LONG REAL
;
;-------------------------------------------------------
DW2LR   PROC FAR

PUSH    BP
PUSH    ES
PUSH    SI
MOV     BP,SP

;-------------------------------------------------------
LES     SI,DWORD PTR [BP+14]
FILD    ES:DWORD PTR [SI]

LES     SI,DWORD PTR [BP+10]
FSTP    ES:QWORD PTR [SI]

;-------------------------------------------------------
POP     SI
POP     ES
POP     BP
RET     8
```

```
DW2LR ENDP

;-----------------------------------------------
;       COMPARE LR1 WITH LR2 A LA
;
;       REAL RESULT = LR1 - LR2
;
;-----------------------------------------------
LR_CMP          PROC FAR

PUSH    BP
PUSH    ES
PUSH    SI
MOV     BP,SP

;-----------------------------------------------
LES     SI,DWORD PTR [BP+18]
FLD     ES:QWORD PTR [SI]
LES     SI,DWORD PTR [BP+14]
FLD     ES:QWORD PTR [SI]
FSUB

LES     SI,DWORD PTR [BP+10]
FSTP    ES:DWORD PTR [SI]

;-----------------------------------------------
POP     SI
POP     ES
POP     BP
RET     12

LR_CMP ENDP

;-----------------------------------------------
;       CALCULATE EXPONENTIAL OF A LONG REAL
;
;       LR1 = EXP(LR1)
;
;-----------------------------------------------
LR_EXP          PROC FAR

PUSH    BP
PUSH    ES
PUSH    SI
MOV     BP,SP
PUSH    DS
MOV     AX,LR_DATA
MOV     DS,AX

;-----------------------------------------------
LES     SI,DWORD PTR [BP+10]            ;
FLD     ES:QWORD PTR [SI]    ; LOAD LR1
FABS                                    ; MAKE SURE +VE
FLDL2E                                  ; LOAD LOG2(e)
FMUL                                    ; CALC X*LOG2(e)
FADD    ST,ST                           ; DOUBLE UP

;-----------------------------------------------
;       SET ROUNDING CONTROL TO ROUND DOWN

FSTCW   CONTROL_WORD
FWAIT
AND     CONTROL_WORD,0F3FFH
OR      CONTROL_WORD,0400H
FLDCW   CONTROL_WORD
FWAIT

;-----------------------------------------------
FIST    COUNTER                         ; STORE INTEGER
FILD    COUNTER                         ; AND LOAD
FSUB                                    ; SUBTRACT
FIDIV   INT2                            ; HALVE TO GIVE
RESULT <=0.5 FOR 2**POWER

F2XM1                                   ; 2**ST
FLD1                                    ;
FADD                                    ; ADD ONE BECAUSE YOU
HAVE TO

;-----------------------------------------------
CMP     COUNTER,0000H                   ;
JE      END_EXP                         ; JMP TO END IF
COUNTER = 0

;-----------------------------------------------
MOV     CX,COUNTER                      ;
FLD1                                    ;

ROOTLOOP:                       ; GET THE REQUIRED AMOUNT OF
ROOT2'S
        FMUL    ROOT2                   ;
        LOOP    ROOTLOOP    ;

FMUL                                    ; AND MULT BY ST
;-----------------------------------------------
END_EXP:

LES     SI,DWORD PTR [BP+10]            ;
FLD     ES:QWORD PTR [SI]    ; LOAD LR1
FLDZ                                    ; LOAD ZERO
FCOMP                                   ; COMPARE THEM
FSTP    ST                              ;

FSTSW   STATUS_WORD                     ;CHECK FLAGS
FWAIT
MOV     AH,BYTE PTR STATUS_WORD+1
SAHF
JBE     MONEG                           ; IF +VE THEN CARRY
ON

;-----------------------------------------------
FLD1                                    ; OTHERWISE INVERT
FDIVR                                   ;

;-----------------------------------------------
MONEG:

FSTP    ES:QWORD PTR [SI]    ; SAVE RESULT IN LR1

;-----------------------------------------------
POP     DS
POP     SI
POP     ES
POP     BP
RET     4

LR_EXP  ENDP

;-----------------------------------------------
;       CALCULATE A LONG REAL TO THE POWER OF A REAL
;
;       LR2 = LR1 ** RE1
;
;-----------------------------------------------
LR_POWER_RE         PROC FAR

PUSH    BP
PUSH    ES
PUSH    SI
MOV     BP,SP
PUSH    DS
MOV     AX,LR_DATA
MOV     DS,AX

;-----------------------------------------------
FLD1                                    ;
LES     SI,DWORD PTR [BP+18]            ;
FLD     ES:QWORD PTR [SI]    ; LOAD LR1
FYL2X                                   ; CALCULATE
1*LOG2(LR1)

LES     SI,DWORD PTR [BP+14]            ;
FLD     ES:DWORD PTR [SI]    ; LOAD REAL
FMUL                                    ; CALC RE1*LOG2(LR1)
FST     TEMP_LR2                ; AND STORE
FABS                                    ; MAKE +VE
FADD    ST,ST                           ; DOUBLE

;-----------------------------------------------
;       SET ROUNDING CONTROL TO ROUND DOWN

FSTCW   CONTROL_WORD
FWAIT
AND     CONTROL_WORD,0F3FFH
OR      CONTROL_WORD,0400H
FLDCW   CONTROL_WORD
FWAIT

;-----------------------------------------------
FIST    COUNTER                         ; STORE INTEGER
FILD    COUNTER                         ; LOAD AGAIN
FSUB                                    ; SUB
FIDIV   INT2                            ; HALVE REMAINDER
<=0.5 FOR 2**POWER

F2XM1                                   ; CALCULATE 2**ST
FLD1                                    ; LOAD 1
FADD                                    ; ADD 1

;-----------------------------------------------
CMP     COUNTER,0000H                   ;
JE      END_POWER               ; JMP TO END IF COUNTER = 0

;-----------------------------------------------
;MOV     CX,COUNTER                      ;
;FLD1                                    ;
;
;ROOTLOOP2:                              ; GET THE RIGHT
NUMBER OF ROOT2'S
;       FMUL    ROOT2                   ;
;       LOOP    ROOTLOOP2 ;
;FMUL                                    ; AND MUL BY ST
;
;-----------------------------------------------
```

```
MOV         CX,COUNTER                  ;GET COUNTER
AND         CX,01H                      ;CHECK BIT 0
CMP         CX,01H                      ;IF ODD WORD
JE          LDROOT1                     ;GOTO ROOT2 LOAD
;-----------------------------------------------------
                                        ;IF EVEN LOAD ONE
FLD1
JMP         SHIFTCNT1
;-----------------------------------------------------
LDROOT1:
FLD         ROOT2                       ;IF ODD LOAD ROOT2
;-----------------------------------------------------
SHIFTCNT1:

SHR         COUNTER,1           ;DIVIDE COUNTER BY 2

MOV         CX,COUNTER
CMP         CX,14                       ;TEST IF OVERRUN ON
SAL
JBE         SMALL_POWER1                ;IF NOT THEN OK...
;-----------------------------------------------------
SUB         CX,15
MOV         COUNTER,0001H
SAL         COUNTER,CL                  ;2 TO POWER COUNTER
FIMUL       COUNTER                     ;AN * BY ST
FIMUL       BIT15                       ;*BY 32K
FMUL                                    ;*BY PREVIOUS ST
JMP         END_POWER
;-----------------------------------------------------
SMALL_POWER1:

MOV         COUNTER,0001H
SAL         COUNTER,CL                  ;2 TO POWER COUNTER
FIMUL       COUNTER                     ;AN * BY ST
FMUL                                    ;*BY PREVIOUS ST
;-----------------------------------------------------
END_POWER:

FLD         TEMP_LR2            ; LOAD POWER
FLDZ                                    ; LOAD ZERO
FCOMPP                                  ; COMPARE THE TWO

FSTSW       STATUS_WORD                 ;CHECK FLAGS
FWAIT
MOV         AH,BYTE PTR STATUS_WORD+1
SAHF
JBE         NO_NEG_PWR                  ; IF +VE THE OK
;-----------------------------------------------------
FLD1                                    ; OTHERWISE INVERT
FDIVR
;-----------------------------------------------------
NO_NEG_PWR:

LES         SI,DWORD PTR [BP+10]        ;
FSTP        ES:QWORD PTR [SI]   ; SAVE RESULT
;-----------------------------------------------------
POP         DS
POP         SI
POP         ES
POP         BP
RET         12

LR_POWER_RE  ENDP

;-----------------------------------------------------
;           LOCAL       LR_POWER_LR CALCULATION
;
;           LR3 = LR1 ** LR2
;
;-----------------------------------------------------
LR_POWER_LR             PROC NEAR

;-----------------------------------------------------
FLD1                                    ;
FLD         TEMP_LR1            ;
FYL2X                                   ; CALC LOG2(LR1)

FLD         TEMP_LR2            ; LOAD POWER
FMUL                                    ; GET LR2*LOG2(LR1)
FST         TEMP_LR2
FABS                                    ; MAKE +VE
FADD        ST,ST                       ; * BY 2
;-----------------------------------------------------
;           SET ROUNDING CONTROL TO ROUND DOWN

FSTCW       CONTROL_WORD
FWAIT

AND         CONTROL_WORD,0F3FFH
OR          CONTROL_WORD,0400H

FLDCW       CONTROL_WORD
FWAIT

;-----------------------------------------------------
FIST        COUNTER                     ; STORE INT COUNTER
FILD        COUNTER                     ; LOAD AGAIN
FSUB                                    ; SUB
FIDIV       INT2                        ; DIV BY 2 TO LEAVE
POWER <=0.5

F2XM1                                   ;
FLD1                                    ;
FADD                                    ; GET 2**ST
;-----------------------------------------------------
CMP         COUNTER,0000H               ;
JE          END_POWER2                  ; IF COUNTER = 0 THEN
END
;-----------------------------------------------------
;MOV        CX,COUNTER                  ; OTHERWISE
;FLD1                                   ;
;                                       ; GET SOME ROOT2'S
;ROOTLOOP3:                             ;
;           FMUL        ROOT2           ;
;           LOOP        ROOTLOOP3 ;     ; AND MULT BY ST
;FMUL                                   ;
;-----------------------------------------------------
MOV         CX,COUNTER                  ;GET COUNTER
AND         CX,01H                      ;CHECK BIT 0
CMP         CX,01H                      ;IF ODD WORD
JE          LDROOT2                     ;GOTO ROOT2 LOAD
;-----------------------------------------------------
FLD1                                    ;IF EVEN LOAD ONE
JMP         SHIFTCNT
;-----------------------------------------------------
LDROOT2:

FLD         ROOT2                       ;IF ODD LOAD ROOT2
;-----------------------------------------------------
SHIFTCNT:

SHR         COUNTER,1           ;DIVIDE COUNTER BY 2
MOV         CX,COUNTER

SHIFT_CNT_LOOP:

CMP         CX,14
JBE         SMALL_POWER

SUB         CX,15
FIMUL       BIT15
JMP         SHIFT_CNT_LOOP

;MOV        CX,COUNTER
;CMP        CX,14                       ;TEST IF OVERRUN ON
SAL
;JBE        SMALL_POWER                 ;IF NOT THEN OK...
;-----------------------------------------------------
;SUB        CX,15
;MOV        COUNTER,0001H
;SAL        COUNTER,CL                  ;2 TO POWER COUNTER
;FIMUL      COUNTER                     ;AN * BY ST
;FIMUL      BIT15                       ;*BY 32K
;FMUL                                   ;*BY PREVIOUS ST
;JMP        END_POWER2
;-----------------------------------------------------
SMALL_POWER:

MOV         COUNTER,0001H
SAL         COUNTER,CL                  ;2 TO POWER COUNTER
FIMUL       COUNTER                     ;AN * BY ST
FMUL                                    ;*BY PREVIOUS ST
;-----------------------------------------------------
END_POWER2:
;-----------------------------------------------------
FLD         TEMP_LR2            ; LOAD POWER
FLDZ                                    ; LOAD ZERO
FCOMPP                                  ; COMPARE

FSTSW       STATUS_WORD                 ;CHECK FLAGS
FWAIT
MOV         AH,BYTE PTR STATUS_WORD+1
SAHF
JBE         NO_NEG_PWR2                 ; IF NOT NEGATIVE
THEN OK
```

```
FLD1                                    ; OTHERWISE INVERT
FDIVR                                   ;

;-------------------------------------------------
NO_NEG_PWR2:

FSTP    TEMP_LR3            ; STORE IN LR3

;-------------------------------------------------
RET

LR_POWER_LR   ENDP

;-------------------------------------------------
;       ROUND OFF TO 4 SIG FIGS

SIG_FIG4  PROC FAR

PUSH    BP
PUSH    ES
PUSH    SI
MOV     BP,SP

;-------------------------------------------------
;       SET ROUNDING CONTROL TO NEAREST

FSTCW   CONTROL_WORD
FWAIT
AND     CONTROL_WORD,0F3FFH
FLDCW   CONTROL_WORD
FWAIT

LES     SI,DWORD PTR [BP+10]
FLD     ES:QWORD PTR [SI]

FIMUL   INT10000
FRNDINT
FIDIV   INT10000
FSTP    ES:QWORD PTR [SI]

;-------------------------------------------------
POP     SI
POP     ES
POP     BP
RET     4

SIG_FIG4 ENDP

;-------------------------------------------------
LR_MATHS ENDS

END
```

## Appendix 3.4 Part-program routines

This appendix includes the part-program routines that were employed in the intelligent control system design. The part-program routines provided the operation sequence for the intelligent control system and provided the link between Siprom, CSI and 8600 operating system software.

# PECKER

## Master pecking cyle sequence module

```
;-------------------------
; Main part-program package
; for driving CSI/SIPROM/P-P
; routines in the right
; order (???)
;
; (C)Sean Kelly 1988.
;
; Data file initialisation test.
; Modified 21/11/1988
;
;-------------------------
; CANCEL ALL MESSAGES,FLAGS ETC
;
SK564=0
SK601=0
SK602=0
SK603=0
SK606=0
SK608=0
SK611=0
SK612=0
SK613=0
SK614=0
SK615=0
SK616=255
SK620=0
SK621=0
SK622=0
SK623=0
SK624=0
SK625=0
SK626=0
SK627=0
SK628=0
;
;-------------------------
; ZERO COUNTERS
;
E119=0
E120=0
;
;-------------------------
; GRAPHICS OFF
;
SK614=255
;
; SIZING FILTER ON
;
SK609=1
;
; CSI ENABLED
;
SK612=1
;
;-------------------------
; CALL DISPLAY NO. 1
;
"MAINST"
SK611=1
SK601=1
;
"DS1FIN"
(DLY,0.1)
(BNE,SK611,255,DS1FIN)
(BNE,SK606,255,FILERR)
;
;-------------------------
; TEST FOR CSI FILE INIT
;
(BEQ,SK636,0,OPT1)
(CLS,FINIT)
;
;-------------------------
"CONT"
SYVAR.3CH="Y  "
(INP,"FILES INIT. CONTINUE ? (Y/N) ",SYVAR.3CH)
(BEQ,SYVAR.3CH,"Y  ",OPT1)
(BEQ,SYVAR.3CH,"N  ",THEEND)
(BNC,CONT)
;
;-------------------------
;
"OPT1"
(DLY,0.1)
E10=0
SK620=0
SK621=0
;
; POWER LIMITER ON
;
SK616=1
;
;-------------------------
; SELECT OPTION:
;
(INP,"OPTION ? ",E10)
(BEQ,E10,1,GRIND)
(BEQ,E10,2,EDIT)
(BEQ,E10,3,SETBED)
(BEQ,E10,4,DRESS)
(BEQ,E10,5,ZERO)
(BEQ,E10,6,THEEND)
(BNC,OPT1)
;
;-------------------------
; FLY OFF TO I/O ROUTINE
; BUT TURN OFF FAST CSI
;
"EDIT"
SK612=0
(CLS,GKNIO2/MP2)
SK612=1
(BNC,MAINST)
;
;-------------------------
; FLYING JUMP TO SUBROUTINE
; TO SET BED LENGTH
;
"SETBED"
(CLS,SETBED/MP2)
(BEQ,SK624,8,THEEND)
(BNC,MAINST)
;
;-------------------------
; WHEEL DRESSING OPTION
;
"DRESS"
(CLS,DRESS/MP2)
(BEQ,SK624,8,THEEND)
(BNC,MAINST)
;
;-------------------------
; ZERO BATCH COUNTER OPTION
; LOAD COUNTER FILE MCDAT1
;
"ZERO"
SK613=2
SK601=1
(DLY,0.1)
E30=0
SK613=4
SK601=1
(DLY,0.1)
(BNC,MAINST)
;
;-------------------------
; START GRINDING SEQUENCE
; BY CALCULATING VF,VW ETC..
; AND TEST FOR DRESS
;
"GRIND"
;
;-------------------------
; LOAD COUNTER FILE MCDAT1
;
SK613=2
SK601=1
(DLY,0.1)
;
; TEST FOR DRESS
;
(BLT,E31,E32,DRESOK)
(BNC,DRESS)
;
;-------------------------
; CALL CALC VF,VW
;
"DRESOK"
SK602=1
;
"VRFIN1"
(DLY,0.1)
(BNE,SK602,255,VRFIN1)
;
;-------------------------
; CALL DISPLAY NO. 2
;
SK602=2
SK601=1
;
"DS2FIN"
(DLY,0.1)
(BNE,SK611,255,DS2FIN)
```

```
;
;-------------------------
; RUN WHEELS UP TO SPEED?
;
(CLS,WHEELS/MP2)
(BEQ,SK624,0,THEEND)
;
;-------------------------
; TURN ON NO LOAD POWER CALIBRATE
;
SK603=2
;
;-------------------------
; CALL SUBROUTINE TO CALIBRATE
; PROBES
;
(CLS,CALSIZ)
;
;-------------------------
; ZERO COUNTER
;
E120=0
SK602=8
SK601=1
;
;-------------------------
; WAIT FOR PENDANT START TO GRIND
; PUT INTO HOLD
;
"CYCLE"
;
SK620=1
SK621=2
SK622=0
SK242=0
M10
;
;-------------------------
; ZERO KBD PECKER INTERRUPT
;
SK627=0
;
; CALIBRATE NO LOAD POWER
;
SK603=2
;
;-------------------------
; ENABLE LARGE SCREEN
;
;SK602=8
;SK601=1
;
;"DS3FIN"
;(DLY,0.1)
;(BNE,SK611,255,DS3FIN)
;
;-------------------------
; PUT INTO HOLD
;
SK240=1
;
"REPT6"
(BEQ,SK240,1,REPT6)
(BEQ,SK240,2,MAINST)
;
;-------------------------
; GRINDING SECTION
;
SK620=0
SK621=7
;
;-------------------------
; START TIMER
;
E80=TIM
;
;-------------------------
; SET PECK COUNTER
;
E119=1
;
;-------------------------
; INCREMENT GRINDING COUNTER
; AND TX BLK
;
E120=E120+1
;(CLS,TXBLK1)
;
; START PMAX AND USMAX SEARCH
; AND TURN OFF NO LOAD CAL
;
SK603=1
;
;-------------------------
; LOG INITIAL CMPT DIAM
;
SK242=1
(DLY,3)
E71=E57
;E72=E58
E72=E57
SK242=0
;
; TEST SENSIBLE SIZE / NO INTERRUPT
;
(BGT,E71,1,CYCLE)
(BGT,E72,1,CYCLE)
(BEQ,SK627,255,CYCLE)
;
;-------------------------
; E101=LARGEST OF PROBES
;
(BGE,E71,E72,BIGE1)
E101=E72
(BNC,CALOS1)
;
"BIGE1"
E101=E71
;
;-------------------------
; E70 = AMOUNT TO REMOVE
; E93 = SAVED VALUE
;
"CALOS1"
;
E93=(E71+E72)*0.5
E12=130+2*(E119-1)
E70=E93-E(E12)-E69
;
(BGT,E70,0,CALOS2)
E70=0
E101=E52-E53
;
;-------------------------
; CALCULATE OVERSHOOT ON E70
;
"CALOS2"
;
SK608=4
;
"OSFIN1"
(DLY,0.1)
(BEQ,SK637,255,CYCLE)
(BNE,SK608,255,OSFIN1)
;
;-------------------------
; CALL CALC EC*
;
SK602=3
;
"ECFIN1"
(DLY,0.1)
(BNE,SK602,255,ECFIN1)
;
;(CLS,TXBLK2)
;
;-------------------------
; PECKING CYCLE START
;
M11
G00 XE50
;
"PECK"
;
; POWER LIMITER ON
;
SK616=1
;
;-------------------------
; GOTO HSBP THEN TARGET POSITION
; CALCULATE FEEDRATE IN MM/MIN
;
SK623=5
;
E27=E26*60
;
; DWELL
;
TMR=E54
;
E102=E53+E100+E101
E84=E53-E91-E68+E69
M08 SE25
G1 G27 XE102 F425
G29 G4 G1 XE84 FE27
;
;-------------------------
; RAPID TRAVERSE TO HOME
;
G XE50 M09
;
SK623=0
;
;-------------------------
; STORE SIZING ERROR
;
SK242=1
(DLY,3)
```

```
E71=E57
;E72=E58
E72=E57
SK242=0
;
;-------------------------
; CALC NEW TOR
;
E94=(E71+E72)/2
E73=E93-E94
SK608=8
;
"TORFIN"
(DLY,0.1)
(BNE,SK608,255,TORFIN)
;
;-------------------------
; UPDATE DATA BASE TOR
;
E85=1.5*E61
(BLE,E62,E85,NOTNO1)
;
SK618=1
SK601=1
;
"TORMOD"
(DLY,0.1)
(BNE,SK618,255,TORMOD)
;
;-------------------------
; TEST FOR FINISH
;
"NOTNO1"
;
E119=E119+1
;
(BEQ,E119,5,NOPECK)
(BGT,E71,1,NOPECK)
(BGT,E72,1,NOPECK)
(BEQ,SK627,255,NOPECK)
(BEQ,SK628,255,NOPECK)
;
E70=(E71+E72)*0.5-E69
;
(BLE,E70,0.006,NOPECK)
;
;-------------------------
; E101=LARGEST OF PROBES
;
(BGE,E71,E72,BIGE2)
E101=E72
(BNC,CALOS3)
;
"BIGE2"
E101=E71
;
;-------------------------
; E70 = AMOUNT TO REMOVE
; E93 = SAVED VALUE
;
"CALOS3"
;
;-------------------------
E93=(E71+E72)*0.5
E12=130+2*(E119-1)
E70=E93-E(E12)-E69
;
;-------------------------
;
(BGT,E70,0,CALOS4)
E70=0
E101=E52-E53
;
;-------------------------
; CALCULATE OVERSHOOT ON E70
;
"CALOS4"
;
SK608=4
;
"OSFIN2"
(DLY,0.1)
(BEQ,SK637,255,CYCLE)
(BNE,SK608,255,OSFIN2)
;
;(CLS,TXBLK3)
(BNC,PECK)
;
;-------------------------
; IF FINISHED THEN SIZING
; DEVICE OFF, POWER LIMIT
; OFF, ETC
;
"NOPECK"
SK242=0
SK616=255
SK621=1
;
M10
;
;-------------------------
; STOP TIMER AND SEND DATA
;
E83=TIM-E80
;(CLS,TXBLK4)
;
;-------------------------
; UPDATE COUNTERS IF
; PMAX>P NO LOAD
;
E85=1.5*E61
(BLE,E62,E85,CYCLE)
;
; LOAD COUNTER FILE MCDAT1
;
SK613=2
SK601=1
(DLY,0.1)
;
E30=E30+1
E31=E31+1
;
; SAVE COUNTER FILE MCDAT1
;
SK613=4
SK601=1
(DLY,0.1)
;
;-------------------------
; TEST FOR DRESS
;
(BLT,E31,E32,NODRES)
(BNC,DRESS)
;
;-------------------------
; ADAPTIVE CONTROL
;
"NODRES"
;
; VF UPDATE
;
SK608=1
"VFUP"
(DLY,0.1)
(BNE,SK608,255,VFUP)
;
; VW UPDATE
;
;SK608=2
;"VWUP"
;(DLY,0.1)
;(BNE,SK608,0255,VWUP)
;(USS,#E56)
;
; CALC EC*
;
;SK602=3
;"ECFIN2"
;(DLY,0.1)
;(BNE,SK602,255,ECFIN2)
;
(BNC,CYCLE)
;
;-------------------------
"FILERR"
(DIS,"FILE HANDLING ERROR")
(DLY,1)
SK564=1
(BNC,THEEND)
;
;-------------------------
"OSERR"
(DIS,"OVERSHOOT CALC ERROR")
(DLY,1)
SK564=2
(BNC,THEEND)
;
;-------------------------
"THEEND"
;SK620=0
;SK621=0
;SK622=0
;SK623=0
M03 S0
(USS,s0)
(DLY,10)
M05 M10
M30
```

# FINIT

## CSI data base initialisation module

```
;-------------------------
; SMALL ROUTINE TO DETECT
; FILE INITIALISATIONS
;
E11=SK636
E10=MOD(E11,2)
(BNE,E10,1,FINIT2)
"FQUES1"
SYVAR.3CH="Y  "
(INP,"MCDAT1 INIT. CONTINUE ? ",SYVAR.3CH)
(BEQ,SYVAR.3CH,"Y  ",FINIT2)
(BNC,FQUES1)
;
;-------------------------
"FINIT2"
E11=-0.5+E11/2
E10=MOD(E11,2)
(BNE,E10,1,FINIT3)
"FQUES2"
SYVAR.3CH="Y  "
(INP,"MCDAT2 INIT. CONTINUE ? ",SYVAR.3CH)
(BEQ,SYVAR.3CH,"Y  ",FINIT3)
(BNC,FQUES2)
;
;-------------------------
"FINIT3"
E11=-0.5+E11/2
E10=MOD(E11,2)
(BNE,E10,1,FINIT4)
"FQUES3"
SYVAR.3CH="Y  "
(INP,"MCDAT3 INIT. CONTINUE ? ",SYVAR.3CH)
(BEQ,SYVAR.3CH,"Y  ",FINIT4)
(BNC,FQUES3)
;
;-------------------------
"FINIT4"
E11=-0.5+E11/2
E10=MOD(E11,2)
(BNE,E10,1,FINIT5)
"FQUES4"
SYVAR.3CH="Y  "
(INP,"PARTAB INIT. CONTINUE ? ",SYVAR.3CH)
(BEQ,SYVAR.3CH,"Y  ",FINIT5)
(BNC,FQUES4)
;
;-------------------------
"FINIT5"
E11=-0.5+E11/2
E10=MOD(E11,2)
(BNE,E10,1,FINIT6)
"FQUES5"
SYVAR.3CH="Y  "
(INP,"MATTAB INIT. CONTINUE ? ",SYVAR.3CH)
(BEQ,SYVAR.3CH,"Y  ",FINIT6)
(BNC,FQUES5)
;
;-------------------------
"FINIT6"
E11=-0.5+E11/2
E10=MOD(E11,2)
(BNE,E10,1,FINIT7)
"FQUES6"
SYVAR.3CH="Y  "
(INP,"WHETAB INIT. CONTINUE ? ",SYVAR.3CH)
(BEQ,SYVAR.3CH,"Y  ",FINIT7)
(BNC,FQUES6)
;
;-------------------------
"FINIT7"
E11=-0.5+E11/2
E10=MOD(E11,2)
(BNE,E10,1,FINIT8)
"FQUES7"
SYVAR.3CH="Y  "
(INP,"PASTAB INIT. CONTINUE ? ",SYVAR.3CH)
(BEQ,SYVAR.3CH,"Y  ",FINIT8)
(BNC,FQUES6)
;
;-------------------------
"FINIT8"
E11=-0.5+E11/2
E10=MOD(E11,2)
(BNE,E10,1,FINEND)
"FQUES8"
SYVAR.3CH="Y  "
(INP,"CYCTAB INIT. CONTINUE ? ",SYVAR.3CH)
(BEQ,SYVAR.3CH,"Y  ",FINEND)
(BNC,FQUES8)
;
"FINEND"
```

# GKNIO2

## Data base edit module

```
;-----------------------------------------------------
;         ROUTINE TO HANDLE I/O TO
;         CSI DATA BASE
;
;         V1.1 Sean Kelly 1988.
;         V1.1a      Lock operator out of
;                    MCDAT2.
;-----------------------------------------------------
(SCR,ON)
PASS0.10CH=PASS10.10CH
(INP,"Please enter user code : ",PASS0.10CH)
;
"OPTION"
(BEQ,PASS0.10CH,PASS10.10CH,LEVEL1)
(BEQ,PASS0.10CH,PASS20.10CH,LEVEL2)
(BEQ,PASS0.10CH,PASS30.10CH,LEVEL3)
(BNC,NOPRI)
;
;------------------------
; LOW PRIORITY EDITING
;
"LEVEL1"
(SCR,ON)
(DEF,1,4,15,"0. Return to main menu.")
(DEF,2,6,15,"1. Select component.")
(DEF,3,8,15,"2. Edit machine constants.")
(DEF,4,10,15,"3. Edit cycle allowances.")
(OUT)
;
"OPT1"
E10=0
(INP,"Option ? ",E10)
(BEQ,E10,0,ENDPRG)
(BEQ,E10,1,CMPSEL)
(BEQ,E10,2,MCED)
(BEQ,E10,3,CYCED)
(BNC,OPT1)
;
;------------------------
; MEDIUM PRIORITY EDITING
;
"LEVEL2"
(SCR,ON)
(DEF,1,4,15,"0. Return to main menu.")
(DEF,2,6,15,"1. Select component.")
(DEF,3,8,15,"2. Edit machine constants.")
(DEF,4,10,15,"3. Edit cycle allowances.")
(DEF,5,12,15,"4. Edit component.")
(DEF,6,14,15,"5. Select grinding wheel.")
(OUT)
;
"OPT2"
E10=0
(INP,"Option ? ",E10)
(BEQ,E10,0,ENDPRG)
(BEQ,E10,1,CMPSEL)
(BEQ,E10,2,MCED)
(BEQ,E10,3,CYCED)
(BEQ,E10,4,CMPED)
(BEQ,E10,5,WHESEL)
(BNC,OPT2)
;
;------------------------
; HIGH PRIORITY EDITING
;
"LEVEL3"
(SCR,ON)
(DEF,1,4,15,"0. Return to main menu.")
(DEF,2,6,15,"1. Select component.")
(DEF,3,8,15,"2. Edit machine constants.")
(DEF,4,10,15,"3. Edit cycle allowances.")
(DEF,5,12,15,"4. Edit component.")
(DEF,6,14,15,"5. Select grinding wheel.")
(DEF,7,16,15,"6. Edit grinding wheel.")
(DEF,8,18,15,"7. Edit material.")
(DEF,9,20,15,"8. Edit system over-rides.")
(OUT)
;
"OPT3"
E10=0
(INP,"Option ? ",E10)
(BEQ,E10,0,ENDPRG)
(BEQ,E10,1,CMPSEL)
(BEQ,E10,2,MCED)
(BEQ,E10,3,CYCED)
(BEQ,E10,4,CMPED)
(BEQ,E10,5,WHESEL)
(BEQ,E10,6,WHEED)
(BEQ,E10,7,MATED)
(BEQ,E10,8,ORED)
(BNC,OPT3)
;
;------------------------
; COMPONENT SELECT ROUTINE
;
"CMPSEL"
;
;------------------------
; CALL DISPLAY TABLE
;
(EPP,START1,END1)
;
;------------------------
"CMPINP"
E10=0
(INP,"Grind component No.? ",E10)
(BGT,E10,10,CMPINP)
(BLT,E10,0,CMPINP)
(BEQ,E10,0,OPTION)
;
;------------------------
; ALLOCATE COMPONENT
; -SAVE IN E12 BEFORE
;  LOADING
;
E12=E10
;
SK611=2
SK601=1
(DLY,0.1)
;
E10=E12
;
SK611=4
SK601=1
(DLY,0.2)
;
(BNC,OPTION)
;
;------------------------
; CYCLE EDIT ROUTINE
;
"CYCED"
;
SK610=64
SK601=1
;
(DLY,0.2)
;
(EPP,STAR11,END11)
;
"CYCINP"
E12=0
(INP,"Modify parameter ? ",E12)
(BEQ,E12,0,ENDCYC)
(BEQ,E12,1,CYC1)
(BEQ,E12,2,CYC2)
(BEQ,E12,3,CYC3)
(BEQ,E12,4,CYC4)
(BEQ,E12,5,CYC5)
(BEQ,E12,6,CYC6)
(BEQ,E12,7,CYC7)
(BEQ,E12,8,CYC8)
(BNC,CYCINP)
;
"CYC1"
(INP,"Allowance No. 1 (mm) ? ",E130)
(EPP,STAR11,END11)
(BNC,CYCINP)
"CYC2"
(INP,"Allowance No. 2 (mm) ? ",E132)
(EPP,STAR11,END11)
(BNC,CYCINP)
"CYC3"
(INP,"Allowance No. 3 (mm) ? ",E134)
(EPP,STAR11,END11)
(BNC,CYCINP)
"CYC4"
(INP,"Allowance No. 4 (mm) ? ",E136)
(EPP,STAR11,END11)
(BNC,CYCINP)
;
"CYC5"
(INP,"Time constant No. 1 (s) ? ",E131)
(EPP,STAR11,END11)
(BNC,CYCINP)
"CYC6"
(INP,"Time constant No. 2 (s) ? ",E133)
(EPP,STAR11,END11)
(BNC,CYCINP)
"CYC7"
(INP,"Time constant No. 3 (s) ? ",E135)
(EPP,STAR11,END11)
(BNC,CYCINP)
```

```
"CYC8"
(INP,"Time constant No. 4 (s) ? ",E137)
(EPP,START11,END11)
(BNC,CYCINP)
;
"ENDCYC"
SK610=128
SK601=1
(DLY,0.2)
;
(BNC,OPTION)
;
;-------------------------
; COMPONENT EDIT ROUTINE
;
"CMPED"
;
;-------------------------
; CALL DISPLAY TABLE
;
(EPP,START1,END1)
;
;-------------------------
"CEDINP"
E10=0
(INP,"Edit component No.? ",E10)
(BGT,E10,10,CEDINP)
(BLT,E10,0,CEDINP)
(BEQ,E10,0,OPTION)
;
E10=E10-1
;
;-------------------------
; CALL INPUT TABLE
;
(EPP,START2,END2)
(DLY,0.1)
SK615=1
SK601=1
;
;
E11=E10*6
(INP,"New component name (6 CH) ? ",SYVAR(E11).6CH)
(EPP,START2,END2)
"FDINP"
E11=30+E10
(INP,"Final diameter (mm) ? ",E(E11))
;
;-------------------------
; TEST COMPONENT DIMENSIONS
;
(BLT,E(E11),40,FDERR)
(BGT,E(E11),120,FDERR)
(EPP,START2,END2)
(BNC,SDINP)
;
"FDERR"
(DEF,15,1,1,"DIMENSIONS OUT OF RANGE",R)
(OUT)
(DLY,1)
(EPP,START2,END2)
(BNC,FDINP)
;
;-------------------------
"SDINP"
E11=40+E10
E12=30+E10
(INP,"Start diameter (mm) ? ",E(E11))
;
; TEST COMPONENT START DIAM > FINISH
;
(BLE,E(E11),E(E12),SDERR)
(BGT,E(E11),120,SDERR)
(EPP,START2,END2)
(BNC,IDINP)
;
;-------------------------
"SDERR"
(DEF,15,1,1,"START DIAM. < FINISH DIAM.",R)
(DEF,16,2,1,"OR OUT OF RANGE.          ",R)
(OUT)
(DLY,1)
(EPP,START2,END2)
(BNC,SDINP)
;
;-------------------------
"IDINP"
E11=50+E10
E12=30+E10
(INP,"Internal diameter (mm) ? ",E(E11))
;
; TEST COMPONENT INTERNAL DIAM < FINAL DIAM.
;
(BGE,E(E11),E(E12),IDERR)
(EPP,START2,END2)
(BNC,LINP)
;
"IDERR"
(DEF,15,1,1,"INTERNAL DIAM. > FINISH DIAM.",R)
(OUT)
(DLY,1)
(EPP,START2,END2)
(BNC,IDINP)
;
;-------------------------
"LINP"
E11=60+E10
(INP,"Length (mm) ? ",E(E11))
;
; TEST COMPONENT LENGTH
;
(BLT,E(E11),100,LERR)
(BGT,E(E11),300,LERR)
(EPP,START2,END2)
(BNC,CMPMOD)
;
"LERR"
(DEF,15,1,1,"DIMENSIONS OUT OF RANGE",R)
(OUT)
(DLY,1)
(EPP,START2,END2)
(BNC,LINP)
;
;-------------------------
; ALLOCATE MATERIAL
;
"CMPMOD"
;
;-------------------------
; LOAD UP MATERIALS SELECTION
;
(EPP,START4,END4)
;
;-------------------------
"MATSEL"
E12=0
(INP,"Material No.? ",E12)
(BGT,E12,10,MATSEL)
(BLT,E12,0,MATSEL)
(BEQ,E12,0,OPTION)
;
;
E11=E10*2+180
SYVAR(E11)=E12
;
SK610=2
SK601=1
;
"CMPWA2"
(DLY,0.1)
(BNE,SK610,255,CMPWA2)
;
(BNE,SK606,255,ERROR)
;
(BNC,CMPSEL)
;
;-------------------------
; WHEEL SELECT ROUTINE
;
"WHESEL"
;
;-------------------------
; CALL DISPLAY TABLE
;
(EPP,START3,END3)
;
;-------------------------
"WHEINP"
E10=0
(INP,"Select wheel No.? ",E10)
(BGT,E10,3,WHEINP)
(BLT,E10,0,WHEINP)
(BEQ,E10,0,OPTION)
;
;-------------------------
; ALLOCATE WHEEL
; -SAVE IN E12 BEFORE
;  LOADING
;
E12=E10
;
SK611=2
SK601=1
(DLY,0.2)
;
E11=E12
;
SK611=4
SK601=1
(DLY,0.2)
;
(BNC,OPTION)
;
;-------------------------
; WHEEL EDIT ROUTINE
;
"WHEED"
```

```
;
;-------------------------
; CALL WHEEL DISPLAY TABLE
;
(EPP,START3,END3)
;
;-------------------------
"WEDINP"
E10=0
(INP,"Edit wheel No.? ",E10)
(BGT,E10,3,WEDINP)
(BLT,E10,0,WEDINP)
(BEQ,E10,0,OPTION)
;
E10=E10-1
;
;-------------------------
; CALL WHEEL INPUT TABLE
;
(EPP,START5,END5)
;
;-------------------------
E11=E10*8
(INP,"New wheel name (8ch) ? ",SYVAR(E11).8CH)
(EPP,START5,END5)
E11=E10+50
(INP,"Maximum wheel speed (m/s) ? ",E(E11))
(EPP,START5,END5)
E11=E10+56
(INP,"Start wheel diameter ? ",E(E11))
(EPP,START5,END5)
E11=E10+59
(INP,"Current wheel diameter ? ",E(E11))
(EPP,START5,END5)
E11=E10+62
(INP,"Minimum wheel diameter ? ",E(E11))
(EPP,START5,END5)
E11=E10+65
(INP,"Wheel width ? ",E(E11))
(EPP,START5,END5)
(DLY,0.2)
;
;-------------------------
; CALL WHEEL INPUT TABLE N02
;
(EPP,START6,END6)
;
;-------------------------
E11=E10+53
(INP,"Grit length ? ",E(E11))
(EPP,START6,END6)
E11=E10+68
(INP,"Diffusivity (m*m/s) ? ",E(E11))
(EPP,START6,END6)
E11=E10+71
(INP," Thermal conductvity (W/mK) ? ",E(E11))
(EPP,START6,END6)
(DLY,0.2)
;
;-------------------------
SK610=32
SK601=1
;
"WHEWA2"
(DLY,0.1)
(BNE,SK610,255,WHEWA2)
;
(BNE,SK606,255,ERROR)
;
(BNC,WHESEL)
;
;-------------------------
; MATL EDIT ROUTINE
;
"MATED"
;
;-------------------------
; CALL MATL DISPLAY TABLE
;
(EPP,START4,END4)
;
;-------------------------
"MADINP"
E12=0
(INP,"Edit matl No.? ",E12)
(BGT,E12,10,MADINP)
(BLT,E12,0,MADINP)
(BEQ,E12,0,OPTION)
;
E12=E12-1
;
;-------------------------
; CALL MATL INPUT TABLE
;
(EPP,START7,END7)
;
;-------------------------
E11=120+E12*6
(INP,"New material name (6 CH) ? ",SYVAR(E11).6CH)
(EPP,START7,END7)
E11=80+E12
(INP,"Max crit temp (deg C) ? ",E(E11))
(EPP,START7,END7)
E11=90+E12
(INP,"Diffusivity (m*m/s) ? ",E(E11))
(EPP,START7,END7)
E11=100+E12
(INP," Thermal conductvity (W/mK) ? ",E(E11))
(EPP,START7,END7)
E11=110+E12
(INP,"Grinding wheel speed (m/s) ? ",E(E11))
(EPP,START7,END7)
E11=120+E12
(INP,"Aspect ratio ? ",E(E11))
(EPP,START7,END7)
E11=130+E12
(INP,"Mean chip volume (mm**3) *E-6 ? ",E(E11))
(EPP,START7,END7)
;
;-------------------------
SK610=8
SK601=1
;
"MATWA2"
(DLY,0.1)
(BNE,SK610,255,MATWA2)
;
(BNE,SK606,255,ERROR)
;
(BNC,OPTION)
;
;-------------------------------------------------
; OPEN PROCESS FILE
;
"MCED"
SK611=2
SK601=1
(DLY,0.2)
;
(EPP,START8,END8)
;
"MC1INP"
E12=0
(INP,"Modify parameter ? ",E12)
(BEQ,E12,0,ENDPR1)
(BEQ,E12,1,PRO1)
(BEQ,E12,2,PRO2)
(BEQ,E12,3,PRO3)
(BEQ,E12,4,PRO4)
(BEQ,E12,5,PRO5)
(BEQ,E12,8,PRO8)
(BEQ,E12,9,PRO9)
(BEQ,E12,10,PRO10)
(BNC,MC1INP)
;
"PRO1"
(INP,"Bed length (mm) ? ",E33)
(EPP,START8,END8)
(BNC,MC1INP)
"PRO2"
(INP,"Safety distance (mm) ? ",E34)
(EPP,START8,END8)
(BNC,MC1INP)
"PRO3"
(INP,"Cycle distance (mm) ? ",E35)
(EPP,START8,END8)
(BNC,MC1INP)
"PRO4"
(INP,"Control wheel diam. (mm) ? ",E36)
(EPP,START8,END8)
(BNC,MC1INP)
"PRO5"
(INP,"Dwell (s) ? ",E37)
(EPP,START8,END8)
(BNC,MC1INP)
"PRO8"
(INP,"No. of grinds between dresses ? ",E32)
(EPP,START8,END8)
(BNC,MC1INP)
"PRO9"
(INP,"Bed length offset (mm) ? ",E38)
(BGT,E38,0.5,PRO9)
(BLT,E38,-0.5,PRO9)
(EPP,START8,END8)
(BNC,MC1INP)
;
"PRO10"
(INP,"Component allowance (mm) ? ",E39)
(BGT,E39,0.1,PRO10)
(BLT,E39,-0.1,PRO10)
(EPP,START8,END8)
(BNC,MC1INP)
;
;
"ENDPR1"
SK611=4
SK601=1
(DLY,0.2)
```

```
)
(BEQ,PASS0.10CH,PASS20.10CH,MC2ED)
(BEQ,PASS0.10CH,PASS30.10CH,MC2ED)
(BNC,OPTION)
;
;-------------------------
; EDIT MCDATA2 FILE
;
"MC2ED"
SK611=8
SK601=1
(DLY,0.2)
;
;-------------------------
; LOAD MCDATA2 DISPLAY
;
(EPP,START9,END9)
;
;-------------------------
"MC2INP"
E12=0
(INP,"Modify parameter ? ",E12)
(BEQ,E12,0,ENDPR2)
(BEQ,E12,1,PRO11)
(BEQ,E12,2,PRO12)
(BEQ,E12,3,PRO13)
(BEQ,E12,4,PRO14)
(BEQ,E12,5,PRO15)
(BEQ,E12,6,PRO16)
(BEQ,E12,7,PRO17)
(BEQ,E12,8,PRO18)
(BNC,MC2INP)
;
;-------------------------
"PRO11"
(INP,"Adapt. VF relax factor ( ) ? ",E30)
(EPP,START9,END9)
(BNC,MC2INP)
"PRO12"
(INP,"Adapt. VM relax factor ( ) ? ",E31)
(EPP,START9,END9)
(BNC,MC2INP)
"PRO13"
(INP,"Min cmpt revs/s ? ",E32)
(EPP,START9,END9)
(BNC,MC2INP)
"PRO14"
(INP,"Min Q ratio ? ",E33)
(EPP,START9,END9)
(BNC,MC2INP)
"PRO15"
(INP,"Power measurement filter ? ",E34)
(EPP,START9,END9)
(BNC,MC2INP)
"PRO16"
(INP,"Sizing device filter ? ",E35)
(EPP,START9,END9)
(BNC,MC2INP)
"PRO17"
(INP,"Data base learning relaxation ? ",E36)
(EPP,START9,END9)
(BNC,MC2INP)
"PRO18"
(INP,"Power limit perc/ms ? ",E37)
(BGT,E37,1,PLERR)
(BLT,E37,0.01,PLERR)
(EPP,START9,END9)
(BNC,MC2INP)
;
"PLERR"
(DEF,20,1,1," POWER LIMIT FACTOR NOT ",R)
(DEF,21,2,1," BETWEEN 0.1 AND 1      ",R)
(OUT)
(DLY,1)
(EPP,START9,END9)
(BNC,PRO18)
;
"ENDPR2"
SK611=16
SK601=1
(DLY,0.2)
;
(BNC,OPTION)
;
;-------------------------
; MACHINE OVER-RIDES FILE
;
"ORED"
;
; OPEN OVERRIDES FILE
;
SK611=32
SK601=1
(DLY,0.2)
;
(EPP,STAR10,END10)
;
;-------------------------
"ORINP"
E12=0
(INP,"Modify parameter ? ",E12)
(BEQ,E12,0,ENDPR3)
(BEQ,E12,1,PRO21)
(BEQ,E12,2,PRO22)
(BEQ,E12,3,PRO23)
(BEQ,E12,4,PRO24)
(BEQ,E12,5,PRO25)
(BEQ,E12,6,PRO26)
(BEQ,E12,7,PRO27)
(BEQ,E12,8,PRO28)
(BNC,ORINP)
;
"PRO21"
(INP,"Feeds and speeds over-ride : ",SYVAR0.6CH)
(EPP,STAR10,END10)
(BEQ,SYVAR0.6CH,"OFF    ",ORINP)
(BEQ,SYVAR0.6CH,"ON     ",ORINP)
(BNC,PRO21)
;
"PRO22"
(INP,"Adaptive control over-ride : ",SYVAR6.6CH)
(EPP,STAR10,END10)
(BEQ,SYVAR6.6CH,"OFF    ",ORINP)
(BEQ,SYVAR6.6CH,"ON     ",ORINP)
(BNC,PRO22)
;
"PRO23"
(INP,"Over shoot calc. over-ride : ",SYVAR12.6CH)
(EPP,STAR10,END10)
(BEQ,SYVAR12.6CH,"OFF    ",ORINP)
(BEQ,SYVAR12.6CH,"ON     ",ORINP)
(BNC,PRO23)
;
"PRO24"
(INP,"Pecking cycle over-ride : ",SYVAR18.6CH)
(EPP,STAR10,END10)
(BEQ,SYVAR12.6CH,"OFF    ",ORINP)
(BEQ,SYVAR12.6CH,"ON     ",ORINP)
(BNC,PRO24)
;
"PRO25"
E40=E30
(INP,"O-R grinding wheel (rpm) ? ",E30)
(BGT,E30,1600,VSORER)
(BLT,E30,1000,VSORER)
(EPP,STAR10,END10)
(BNC,ORINP)
;
"VSORER"
(DEF,18,1,1,"WHEEL SPEED OUT OF RANGE",R)
(OUT)
(DLY,1)
E30=E40
(EPP,STAR10,END10)
(BNC,PRO25)
;
"PRO26"
E40=E31
(INP,"O-R control wheel (rpm) ? ",E31)
(BGT,E31,40,VWORER)
(BLT,E31,10,VWORER)
(EPP,STAR10,END10)
(BNC,ORINP)
;
"VWORER"
(DEF,18,1,1,"WHEEL SPEED OUT OF RANGE",R)
(OUT)
(DLY,1)
E31=E40
(EPP,STAR10,END10)
(BNC,PRO26)
;
"PRO27"
E40=E32
(INP,"O-R feed rate (mm/s) ? ",E32)
(BGT,E32,7,VFORER)
(BLT,E32,0,VFORER)
(EPP,STAR10,END10)
(BNC,ORINP)
;
"VFORER"
(DEF,18,1,1,"FEED RATE OUT OF RANGE",R)
(OUT)
(DLY,1)
E32=E40
(EPP,STAR10,END10)
(BNC,PRO27)
;
"PRO28"
E40=E33
(INP,"O-R over shoot (mm) ? ",E33)
(BGT,E33,1,OSORER)
(BLT,E33,0,OSORER)
(EPP,STAR10,END10)
(BNC,ORINP)
;
"OSORER"
(DEF,18,1,1,"OVER SHOOT OUT OF RANGE",R)
```

```
(OUT)
(DLY,1)
E33=E40
(EPP,STAR10,END10)
(BNC,PRO28)
;
"ENDPR3"
SK611=64
SK601=1
(DLY,0.2)
;
(BNC,OPTION)
;
;-------------------------
; GP DISPLAY PAGE FOR
; COMPONENT SELECT AND
; EDIT
;
"START1"
SK610=1
SK601=1
;
"CMPWA1"
(DLY,0.1)
(BNE,SK610,255,CMPWA1)
;
(BNE,SK606,255,ERROR)
;
(SCR,ON)
(DEF,1,6,5,$YVAR0.6CH)
(DEF,2,7,5,$YVAR6.6CH)
(DEF,3,8,5,$YVAR12.6CH)
(DEF,4,9,5,$YVAR18.6CH)
(DEF,5,10,5,$YVAR24.6CH)
(DEF,6,11,5,$YVAR30.6CH)
(DEF,7,12,5,$YVAR36.6CH)
(DEF,8,13,5,$YVAR42.6CH)
(DEF,9,14,5,$YVAR48.6CH)
(DEF,10,15,5,$YVAR54.6CH)
;
(DEF,11,6,13,E30)
(DEF,12,7,13,E31)
(DEF,13,8,13,E32)
(DEF,14,9,13,E33)
(DEF,15,10,13,E34)
(DEF,16,11,13,E35)
(DEF,17,12,13,E36)
(DEF,18,13,13,E37)
(DEF,19,14,13,E38)
(DEF,20,15,13,E39)
;
(DEF,21,6,22,E40)
(DEF,22,7,22,E41)
(DEF,23,8,22,E42)
(DEF,24,9,22,E43)
(DEF,25,10,22,E44)
(DEF,26,11,22,E45)
(DEF,27,12,22,E46)
(DEF,28,13,22,E47)
(DEF,29,14,22,E48)
(DEF,30,15,22,E49)
;
(DEF,41,6,31,E50)
(DEF,42,7,31,E51)
(DEF,43,8,31,E52)
(DEF,44,9,31,E53)
(DEF,45,10,31,E54)
(DEF,46,11,31,E55)
(DEF,47,12,31,E56)
(DEF,48,13,31,E57)
(DEF,49,14,31,E58)
(DEF,50,15,31,E59)
;
(DEF,61,6,40,E60)
(DEF,62,7,40,E61)
(DEF,63,8,40,E62)
(DEF,64,9,40,E63)
(DEF,65,10,40,E64)
(DEF,66,11,40,E65)
(DEF,67,12,40,E66)
(DEF,68,13,40,E67)
(DEF,69,14,40,E68)
(DEF,70,15,40,E69)
;
(DEF,81,6,49,$YVAR60.6CH)
(DEF,82,7,49,$YVAR66.6CH)
(DEF,83,8,49,$YVAR72.6CH)
(DEF,84,9,49,$YVAR78.6CH)
(DEF,85,10,49,$YVAR84.6CH)
(DEF,86,11,49,$YVAR90.6CH)
(DEF,87,12,49,$YVAR96.6CH)
(DEF,88,13,49,$YVAR102.6CH)
(DEF,89,14,49,$YVAR108.6CH)
(DEF,90,15,49,$YVAR114.6CH)
;
(DEF,91,6,1," 1")
(DEF,92,7,1," 2")
(DEF,93,8,1," 3")
(DEF,94,9,1," 4")
(DEF,95,10,1," 5")
(DEF,96,11,1," 6")
(DEF,97,12,1," 7")
(DEF,98,13,1," 8")
(DEF,99,14,1," 9")
(DEF,100,15,1,"10")
(DEF,117,18,1," 0  Return to edit menu.")
;
(DEF,101,2,4,"LINER")
(DEF,102,3,4,"TYPE")
(DEF,103,2,13,"FINAL")
(DEF,104,3,13,"DIAM")
(DEF,105,4,13,"(mm)")
(DEF,106,2,22,"START")
(DEF,107,3,22,"DIAM")
(DEF,108,4,22,"(mm)")
(DEF,109,2,31,"INTERN")
(DEF,110,3,31,"DIAM")
(DEF,111,4,31,"(mm)")
(DEF,112,2,40,"LENGTH")
(DEF,113,4,40,"(mm)")
(DEF,116,2,49,"MATRL")
;
; TURN GRAPHIC OFF
;
SK614=255
(OUT)
;
"END1"
;
;-------------------------
; COMPONENT EDIT BIT
;
"START2"
(SCR,ON)
E11=E10+1
E11=E10*6
(DEF,1,3,34,$YVAR(E11).6CH)
E11=30+E10
(DEF,2,10,52,E(E11))
E11=40+E10
(DEF,3,3,49,E(E11))
E11=50+E10
(DEF,4,10,4,E(E11))
E11=60+E10
(DEF,5,17,29,E(E11))
E11=60+E10*6
(DEF,6,3,42,$YVAR(E11).6CH)
;
(DEF,8,1,49,"START")
(DEF,9,2,49,"DIAM")
(DEF,10,1,34,"LINER")
(DEF,11,2,34,"TYPE")
(DEF,12,1,42,"MATRL")
(OUT)
SK614=1
"END2"
;
;-------------------------
; GP DISPLAY PAGE FOR
; WHEEL SELECT AND
; EDIT
;
"START3"
SK610=16
SK601=1
;
"WHEWA1"
(DLY,0.1)
(BNE,SK610,255,WHEWA1)
;
(BNE,SK606,255,ERROR)
;
(SCR,ON)
(DEF,1,6,6,$YVAR0.8CH)
(DEF,2,8,6,$YVAR8.8CH)
(DEF,3,10,6,$YVAR16.8CH)
(DEF,4,6,3,"1.")
(DEF,5,8,3,"2.")
(DEF,6,10,3,"3.")
(DEF,7,6,15,E50)
(DEF,8,8,15,E51)
(DEF,9,10,15,E52)
(DEF,10,6,24,E56)
(DEF,11,8,24,E57)
(DEF,12,10,24,E58)
(DEF,13,6,33,E59)
(DEF,14,8,33,E60)
(DEF,15,10,33,E61)
(DEF,16,6,42,E62)
(DEF,17,8,42,E63)
(DEF,18,10,42,E64)
(DEF,19,6,51,E65)
(DEF,20,8,51,E66)
(DEF,21,10,51,E67)
;
(DEF,22,2,6,"WHEEL")
(DEF,23,2,15,"VMAX")
(DEF,24,3,15,"(m/s)")
```

```
(DEF,25,2,24,"INITIAL")
(DEF,26,3,24,"DIAM")
(DEF,27,4,24,"(mm)")
(DEF,28,2,33,"CURRENT")
(DEF,29,3,33,"DIAM")
(DEF,30,4,33,"(mm)")
(DEF,31,2,42,"MINIMUM")
(DEF,32,3,42,"DIAM")
(DEF,33,4,42,"(mm)")
(DEF,34,2,51,"WIDTH")
(DEF,35,3,51,"(mm)")
(DEF,117,18,1," 0  Return to edit menu.")
;
; GRAPHIC OFF
;
SK614-255
(OUT)
;
"END3"
;
;--------------------------
; GP DISPLAY PAGE FOR
; MATERIAL EDIT
;
"START4"
SK610-4
SK601-1
;
"MATWA1"
(DLY,0.1)
(BNE,SK610,255,MATWA1)
;
(BNE,SK606,255,ERROR)
;
(SCR,ON)
(DEF,1,6,5,$YVAR120.6CH)
(DEF,2,7,5,$YVAR126.6CH)
(DEF,3,8,5,$YVAR132.6CH)
(DEF,4,9,5,$YVAR138.6CH)
(DEF,5,10,5,$YVAR144.6CH)
(DEF,6,11,5,$YVAR150.6CH)
(DEF,7,12,5,$YVAR156.6CH)
(DEF,8,13,5,$YVAR162.6CH)
(DEF,9,14,5,$YVAR168.6CH)
(DEF,10,15,5,$YVAR174.6CH)
;
(DEF,11,6,13,E80)
(DEF,12,7,13,E81)
(DEF,13,8,13,E82)
(DEF,14,9,13,E83)
(DEF,15,10,13,E84)
(DEF,16,11,13,E85)
(DEF,17,12,13,E86)
(DEF,18,13,13,E87)
(DEF,19,14,13,E88)
(DEF,20,15,13,E89)
;
(DEF,21,6,22,E90)
(DEF,22,7,22,E91)
(DEF,23,8,22,E92)
(DEF,24,9,22,E93)
(DEF,25,10,22,E94)
(DEF,26,11,22,E95)
(DEF,27,12,22,E96)
(DEF,28,13,22,E97)
(DEF,29,14,22,E98)
(DEF,30,15,22,E99)
;
(DEF,31,6,31,E100)
(DEF,32,7,31,E101)
(DEF,33,8,31,E102)
(DEF,34,9,31,E103)
(DEF,35,10,31,E104)
(DEF,36,11,31,E105)
(DEF,37,12,31,E106)
(DEF,38,13,31,E107)
(DEF,39,14,31,E108)
(DEF,40,15,31,E109)
;
(DEF,41,6,40,E110)
(DEF,42,7,40,E111)
(DEF,43,8,40,E112)
(DEF,44,9,40,E113)
(DEF,45,10,40,E114)
(DEF,46,11,40,E115)
(DEF,47,12,40,E116)
(DEF,48,13,40,E117)
(DEF,49,14,40,E118)
(DEF,50,15,40,E119)
;
(DEF,51,6,47,E120)
(DEF,52,7,47,E121)
(DEF,53,8,47,E122)
(DEF,54,9,47,E123)
(DEF,55,10,47,E124)
(DEF,56,11,47,E125)
(DEF,57,12,47,E126)
(DEF,58,13,47,E127)
(DEF,59,14,47,E128)
(DEF,60,15,47,E129)
;
(DEF,61,6,58,E130)
(DEF,62,7,58,E131)
(DEF,63,8,58,E132)
(DEF,64,9,58,E133)
(DEF,65,10,58,E134)
(DEF,66,11,58,E135)
(DEF,67,12,58,E136)
(DEF,68,13,58,E137)
(DEF,69,14,58,E138)
(DEF,70,15,58,E139)
;
(DEF,71,6,1," 1")
(DEF,72,7,1," 2")
(DEF,73,8,1," 3")
(DEF,74,9,1," 4")
(DEF,75,10,1," 5")
(DEF,76,11,1," 6")
(DEF,77,12,1," 7")
(DEF,78,13,1," 8")
(DEF,79,14,1," 9")
(DEF,80,15,1,"10")
(DEF,117,18,1," 0  Return to edit menu.")
;
(DEF,81,2,5,"MATRL")
(DEF,82,3,5,"TYPE")
(DEF,83,2,13,"MAX CRIT")
(DEF,84,3,13,"TEMP")
(DEF,85,4,13,"deg C.")
(DEF,86,2,22,"DIFFUS")
(DEF,87,3,22,"m*m/s")
(DEF,88,4,22,"*E-6")
(DEF,89,2,31,"THERMAL")
(DEF,90,3,31,"COND")
(DEF,91,4,31,"W/mK")
(DEF,92,2,40,"WHEEL")
(DEF,93,3,40,"SPEED")
(DEF,94,4,40,"m/s")
(DEF,95,2,47,"ASPECT")
(DEF,96,3,47,"RATIO")
(DEF,97,2,58,"CHIP")
(DEF,98,3,58,"mm**3")
(DEF,99,4,58,"*E-6")
;
; GRAPHIC OFF
;
SK614-255
(OUT)
;
"END4"
;
;--------------------------
; EDIT WHEEL BIT
;
"START5"
(SCR,ON)
;
(DEF,22,2,6,"WHEEL")
(DEF,23,2,15,"VMAX")
(DEF,24,3,15,"(m/s)")
(DEF,25,2,24,"INITIAL")
(DEF,26,3,24,"DIAM")
(DEF,27,4,24,"(mm)")
(DEF,28,2,33,"CURRENT")
(DEF,29,3,33,"DIAM")
(DEF,30,4,33,"(mm)")
(DEF,31,2,42,"MINIMUM")
(DEF,32,3,42,"DIAM")
(DEF,33,4,42,"(mm)")
(DEF,34,2,51,"WIDTH")
(DEF,35,3,51,"(mm)")
E11=E10+1
(DEF,4,6,2,E11)
E11=E10*8
(DEF,1,6,6,$YVAR(E11).8CH)
E11=E10+50
(DEF,7,6,15,E(E11))
E11=E10+56
(DEF,10,6,24,E(E11))
E11=E10+59
(DEF,13,6,33,E(E11))
E11=E10+62
(DEF,16,6,42,E(E11))
E11=E10+65
(DEF,19,6,51,E(E11))
"END5"
;
"START6"
(SCR,ON)
E11=E10+1
(DEF,4,6,2,E11)
E11=E10*8
(DEF,1,6,6,$YVAR(E11).8CH)
E11=E10+53
(DEF,7,6,15,E(E11))
E11=E10+68
(DEF,10,6,24,E(E11))
E11=E10+71
```

```
(DEF,13,6,33,E(E11))
(DEF,22,2,6,"WHEEL")
(DEF,23,2,15,"GRIT")
(DEF,24,3,15,"SIZE")
(DEF,31,4,15,"(mm)")
(DEF,25,2,24,"DIFFUS")
(DEF,26,3,24,"m*m/S")
(DEF,27,4,24,"*E-6")
(DEF,28,2,33,"THERMAL")
(DEF,29,3,33,"COND")
(DEF,30,4,33,"W/mK")
(OUT)
"END6"
;
;-------------------------
; MATL FILE EDIT DISPLAY
;
"START7"
(SCR,ON)
E11=E12+1
(DEF,1,6,1,E11)
E11=120+E12*6
(DEF,2,6,5,SYVAR(E11).6CH)
E11=80+E12
(DEF,3,6,13,E(E11))
E11=90+E12
(DEF,4,6,22,E(E11))
E11=100+E12
(DEF,5,6,31,E(E11))
E11=110+E12
(DEF,6,6,40,E(E11))
E11=120+E12
(DEF,7,6,47,E(E11))
E11=130+E12
(DEF,8,6,58,E(E11))
;
(DEF,9,2,5,"MATRL")
(DEF,10,3,5,"TYPE")
(DEF,11,2,13,"MAX CRIT")
(DEF,12,3,13,"TEMP")
(DEF,13,4,13,"deg C.")
(DEF,14,2,22,"DIFFUS")
(DEF,15,3,22,"m*m/s")
(DEF,16,4,22,"*E-6")
(DEF,17,2,31,"THERMAL")
(DEF,18,3,31,"COND")
(DEF,19,4,31,"W/mK")
(DEF,20,2,40,"WHEEL")
(DEF,21,3,40,"SPEED")
(DEF,22,4,40,"m/s")
(DEF,23,2,47,"ASPECT")
(DEF,24,3,47,"RATIO")
(DEF,25,2,58,"CHIP")
(DEF,26,3,58,"mm**3")
(DEF,27,4,58,"E-6")
(DLY,0.2)
(OUT)
"END7"
;
;-------------------------
; MCDATA1 I/O DISPLAY
;
"START8"
(SCR,ON)
(DEF,1,6,14,"0. Return to main menu   ")
(DEF,2,7,14,"1. Bed length (mm)       :")
(DEF,3,7,40,E33,6)
(DEF,4,8,14,"2. Safety distance (mm) :")
(DEF,5,8,40,E34,6)
(DEF,6,9,14,"3. Cycle distance (mm)  :")
(DEF,7,9,40,E35,6)
(DEF,8,10,14,"4. Ctrl wheel diam (mm) :")
(DEF,9,10,40,E36,6)
(DEF,10,11,14,"5. Dwell (s)           :")
(DEF,11,11,40,E37,6)
(DEF,12,12,14,"6. Total no. of grinds  :")
(DEF,13,12,40,E30,6)
(DEF,14,13,14,"7. No. since dressed    :")
(DEF,15,13,40,E31,6)
(DEF,16,14,14,"8. No. between dresses  :")
(DEF,17,14,40,E32,6)
(DEF,18,15,14,"9. Bed offset (mm)      :")
(DEF,19,15,40,E38,6)
(DEF,20,16,13,"10. Cmpt allowance (mm)  :")
(DEF,21,16,40,E39,6)
(OUT)
"END8"
;
;-------------------------
; MCDATA2 I/O DISPLAY
;
"START9"
(SCR,ON)
(DEF,1,6,14,"0. Return to main menu   ")
(DEF,2,7,14,"1. Adapt. VF relax ( )  :")
(DEF,3,7,40,E30,6)
(DEF,4,8,14,"2. Adapt. VW relax ( )  :")
(DEF,5,8,40,E31,6)
(DEF,6,9,14,"3. Min. cmpt. revs/s    :")
(DEF,7,9,40,E32,6)
(DEF,8,10,14,"4. Min Q ratio          :")
(DEF,9,10,40,E33,6)
(DEF,10,11,14,"5. Power filter    ( )  :")
(DEF,11,11,40,E34,6)
(DEF,12,12,14,"6. Sizing filter   ( )  :")
(DEF,13,12,40,E35,6)
(DEF,14,13,14,"7. Adapt. DB relax ( )  :")
(DEF,15,13,40,E36,6)
(DEF,16,14,14,"8. Power limiter ( )    :")
(DEF,17,14,40,E37,6)
(OUT)
"END9"
;
;-------------------------
; MACHINE OVER-RIDES DISPLAY
;
"START10"
(SCR,ON)
(DEF,1,6,14,"0. Return to main menu   ")
(DEF,2,7,14,"1. Speed/feed over-ride :")
(DEF,3,7,40,SYVAR0.6CH)
(DEF,4,8,14,"2. Adapt ctrl over-ride :")
(DEF,5,8,40,SYVAR6.6CH)
(DEF,6,9,14,"3. Over shoot over-ride :")
(DEF,7,9,40,SYVAR12.6CH)
(DEF,8,10,14,"4. Peck cycle over-ride :")
(DEF,9,10,40,SYVAR18.6CH)
(DEF,10,11,14,"5. O-R GDG wheel (rpm)  :")
(DEF,11,11,40,E30,6)
(DEF,12,12,14,"6. O-R CTR wheel (rpm)  :")
(DEF,13,12,40,E31,6)
(DEF,14,13,14,"7. O-R feed rate (mm/s) :")
(DEF,15,13,40,E32,6)
(DEF,16,14,14,"8. O-R over shoot (mm)  :")
(DEF,17,14,40,E33,6)
(OUT)
"END10"
;
;-------------------------
; CYCLE DEFINITION DISPLAY
;
"STAR11"
;
(SCR,ON)
(DEF,1,6,08,"0. Return to main menu   ")
(DEF,2,7,08,"1. Allow 1 (mm) :")
(DEF,3,7,26,E130,6)
(DEF,4,8,08,"2. Allow 2 (mm) :")
(DEF,5,8,26,E132,6)
(DEF,6,9,08,"3. Allow 3 (mm) :")
(DEF,7,9,26,E134,6)
(DEF,8,10,08,"4. Allow 4 (mm) :")
(DEF,9,10,26,E136,6)
;
(DEF,10,7,35,"5. Time const 1 :")
(DEF,11,7,53,E131,6)
(DEF,12,8,35,"6. Time const 2 :")
(DEF,13,8,53,E133,6)
(DEF,14,9,35,"7. Time const 3 :")
(DEF,15,9,53,E135,6)
(DEF,16,10,35,"8. Time const 4 :")
(DEF,17,10,53,E137,6)
(OUT)
"END11"
;
;-------------------------
"NOPRI"
(SCR,ON)
(DEF,1,1,1," INCORRECT PASS-WORD ",R)
(OUT)
(DLY,2)
(BNC,ENDPRG)
;
;-------------------------
"ERROR"
(SCR,ON)
(DEF,1,1,1," DATA BASE I/O ERROR ",R)
(OUT)
(DLY,2)
(BNC,ENDPRG)
;
;-------------------------
"ENDPRG"
```

# SETBED

## Sub-routine to peform calibration of machine bed length

```
;-------------------------
; A little routine to allow
; setting of the machine bed
; length.
;
; Sean Kelly 1988
;
; Modified to retract infeed
; by 10mm after logging.
;
; Sean Kelly 21/11/88
;
;-------------------------
; RESET ALL PARAMS WITHOUT
; SETTTING DISP1
;
SK626=1
SK611=1
SK601=1
;
"DS1FI2"
(DLY,0.1)
(BNE,SK611,255,DS1FI2)
(BNE,SK606,255,SETEND)
;
;-------------------------
; TURN ON FAST CSI JUST IN CASE...
;
SK612=1
;
;-------------------------
; GET HOME POS BY CALL CALC VF,VW
;
SK602=1
;
"VFFINY"
(DLY,0.1)
(BNE,SK602,255,VFFINY)
;
;-------------------------
; CALL DISPLAY NO. 2
;
SK602=2
SK601=1
;
"DS2FIY"
(DLY,0.1)
(BNE,SK611,255,DS2FIY)
;
;-------------------------
SK620=3
SK621=0
SK622=0
SK623=0
;
;-------------------------
; RUN WHEELS UP TO SPEED?
;
(CLS,WHEELS/MP2)
(BEQ,SK624,16,BEDS1)
(BNE,SK624,0,SETEND)
;
;-------------------------
; CALL SUBROUTINE TO CALIBRATE
; PROBES
;
"BEDS1"
(CLS,CALSI2)
;
;-------------------------
; SET UP JOG MODE
; CALL TO SIPROM
;
"JOGMOD"
SK621=3
SK623=6
SK200=1
;
;-------------------------
; SAVE TARGET POS ON EXIT FROM JOG
; DELAY TO ALLOW SYNCH
;
(DLY,0.1)
E89=E66
;
;-------------------------
; RETRACT TO PREVIOUS POS
;
M11
E88=E89+10
G XE88
M10
SK621=6
SK623=0
;
;-------------------------
; STORE SIZING READINGS
;
SK242=1
(DLY,3)
E71=E57
E72=E58
SK242=0
;
;-------------------------
; E86=LARGEST VALUE
;
(BGE,E71,E72,BIGE71)
E86=E72+E87
(BNC,CDTEST)
;
"BIGE71"
E86=E71+E87
;
;
;-------------------------
; TEST COMPONENT DIMENSIONS
;
"CDTEST"
;
(BLT,E86,40,CDERR)
(BGT,E86,120,CDERR)
(BNC,CALBED)
;
;-------------------------
"CDERR"
SK621=4
(DLY,2)
SK621=6
(BNC,SETEND)
;
;-------------------------
; CALC NEW BED LENGTH
;
"CALBED"
E86=E86-E89
;
;-------------------------
; TEST BEDLEN DIMENSIONS
;
(BLT,E86,40,BEDERR)
(BGT,E86,150,BEDERR)
(BNC,CSIOFF)
;
;-------------------------
"BEDERR"
SK621=5
(DLY,2)
(BNC,SETEND)
;
;-------------------------
; TURN OFF CSI
;
"CSIOFF"
SK612=0
;
; OPEN PROCESS FILE
;
SK611=2
SK601=1
(DLY,0.2)
;
E33=E86
E38=0
;
; CLOSE PROCESS FILE WITH NEW BED LENGTH
;
SK611=4
SK601=1
(DLY,0.2)
;
; TURN ON CSI
;
SK612=1
;
;-------------------------
"SETEND"
SK626=0
```

# WHEELS

## Sub-routine to run grinding and control wheel to required speed

---

```
;-------------------------
; Subroutine to run wheels up
; to calc speed.
;
; -Time out after 45s
; -Will not accel wheels if
;  within 0.95/1.05 * calc speed
;
;-------------------------
; RETRACT INFEED TO CYCLE HOME
;
SK621-E0
(BGT,E0,0,MINVS)
;
"REP:"
SYVAR.3CH="Y  "
(INP,"INFEED TO CYCLE HOME ? (Y/N) ",SYVAR.3CH)
(BEQ,SYVAR.3CH,"Y  ",HOME1)
(BEQ,SYVAR.3CH,"N  ",MINVS)
(BNC,REPT)
;
;-------------------------
"HOME1"
M11
G XE50
M10
SK621-1
;
"MINVS"
SK624-0
;
;-------------------------
; MIN GRINDING WHEEL SPEED
;
E81=0.95*E25
;
; MIN CONTROL WHEEL SPEED
;
E82=0.95*E56
;
;-------------------------
; MAX GRINDING WHEEL SPEED
;
E78=1.05*E25
;
; MAX CONTROL WHEEL SPEED
;
E79=1.05*E56
;
;-------------------------
; TEST WHEEL SPEEDS
;
(BLT,E28,E81,REPT1)
(BGT,E28,E78,REPT1)
;
(BLT,E59,E82,REPT1)
(BGT,E59,E79,REPT1)
;
(BNC,VSOK)
;
;-------------------------
;
"REPT1"
SYVAR.3CH="Y  "
(INP,"RUN WHEELS TO SPEED ? (Y/N) ",SYVAR.3CH)
(BEQ,SYVAR.3CH,"Y  ",WHEELS)
(BEQ,SYVAR.3CH,"N  ",WHEND1)
(BNC,REPT1)
;
;-------------------------
"WHEELS"
;
M03 SE25
(USS,sE56)
;
;-------------------------
; WAIT UNTIL SPEED REACHED OR TIME OUT (45s)
; E80-90 SCRATCH PAD
;
E80=TIM
;
; WHEEL ACCEL MESSAGE
;
SK623=1
;
;-------------------------
"REPT2"
(DLY,0.1)
(BGT,E59,E82,VWOK)
E83=TIM-E80
(BGT,E83,45,ERR1)
(BNC,REPT2)
;
;-------------------------
"VWOK"
SK622=1
E80=TIM
;
"REPT3"
(DLY,0.1)
(BGT,E28,E81,VSOK)
E83=TIM-E80
(BGT,E83,45,ERR1)
(BNC,REPT3)
;
;-------------------------
"VSOK"
SK622=2
SK623=0
(DLY,0.2)
SK622=0
(BNC,WHEND2)
;
;-------------------------
; WHEEL ACCEL TIME OUT
;
"ERR1"
M03 S0
(USS,s0)
M05
SK623=2
SK624=8
SK564=4
(BNC,WHEND3)
;
;-------------------------
; WHEELS NOT RUN UP BY OPER
;
"WHEND1"
SK624=16
(BNC,WHEND3)
;
;-------------------------
;WHEELS OK FLAG
;
"WHEND2"
SK624=0
;
;-------------------------
"WHEND3"
```

# CALSIZ

## Sub-routine to calibrate sizing devices on master workpiece

```
;-------------------------
;CALIBRATE SIZING DEVICES
;
"SIZER"
SYVAR.3CH-"Y  "
(INP,"CALIBRATE SIZING DEVICE ? (Y/N) ",SYVAR.3CH)
(BEQ,SYVAR.3CH,"Y  ",REPT5)
(BEQ,SYVAR.3CH,"N  ",SIZEND)
(BNC,SIZER)
;
;-------------------------
; CHECK FOR AT HOME E0,BY = 1
;
"REPT5"
(BEQ,E0,1,CAL2)
(INP,"INFEED TO CYCLE HOME ? (Y/N) ",SYVAR.3CH)
(BEQ,SYVAR.3CH,"Y  ",HOME2)
(BEQ,SYVAR.3CH,"N  ",CAL2)
(BNC,REPT5)
;
"HOME2"
M11
G XE50
M10
SK621-1
;
;-------------------------
; CHECK TARGET CMPT LOADED
;
"CAL2"
SK622-0
SYVAR.3CH-"Y  "
(INP,"TARGET COMPONENT LOADED ? (Y/N) ",SYVAR.3CH)
(BEQ,SYVAR.3CH,"Y  ",SIZE)
(BEQ,SYVAR.3CH,"N  ",SIZER)
(BNC,CAL2)
;
;-------------------------
; ZERO SIZERS ON TEST PIECE
;
"SIZE"
SK623-3
SK242-1
(DLY,5)
;
SK602-4
"NOTCAL"
(DLY,0.1)
(BNZ,SK602,255,NOTCAL)
(DLY,0.5)
;
SK623-0
SK622-4
SK242-0
;
"SIZEND"
```

# DRESS

## Sub-routine to perform grinding wheel dressing procedure

```
;-------------------------
; Subroutine to organise
; wheel dressing
;
;-------------------------
; GET HOME POS BY CALL CALC VF,VW
;
SK602=1
;
"VFFINX"
(DLY,0.1)
(BNE,SK602,255,VFFINX)
;
;-------------------------
; CALL DISPLAY NO. 2
;
SK602=2
SK601=1
"DS2FIX"
(DLY,0.1)
(BNE,SK611,255,DS2FIX)
;
;-------------------------
SK620=2
SK621=0
SK622=0
SK623=0
SK624=0
SK625=0
;
;-------------------------
; RUN WHEELS UP TO SPEED?
;
(CLS,WHEELS/MP2)
(BNE,SK624,0,DREND)
;
;
;-------------------------
"WDRESS"
$YVAR.3CH="Y  "
(INP,"Wheel dressed ? (Y/N) ",$YVAR.3CH)
(BEQ,$YVAR.3CH,"Y  ",YDRESS)
(BEQ,$YVAR.3CH,"N  ",DREND)
(BRC,WDRESS)
;
;-------------------------
; LOAD COUNTERS AND CSI OFF
;
"YDRESS"
SK612=0
;
SK611=2
SK601=1
(DLY,0.2)
;
; ZERO NO. SINCE DRESSED COUNTER
;
E31=0
;
SK611=4
SK601=1
(DLY,0.2)
;
;-------------------------
; ADJUST WHEEL DIAMETER
; LOAD WHEEL PARAMETERS
; FROM WHEEL NO. E11
;
SK610=16
SK601=1
(DLY,0.2)
;
E12=59+E11-1
E(E12)=E(E12)-0.05
;
; SAVE WHEEL PARAMETERS
;
SK610=32
SK601=1
(DLY,0.2)
;
;-------------------------
; CSI ON
;
SK612=1
(BRC,DREND2)
;
;-------------------------
; WHEELS NOT DRESSED FLAG
;
"DREND"
SK625=255
;
;-------------------------
"DREND2"
```

## Appendix 4. Custom screen displays

This appendix includes the principal displays that were generated by the CSI and ASSET programs for the intelligent control system. Figure 41 illustrates the initial system setup data display generated by CSI. Real time data is output by CSI in the displays illustrated in Figures 42 and 43. An example of workpiece definition data is illustrated in the ASSET generated display of Figure 44.

SUBSYS 1 [redacted] 00:05:58 #3

| GDG WHEEL | VMAX m/s | DIAM (mm) |
|---|---|---|
| UNIVERS1 | 45 | 609.200 |

| M/C | |
|---|---|
| CYCLE NO | 112 |
| DRESS IN | 27 |

| COMPONENT | FINAL DIAM (mm) | START DIAM (mm) | INTRN DIAM (mm) | LENGTH (mm) | MATERIAL |
|---|---|---|---|---|---|
| GKN.01 | 103.248 | 103.650 | 98.2000 | 225.000 | C.I. |

OPTION

1. GRIND
2. EDIT
3. SET BED
4. DRESS
5. NEW BAT
6. QUIT

Figure 41. Initial system setup display

SUBSYS 1 [redacted] 00:05:58 #4

| AXES | | actual | program | M/C | |
|---|---|---|---|---|---|
| GDG WHEEL | rpm | 0.000 | 1410.76 | CYCLE NO | 112 |
| CTRL WHEEL | rpm | 0.000 | 25.10 | DRESS IN | 27 |
| FEED RATE | mm/s | 0.0000 | 0.0998 | PENDANT START TO GRIND<br>O P2 TO QUIT | |
| FEED POSN | mm | +0.0000 | +0.0000 | | |
| MOTOR POWER | kW | 0.00 | 52.90 | EC 19.7628<br>Us 0.0000<br>kW 0.00 NL 0.00 | |

| COMPONENT | FINAL DIAM (mm) | START DIAM (mm) | INTRN DIAM (mm) | LENGTH (mm) | MATERIAL |
|---|---|---|---|---|---|
| GKN.01 | 103.248 | 103.650 | 98.200 | 225.000 | C.I. |

| | | | | |
|---|---|---|---|---|
| SIZE 1 | 103.248 | ADAPT | OSHOT | LIMIT |
| SIZE 2 | 103.248 | DBASE | PECKR | |

Figure 42. Real time grinding cycle display

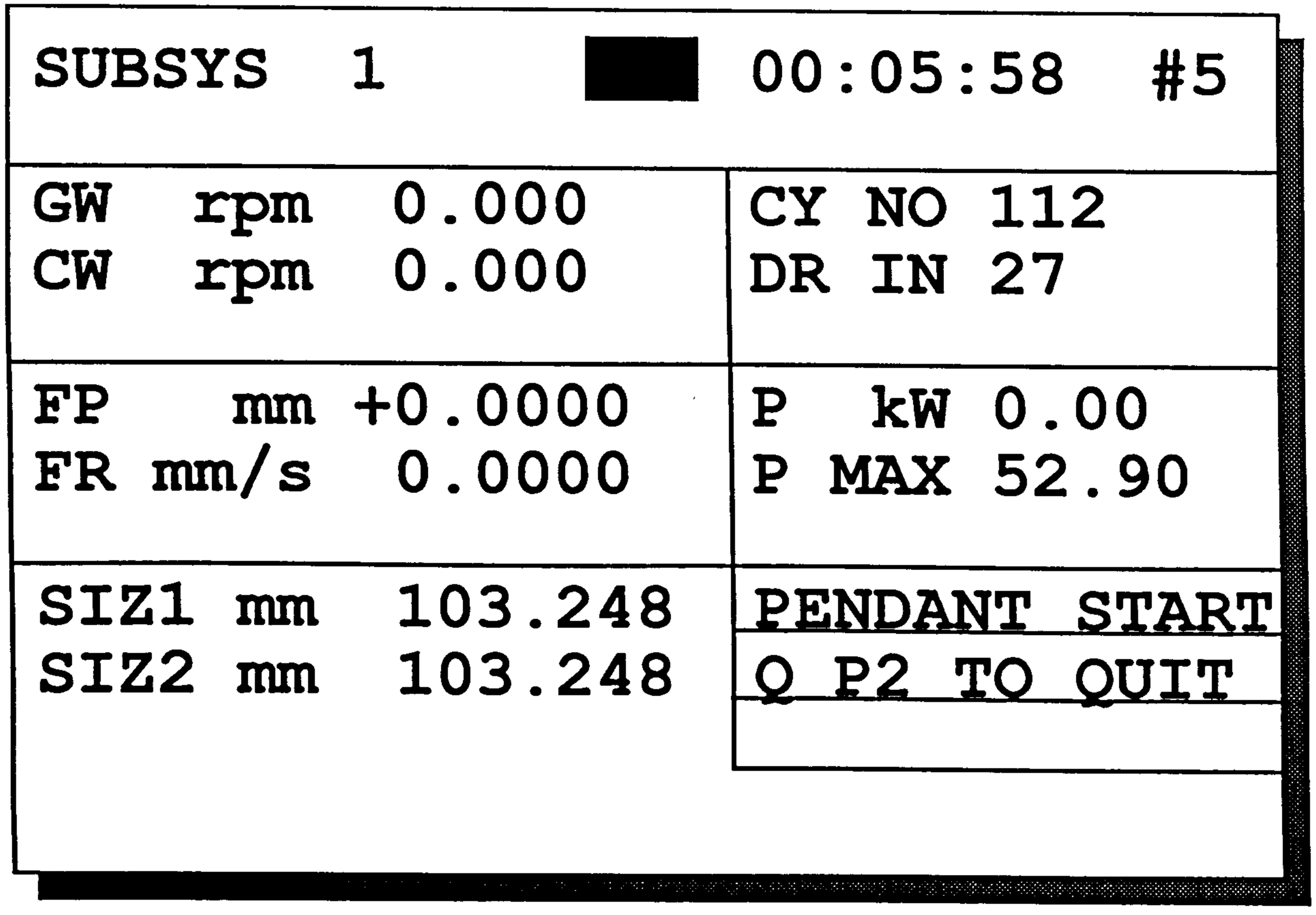



Figure 43. Large character real time grinding cycle display

SUBSYS 1 [redacted] 00:05:58 #2
Edit component No. ? 0_

| | LINER TYPE | FINAL DIAM (mm) | START DIAM (mm) | INTERN DIAM (mm) | LENGTH (mm) | TIME CONST | MATRL |
|---|---|---|---|---|---|---|---|
| 1 | GKN.01 | 103.200 | 103.65 | 98.2 | 225 | 16 | C.I.01 |
| 2 | GKN.02 | 103.220 | 103.65 | 98.2 | 225 | 20 | C.I.02 |
| 3 | GKN.03 | 103.240 | 103.65 | 98.2 | 225 | 19 | C.I.03 |
| 4 | GKN.04 | 103.260 | 103.65 | 98.2 | 225 | 21 | C.I.04 |
| 5 | GKN.05 | 103.280 | 103.65 | 98.2 | 225 | 19 | C.I.05 |
| 6 | GKN.06 | 103.400 | 103.65 | 98.2 | 250 | 19 | C.I.06 |
| 7 | GKN.07 | 103.420 | 103.65 | 98.2 | 250 | 17 | C.I.07 |
| 8 | GKN.08 | 103.440 | 103.65 | 98.2 | 250 | 19 | C.I.08 |
| 9 | GKN.09 | 103.460 | 103.65 | 98.2 | 250 | 16 | C.I.09 |
| 10 | GKN.10 | 103.480 | 103.65 | 98.2 | 250 | 19 | C.I.10 |

0 Return to edit menu.

Figure 44. Workpiece data edit display