A MONITOR FOR REAL-TIME CONTROL SYSTEMS

E. Wulff, B.E.

Thesis project carried out as part of a Master of Engineering Science course at the University of New South Wales.

Table of Contents

PREAMBLE

This thesis was started in 1969 as part of a Master of Engineering Science course at the school of Electrical Engineering at the University of NSW (UNSW) and completed and submitted in 1972. The subjects studied were mainly computer science subjects. An exception was *Reliability Engineering*, a topic which I had previously already been very interested in while designing large electronic switching systems for the telephone industry. In my career as a Software Engineer I concentrated very much on producing reliable software and developed methods to ensure that software was reliable. Since this thesis was produced when I was first introduced to computer software, it already shows my efforts to bring an engineering discipline to the field of software design.

The first version of the Real Time Operation System 'OSCAR' described in this thesis was implemented in 1970 on the PDP-8 computer of the school of Electrical Engineering. It worked beautifully with just the Teletype I/O and high speed tape reader and punch I/O. At the time I was working for the CSR company in Sydney at their research laboratory designing electronic switching systems for factory control using TTL logic IC's. I convinced the company to buy a Data General Nova computer in 1970 to develop industrial control systems using mainly software. The Nova was a more powerful machine. It had 4 16 bit Accumulators compared to a single 12 bit Accumulator for the PDP-8. Hence I ported OSCAR to the Nova computer, without changing the basic design. It worked well for a number of projects carried out at the research laboratory (2 of which are described in the thesis). Unfortunately I was never allowed to publish any papers about this work at the time. It was deemed to be too valuable and was classed as company confidential. Even this thesis was not allowed to be put in the University of NSW library for 10 years and I had to sign a non-disclosure agreement for the next 10 years.

In 1988 I was working for a manufacturer of PLC's in Germany and was given the task of writing the firmware for their new generations of PLC's based on National Semiconductor and later on Siemens 16 bit microprocessors. In both cases I ported the OSCAR design to these processors. The PLC's produced had built in networking and were very fast and reliable. The colleagues working with me particularly liked the modular structure of the OSCAR system. In 2018 I was most gratified to learn, that one of the colleagues working with me on the PLC's was still using the OSCAR design for industrial control computing at another large German manufacturer.

For the 50th anniversary I have taken on the job of scanning the typed manuscript of the thesis using modern optical character recognition (OCR) to produce a LibreOffice (ODF) document. I had to do hardly any corrections – OCR is that good now.

I hope many programmers will still gain some useful insights into an alternate way to structure interrupt systems and task scheduling in a way which is fast and clear to follow.

John E. Wulff

Bowen Mountain, Australia

January 2019

PS: Only minor changes and spelling corrections have been made for this copy.

1 INTRODUCTION

The use of digital computers in a real-time environment imposes many restrictions on programmers which are not encountered in conventional data-processing. These restrictions can be traced to timing problems in making the computer keep pace with the real world. The tendency of modern computers to become faster and faster tends to alleviate such problems, but this tendency also opens the possibility of using digital computers in more complex high speed systems. In the long run it is important that the real-time computer programmer has at his disposal a programming system which handles details of synchronisation with events external to the computer and breaks up execution of segments of program in such a manner that all timing requirements are met.

This thesis will analyse a number of schemes presented in the literature for achieving such aims. It will then outline a system which combines what are felt to be the best features of systems in the literature. From this a working Operating System has been developed.

This Operating System has been used. in two computer control systems, one of which is operational at the time of writing and the other is nearing completion. It has been called OSCAR which is short for "Operating System, C.S.R. Automation Research". The work was carried out at the C.S.R. Research Laboratories in Sydney, Australia.

The use of multi-task systems has now become accepted although at the beginning of this work it was quite rare. Many publications stressed the pitfalls of concurrent operations, pointing out that the sequence in which instructions are executed cannot be defined and hence not tested[1]. Wegner has overcome this problem by introducing the concept of the 'instantaneous description'[2]. The interrupt facility of computers was seen as the main stumbling block, because it effectively inserts instructions at unforeseen points in a program[1]. To a lesser degree data channel transfers modify memory in parallel with program execution. This does not appear to cause as much alarm, possibly because of the well-defined hardware sequence which controls these transfers. Later I will endeavour to show that most interrupt controlled data transfers can be made to operate just like hardware controlled data channel transfers.

A point which I see as a major stumbling block in coming to grips with the interrupt system of a computer is that there is a popular misconception of its function when it comes to incorporating it in a programming system. Because the action of trapping to another location at some indeterminate point in a program forms the major departure from normal operation, it is often forgotten that an interrupt is only the reply by some external device to a previous activation. Instead interrupt handlers are often structured as if interrupts occurred completely spontaneously. This manifests itself in interrupt and device handlers and sometimes even complete systems, which start at the interrupt locations and work their way through a massive number of switches which have recorded the mode of the system at any point in time[3].

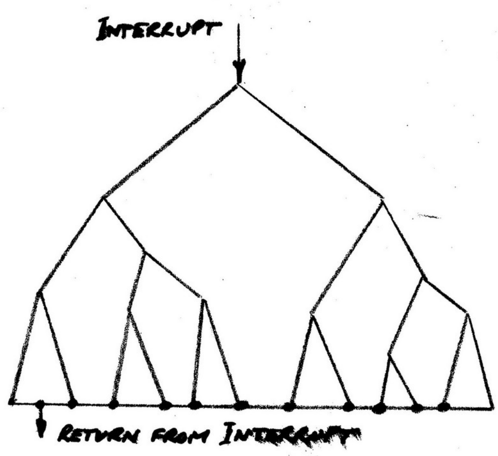


Fig 1

Figure 1 illustrates such a system which is characterised by a tree structure whose root is the interrupt location and whose branches are open ended. They consist of segments of program which terminate when no further computing can be carried out until the next interrupt signals that new data is available. Then the whole maze is traversed again to locate which segment must be executed next. When an attempt is made to introduce multiple interrupts into such a system the whole concept breaks down. Most serious systems therefore do not follow the above approach completely, but nevertheless it stands as the basic philosophy of most systems and appears to guide the thinking of most programmers in dealing with the interrupt.

It is my thesis that if multi-programming systems are structured in such a way as to reflect the simple fact that an interrupt is a signal from a peripheral device that an action initiated by a computer program is now completed, then the way is open to a clear and orderly system which can be debugged simply, and whose operations can be tested rigorously.

This problem has been overcome by the introduction of Synchronising Primitives. Primitive operations have the property that only one at a time can be executed by concurrent processes. These operations will be described in detail in a later section.

1.1 Real-Time Systems

For this thesis a 'real-time' environment is defined as one is which the time scale of computer operations is critical and is dictated by the requirements of an environment external to the computer[1]. The 'response time' of a system becomes important in the real-time context. Adequate response times vary for different applications. Examples quoted in the literature are a few milliseconds for radar scanning systems, three seconds for Airline reservation systems and five minutes for controlling a paper mill.

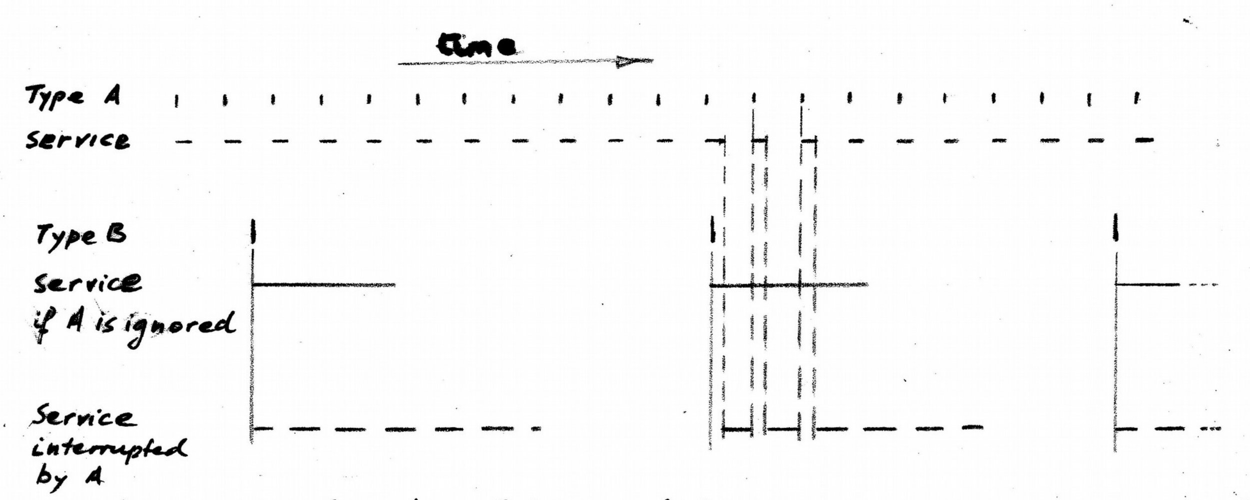
One of the specifications for the operating system outlined in this thesis was a response time of substantially less than one millisecond for certain services. This response time is not required for all services. What is important is, that the priority of a service program and the time to execute service programs at different priorities is such, that a satisfactory response time for a given service can be achieved.

1.1.1 Occupancy

To visualise the interrelations of a number of computer programs the concept of computer occupancy is useful.

$$Occupancy = \frac{\text{Time to service an event}}{\text{Time bettween events}}$$

If we assume that a computer is servicing two types of events simultaneously, Type A at 1 ms intervals and Type B at 100 ms intervals. If we also assume that the time to service each of these events is 60 µs and 6 ms respectively. The occupancy for either type of event is 6% and the occupancy for both together is 12%. If the Type A event causes the highest priority computer response and the Type B event the second highest priority response, it is easy to see that the response time to the-.Type A event is 60 µs. For the Type B event the response time is 12 ms which is made up of the 6 ms actual servicing time interspersed by 100 higher priority services at 60 µs duration, totalling another 6 ms.



Servicing 2 types of Events

Fig. 2

If events activate processes which are executed at different priorities, we can define the following terms -

Time between events activating process `i` = te_i

`te` can be estimated from an analysis of the application when designing the system.

Time to execute process `i` = tx_i

`tx` can be estimated when writing the code for process `i` by counting instructions.

The following relations summarise the previous discussion if we assume processes are numbered in order of priority:

Occupancy of process `i`

$$O_i = \frac{tx_i}{te_i}$$

Total occupancy of processes `1` to `n`

$$\mathrm{TO}_n = \sum_{i=0}^{n} \frac{tx_i}{te_i}$$

Response time of process `n`

$$\mathrm{tr}_n = \mathrm{TO}_n * te_n$$

We can now quantify the definition of a real-time system and state that in a real-time computer system the Total Occupancy of all processes in the system must be less than unity.

By estimating te and tx. while programming it becomes fairly obvious which pieces of code should be made to run efficiently. In programming one often looses sight of the relation of a particular piece of code to the whole program. The above concepts have proved to be helpful as a guideline while programming real-time systems.

1.1.2 A typical system

To highlight some of the features which are required in a real-time operating system and to illustrate some of the time constraints which are encountered in practice, a typical system will now be described. This system is a computer controlled installation for blending and metering liquids, such as petrol. At a distribution depot road tankers are filled with petrol. Because different types of petrol are marketed, various additives must be incorporated in the correct proportions as the tankers are filled. The amount of petrol, the grade and the proportion of each additive for a particular load are entered into the system at a keyboard terminal. This is done as orders are received, and may be several days in advance of actual delivery. The keyboard terminal must respond to a number of questions typed by the operator and its functioning should be independent of the actual filling of tankers. It may be necessary to allow for more terminals if the amount of work requires it. Again terminals should appear to be independent of each other.

A tanker filling station consists of a number of valves or pumps which allow different grades of petrol to flow into the tanker. The flow is measured with an accurate flow-meter. Such meters produce pulses, one pulse for every increment of volume which passes through the meter. These pulses interrupt the computer which counts them to integrate the flow. By arranging to turn off the flow when a given number of pulses have occurred, a given volume of petrol can be metered out. In practice temperature compensation would be desirable to convert this to a mass flow. To incorporate a given proportion of additives, a fixed volume of each additive is pumped into the petrol for a computed number of increments on the main flow. If several additives are required this produces quite a complex sequence of pumping actions.

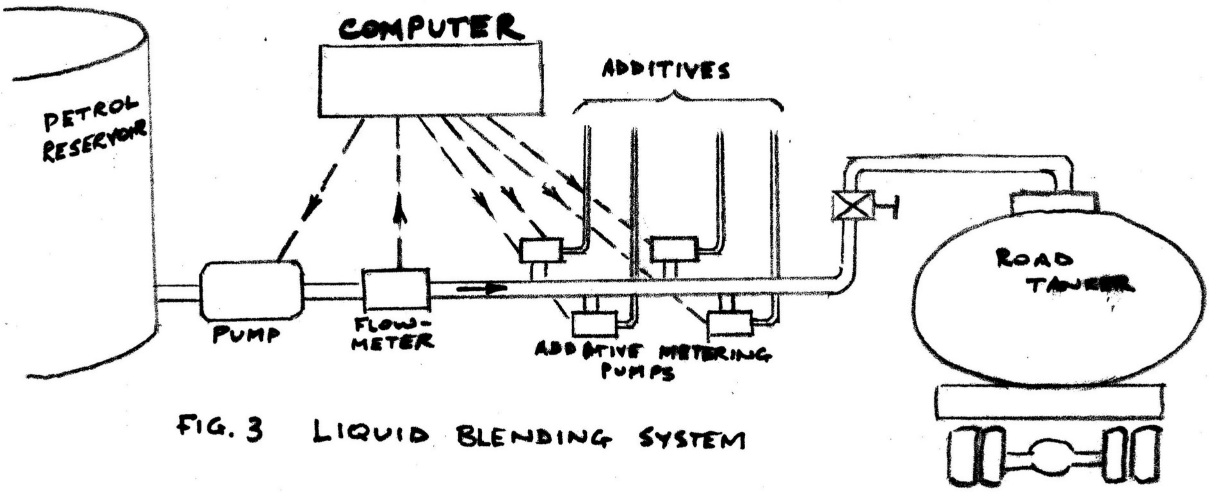


FIG. 3 LIQUID BLENDING SYSTEM

Typical flow-meters with the required accuracy would interrupt the computer 2,000 times a second. Thus te in the worst case is 500 µs. In the OSCAR system interrupt service for flow-meters(and the real-time clock) is 15 µs. Occupancy for each flow-meter (and the real-time clock) is thus 3%.

This shows immediately how many such flow-meters could be serviced in parallel. 3% is a reasonable figure and would allow the implementation of up to 10 filling stations giving a worst case occupancy of 30%. Interrupt handlers in other systems have a typical tx of 100 µs. This would immediately make the Occupancy for one flow-meter 20%. This would make the time-sharing of more than 5 flow-meter processes impossible. In practice something might be left over for the rest.

The proposed blending system is to be designed with 5 outlets. Thus 5 main flow-meters are required. Also the computer programs for each outlet would be very similar, since each outlet is essentially the same. The only difference is in the actual flow-meter and valve. Each outlet has its own set of these. This situation is best handled by what is known as re-entrant programs. Such programs may be active on a number of different instances of the same job at the same time. Thus a system to handle such work should cater for re-entrant programs. The same could be said for more than one terminal.

These could also be handled by multiple activations of the same. re-entrant program. OSCAR allows implementation of such systems.

1.1.3 Difficulties with Uni-programming

Uni-programming is the way most people program a computer. A program is seen as a single thread of instructions. All activities in the outside world will have to be brought into this single thread. This works if external events are initiated by the same program and if the computer can afford to

be idle while waiting for such an external event. This is usually wasteful and in the case of such demanding events as flow-meter pulses, it is hard to visualise a single thread of instructions keeping up with more than one pulse train.

In general such diverse activities as keeping up with a keyboard terminal and running a filling station as described in the last sections are difficult to join into a single program. Yet this is what many people are doing, and it is very hard work.

1.1.4 Multi-processor Systems

Some writers have suggested using more than one computer, one for each major job in a system. I agree with this approach if occupancy considerations make it necessary. Nevertheless, the need for some communication between the separate computers is still necessary, and an overall operating system is still required. Thus to use separate computers just to make programming easier is futile. The same can be achieved with a multi-task system such as OSCAR.

1.1.5 Time-sharing

Time-sharing computer systems as distinct from real-time systems are characterised by the fact that in the limit there are no real-time constraints in relation to the users as far as the computer is concerned. This means that there are a number of terminals connected to the computer and these interact with the computer and in the long run they share the computer equitably. In the short term nothing is lost if the job for one terminal is held up. Only the user's peace of mind and patience are tried while waiting for a printout which will come eventually. In practice this means that users may prefer to switch to a better system if the grade of service is poor. In such instances the concepts of real-time programming outlined in this thesis could be very useful to design a system which will give prompter service.

The starting point would be to classify the human user like any other real-time device. Apart from this the programming for many of the peripheral devices of a time-sharing system such as disks and line-printers can be treated in isolation if a real-time approach is adopted. Even multi-processor systems can be planned in this way, opening the way to a modular upgrading of a facility as its usage increases. Such an approach has been adopted on the latest Burroughs systems.

1.2 System Structure

To achieve adequate response times from a computer system while maintaining total occupancy below unity it is useful to employ a structure in which the total job is divided into a number of sections. Each section which we have previously called a *process* is required to service a particular event or series of events. For the programmer using an operating system such as will be outlined it is useful to regard processes as independent of each other except when deliberate interaction is introduced.

1.2.1 Process Independence

Such independence can best be visualised in a system in which each process is carried out on a separate computer. It then becomes a fairly simple matter to devise a program to carry out the servicing of a particular event. Once this service is completed the computer only has to wait for the next occurrence of the event to repeat the cycle. Such "Wait loops" are common in programs using unbuffered input and output from a single Teletype. Synchronisation with the device is achieved by testing a hardware flag and branching back to the test instruction if the flag indicates that the program should not proceed.

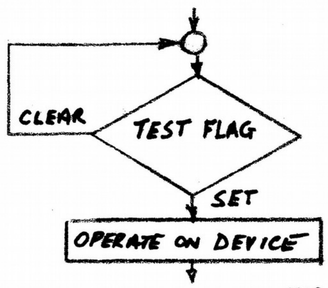


FIG. 4

Such a scheme whose flowchart is shown in Figure 4 is easy to understand by the average programmer, but it would be completely unacceptable in a real-time system.

Since the operation of repeatedly testing the hardware flag is not really useful except in the instance when the flag finally sets, it is possible to specify a system in which the testing of the hardware flag is replaced by a call to an operating system which has the effect of suspending the process which contains the call until the flag sets. During the period of suspension lower priority processes would be able to carry out their work.

Looked at from the outside a process using such a system call would be indistinguishable from a timing point of view from one using continuous testing of a hardware flag. More importantly the program using the system call is virtually unaltered. Only the hardware flag test loop has been replaced by a system call which we shall call the 'WAIT' call. This is the first of four synchronising primitive operations provided by the system.

1.2.2 Function of the Hardware Interrupt

Most digital computers have what is commonly called an interrupt facility. This facility was initially designed to allow overlapping of computing with Input/Output. This can cause serious problems to unwary programmers. It is important to ensure that only a well defined set of registers is modified during Interrupt Handling.

1.2.3 Instantaneous Description

The concept of 'Instantaneous Description' was used extensively in developing this system. Wegner defines it as the contents of all registers in the processing unit and memory of the computer at a given point of the computation[2]. Obviously it is inevitable that any form of interruption will modify such an instantaneous description. It is therefore necessary to settle on a reduced set of registers to which normal user programs are restricted leaving the remainder for interrupt service programs. It is then possible to avoid modification of the reduced instantaneous description.

The setting of a hardware flag marking the completion of some external operation is usually coupled with an interrupt. The interrupt can thus be regarded as the signal to continue operation of a process which has issued a WAIT call for that external operation. In a typical real-time system there are many peripheral devices which can interrupt the computer. It is the function of the INTERRUPT HANDLER program to establish which device is currently interrupting and to invoke the appropriate DEVICE SERVICE ROUTINE. Since all interrupts, whether they belong to high speed or slow speed devices have to pass through the interrupt handler, it is important to make this routine as short as possible to reduce system overhead which can be regarded as an unproductive form of occupancy. The latest implementation of the operating system described in this thesis carries out these functions in six computer instructions.

1.2.4 Timing Considerations

Communication with peripheral devices is a prime requirement in computing. This communication is different from ordinary program flow because the speed of peripheral devices is often based on mechanical movements whereas the messages can be received and transmitted by computers at much higher speeds. This problem is more serious in process control application where the devices that a computer has to communicate with are usually not designed to be computer compatible. Despite this it is the generally slow speed of peripheral devices which makes multiprogramming feasible. The average time to service a device (`tx`) is usually much less than the average time between steps of a device (`te`). Thus the occupancy for servicing the device is less than unity and the remaining computer occupancy can be shared among other devices.

This view of looking at the time scale of operation in the computer purely from the point of view of the delay caused by devices is not always valid but it is an important consideration in all systems which are Input/Output bound. In OSCAR it provides the means for fitting service for all devices into the available time. This implies that there is a priority structure which gives precedence to devices with high rates of activity and allows their service routines to interrupt the service for slower devices. The allocation of priorities is the responsibility of the users of this system. The exact choice is not very critical unless the total occupancy approaches unity in which case the overall interaction of the different sections has to be considered most carefully. The system at this stage does not cater for more than one process which is completely compute-bound. This has not proved a disadvantage in the applications handled so far. It would be a fairly easy step to execute a number of compute-bound processes on a timed, round-robin basis.

1.2.5 Device Service Routines

The structure of the Device Service Routine in the system is a software extension of the hardware. The nucleus of the system only provides certain primitive operations which are not normally provided by computer hardware but which can be regarded by programmers as hardware functions.

Use of these primitives provides a uniform method of generating device service routines which are both efficient in terms of occupancy and easy to understand by the user. It was felt absolutely essential that the user could develop his own device service routines in view of the varied nature of devices encountered in process control situations. Also it was considered important that the user should not be forced to follow a very rigid pattern to implement his ideas.

For the above reasons the philosophy of the Device Service Routines found in a number of Real-Time Operating Systems has not been followed. The implementation in 'RTOS'[10] is typical. In that system a call to perform I/O is accompanied by five parameters which specify a logical device number, a device control word, a data pointer, a data item count and an error return point. Such a call is very useful for the transmission of groups of characters or words of fixed length but the overhead in using such a call for simpler operations, e.g. transmitting one single character to a device, is very high both in execution time and in lengthy calling sequences which a user has to write. Execution time becomes very high because each time the call is executed the full list of parameters has to be interpreted before the action required can be carried out. Nevertheless such a call can be implemented in OSCAR using the synchronising primitives and this has been done for a project involving a number of on-line display terminals.

The structure that has been chosen is based on the criterion that activity both inside and outside the computer can be broken up into intervals whose transitions constitute 'events'. Particularly the actions of peripheral devices can be thought of as a series of events separated by periods of internal activity whose details are not of interest to the programmer.

The usual hardware method of synchronisation with computers follows this principle. A device is usually started by a pulse from the computer. This marks the beginning of a cycle, which is an event. The completion of the cycle is another event which is signalled to the computer by the device sending out a pulse. Since pulses are transient phenomena, the pulses in either direction usually set a bistable device or 'flag'. In the Nova range of computers the symmetry of this situation has been embodied in the design of the standard interface, shown in Figure 5.

In this type of structure, which is commonly called the 'Handshake' system, the 'Device' is activated by the steady state signal put out by the 'BUSY' flag which is set from the computer. When the 'Device' has completed its cycle, it sets the 'DONE' flag which signals the computer to do its share in continuing the action. From the point of view of the computer it starts a device and then waits for the answer signalling completion. Similarly from the point of view of the device, it signals completion of a cycle and then waits for a signal to carry out another cycle. The situation is quite symmetrical.

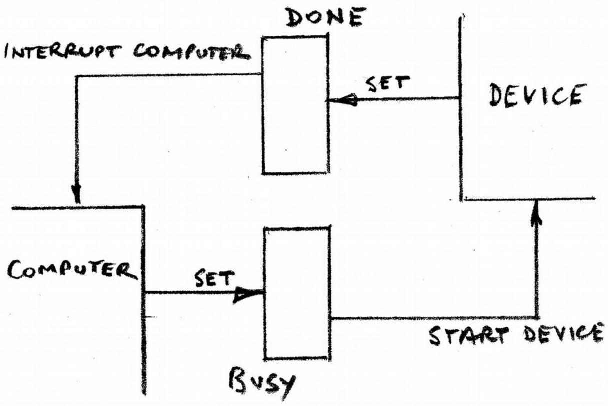


Fig. 5

The general strategy can also be implemented in the software of the computer. This part of the operating system can be thought of as a software interface between the computer I/O system and the user programs. Nico Haberman points out that a pair of hardware flags such as 'BUSY' and 'DONE' as in Figure 5 correspond to a software pair of flags operated on by primitive operations 'WAIT' and 'SIGNAL', which he defines[4]. These have not been used in OSCAR but they perform similar functions to operations in this system. He proves that two such flags are necessary and sufficient to synchronise communication between a program and a peripheral. He then draws the conclusion that there is essentially no need for a supervisory process through which all device requests are channelled and which is the only one that issues commands to the device. This course has been adopted in OSCAR.

In implementing this strategy it is not necessary to stick precisely to the hardware functions for a device. For example, it is possible to implement a software interface with a buffer for a device which only transmits single bytes, such as a Teletype reader. In this case the user makes a call on the operating system to initiate transfer of a number of bytes to the buffer. A buffer address pointer and buffer counter value must be set during this call. The user then waits until the last byte has been transmitted from the hardware. All transmission of bytes and storing of them is done in a device handler program which is activated at intervals by interrupts from the device. When the last byte has been stored the handler, which constitutes the software interface, signals the user to indicate to him that the event that he has been waiting for has now occurred.

This type of structure is analogous to the usual implementation of hardware interfaces for devices with very high transfer rates such as fixed-head discs. A hardware instruction presets the buffer address pointer. The buffer length is usually fixed. Another hardware instruction sets the disc address and initiates the transfer. As each word is received from the disc, it is stored in the computer memory through a hardware facility called a Data Channel or Direct Memory Access (DMA). When the last word has been received, the hardware causes a conventional interrupt to occur which in this case signals that a full buffer has been transferred.

In the software implementation, an analogous device interrupt service routine which is activated by each interrupt from the device takes the place of the Data Channel cycle. The interrupt when the buffer is full is replaced by the operation called 'POST'. This operation is the second synchronising primitive of OSCAR. It marks the occurrence of an event.

For each interrupt (1 character transmitted) except the last one, this Interrupt Service Routine typically steals about 12 instruction times from the program which has been interrupted. This is broken up into 2 instruction times for the interrupt hardware cycle, 3 instruction times for the Standard Interrupt Service which has to determine which device service to branch to, 4 instruction times for the Device Interrupt Service routine and 3 instruction times for the Return From Interrupt which restores the interrupted program.

Twelve instruction times for that part of the device service which is repeated many times is felt to be the best than can be done on the Nova computer on which these concepts were implemented. For other computers with more powerful interrupt hardware this time may be reduced and it is felt that time saved in repetitive interrupt servicing is always worthwhile.

Having disposed of the middle section of Device Service routines, we can now look at the beginning and end. The beginning must originate in a 'user' program which requests a transfer of one or more characters to a certain buffer. Apart from setting the buffer pointer and buffer counter, the device must be started to fetch the first byte. It is this first action which causes the first interrupt when the device has completed its first cycle. In the present implementation hardware I/O instructions are allowed to start the device. In later implementations it is proposed to use hardware which does not allow users to execute such instructions and a form of Supervisor Call will have to be used for initiating all I/O. A typical form of I/O call is shown in the code below. This code also shows the termination of the transfer.

Users, having initiated an I/O operation usually want to suspend their program until the transfer has been completed. This is achieved by the 'WAIT' call to the operating system. The 'WAIT' call has one parameter which is the address of a one word location called an 'EVENT CONTROL WORD'. In the example considered so far the transfer of the last character, which is detected by the Device Interrupt Service shown in Figure 5 results in POSTing the same EVENT CONTROL WORD. The WAIT and POST operation are a pair which achieve synchronisation, while the EVENT CONTROL WORD acts as a pair of flags which carry out a function similar to the BUSY and DONE flags in the hardware interface shown in Figure 5. A more detailed description of these functions will be given in Section 2.

To summarise the operation of the Device Service Routine: a user programme makes an I/O request; interrupts activate the Device Service Routine when required and finally the last interrupt allows control to return to the User.

NOTE

- The user program need not wait for completion of the call immediately the I/O request is made. It is possible to carry out further computation after the I/O request is made and then wait for completion of the transfer when the new data is required.
- In the case where transfer of only one byte is requested, the sequence reduces to the I/O request followed by a WAIT call. The first interrupt signals completion of the transfer and POSTs the caller.

The following code shows a routine to get a single character from a Teletype and the associated Interrupt Service Routine in Nova assembler language.

```
; GET CHARACTER ROUTINE FOR TELETYPE ;
; CALLING SEQUENCE:

          JSR     GET
          next statement

GET:      STA     3,6     ; SAVE RETURN
          .WAIT           ; WAIT FOR TRANSMISSION OF
          TTIEC           ; THE NEXT CHARACTER
```

```
          SUBC    3,3     ; CLEAR ACCUMULATOR 3
          STA     3,TTIEC ; CLEAR TTIEC
          DIAS    0,TTI   ; READ CHAR FROM TELETYPE
          JMP     @6      ; RETURN TO USER

; THE FOLLOWING WORD IS THE 'EVENT CONTROL WORD' LINKING THE TWO SECTIONS

TTIEC:    0               ; INITIALLY CLEARED

; TELETYPE INPUT INTERRUPT SERVICE
; AFTER SAVING  ACCUMULATOR 3,
; THE MAIN INTERRUPT HANDLER BRANCHES TO 'TTIS'
; WHEN A 'TTI' INTERRUPT OCCURS

TTIS:     NIOC    TTI     ; CLEAR THE DONE FLAG TO
                          ; PREVENT FURTHER INTERRUPTS
          .POSTI          ; POST OR SIGNAL THE USER
          TTIEC           ; VIA THE EVENT CONTROL WORD
                          ; SIGNALLING THAT THIS CHARACTER IS READY
```

This interrupt service programme appears somewhat trivial but it allows practical time-sharing or multiprogramming which is quite efficient, while maintaining the simplicity of structure of the Get Character routine using flag testing.

```
; BUSY WAIT GET CHARACTER ROUTINE FOR TELETYPE ;

GET:      SKPDN   TTI     ; WAIT FOR DONE FLAG TO SET
          JMP     .-1
          DIAS    O,TTI   ; READ CHARACTER
          JMP     0,3     ; RETURN TO USER
```

Further ideas on this topic will be taken up in Section 2.3.4 dealing with the synchronisation primitives WAIT and POST.

1.2.6 Processes and Tasks

The word Process has been used loosely in Section 1.2.1 to talk about a computation. E. W. Dijkstra has written a complete monograph on sequential processes[3] and the co-operation between them without ever giving a formal definition of a process. His processes appear to be Algol programs with curious appendages called "parbegin" and "parend", which suddenly endow these programs with a capability to exist in parallel. Lampson[6] summarises the characteristics of a process thus:

> "A process must have, at least conceptually, a processor of its own to run on".

He also speaks of a "process or a processor executing a program. The process is the logical, the processor the physical environment for this execution".

To allow this implementation of more than one logical process on a single processor, these processes must be multiplexed or time shared on the processor. A special data structure is used to carry out this function.

This data structure will be called *Task*. This Task concept is used in OS 360 and many of the concepts which follow have been taken from that system[7]. A Task consists of block of memory in which all those registers of the processor which it must share with other Tasks, are saved and a program which will be executed when the Task is run.

The block of memory in which the processor registers are saved is called the TASK CONTROL BLOCK (TCB). The Task Control Block holds much of the variable part of the instantaneous description mentioned in Section 1.2.2.

In OSCAR the registers saved in the TCB are the four Accumulators, the Carry bit, the Program Counter and eight memory locations. The TCB also contains two other words which are used by the synchronising operations.

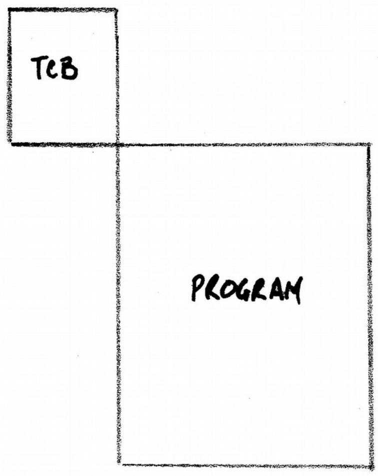


Fig. 6 A TASK

Closely connected with the instantaneous description of a task is the initial representation. This is the static value of the instantaneous description before execution is started. The initial representation is important for the practical implementation of a system. It allows initialisation of each task to be defined by the programmer during the program assembly phase. In OSCAR the initial representation follows immediately after each TCB. A program called SYSTEM START copies the initial representations into each TCB and then enters the task scheduler. This feature was not included in earlier implementations. Here the Initial Values of the registers of the Task Control Block were assembled as constants into the space occupied by the TCB. As a consequence the system could not be re-started once it had been run unless a complete reload was done. This proved tiresome in the real-time situation and the Initial Values were stored separately as part of the Task.

The ability to re-start a system has a more important advantage in systems for controlling machinery. Such systems are usually provided with a Task whose execution checks the operation of all the parts of the computing system as a low priority activity. If such a check uncovers a fault, it is often sufficient to record the occurrence of the fault, give an alarm and then re-start the system, hoping that the fault does not occur too frequently. This scheme allowed the successful operation of a computer control system in a remote location, despite the existence of a minor fault which was rectified during a subsequent regular visit.

1.2.7 Task Control Block

The most important register that must be saved in the TCB and for which an initial value must always be provided is the Program Counter of the processor. The initial value of the PC points to the statement in the program associated with the Task at which execution will start. Thus each Task written by the user has its own starting point which is normally associated only with independent stand-alone programs.

The provision of space for other registers is optional. On the Nova very little computing could be done without the 4 Accumulators and the Carry register so space is provided for these in the TCB. On top of this, certain memory locations are also saved in the TCB for each Task. Thus programs may use these as private memory locations which will not be disturbed by other Tasks using the same locations. In the first implementation of the system, only 2 such private locations were provided. These were chosen as Location 6 and 7 in the memory. (Example: program "GET" Section 1.2.5 uses Location 6 to save an accumulator). This system was tailored for speed and thus no other private registers were used. The latest implementation uses 8 private locations. Two of these are auto-incrementing and two are auto-decrementing registers. These registers are a hardware facility provided in only a few locations in low memory on the Nova. If these locations were not made private, this hardware facility could not be used effectively in tasks. The provision of private locations makes the writing of re-entrant programs much easier. This is very important in a multi-programming system. Re-entrant programs may be executed by a number of tasks simultaneously. They must have the property that they do not modify themselves. The private locations or a work area pointed to by a private location are then the only memory locations which a re-entrant program may modify.

The Floating Point Interpreter supplied by the manufacturers of the Nova is fully re-entrant and only requires private locations 6 and 7 and a work area whose address is in location 7 for each Task calling on the Floating Point Interpreter. This means that every task not only has its own pseudo-hardware processor but also its own floating point processor.

Floating Point Accumulators are stored in the Floating Work Area which can be regarded as an extension to the TCB. The TCB stores 3 other registers which are used by the operating system to schedule tasks. One is a Hardware Priority Mask which determines which device may interrupt the Task when it is active and which may not. The remainder are a Wait Count and a Back Pointer. The Wait Count is used when a Task is suspended either as a counter of how many events should be posted before re-activation, or as a link word, linking a number of suspended tasks into a queue. The Back Pointer gives a backward reference to the entry for this task in a list of all tasks. It is never modified.

A Task is identified by the address of the first location of its TCB. In this implementation 26 external names have been defined which range from TCBA to TCBZ. If the TCB addresses are given one of those names, the TCB will be put on to a Task Queue when the Operating System is loaded. The Priority of Tasks is determined by the ordering in the Task Queue. TCBA is always loaded first, TCBZ last. Thus a priority can be established by naming tasks with TCBA for the highest priority task and TCBZ for the lowest priority tasks and all others in between in alphabetical order. The priority of tasks means that if two tasks are ready for execution, the operating system will schedule the highest priority task first and execute it until it suspends itself or it is interrupted, and a higher priority task is activated as a result of the interruption. A lower priority task can only be executed when all higher priority tasks are suspended. This is a simple task-scheduling strategy which may be augmented in future. At present it satisfies the needs of the systems it is to serve.

1.2.8 Task States

When Tasks are executing they may exist in a number of different states. Only one of these states require a processor. OSCAR distinguishes three states which are described in the following section.

1. Active State: A task is active when a central processor is executing instructions in a program belonging to the task with data also belonging to the task, or shared with other tasks. To mark this state the address of the currently active TCB is stored in a known location (ATCB).
2. Ready State: A task is in this state when it is ready to use a central processor but is not active because higher priority tasks or system programs are using all physical processors.
3. Suspended State: A task is in the suspended state whenever it must wait for the occurrence of an event. Such an event may be the completion of an Input/Output operation, or the execution of one of the synchronising macro-instructions in another task which can re-activate a task.

Some systems[10] [11] recognise another state called the *Dormant* state which is said to be a state which is none of the previous three states. This is a state in which the Task either does not exist yet or the Task has been deleted. In the present implementation which does not cater for dynamic tasks, such a situation cannot occur. The nearest that a user could approach this state is to cause a task to be permanently suspended.

1.2.9 Task Implementation

It should be pointed out here that the method employed in generating tasks in a user system is to reserve space for the Task Control Block and give this space one of the names TCBA to TCBZ, consistent with the priority required. Immediately following the Task Control Block the user must provide an Initialisation Control Word (ICW) followed by a list of Initial values for the TCB. In particular the initial value of the program counter in the TCB must always be given and should have the label of the first instruction to be executed in the task.

This method of generating a task is a static one. Tasks are generated with the programs through the assembler and loader.

1.2.10 Concepts related to Tasks

Some thought has been given to an implementation using dynamic tasks. In such a system one task may create another task and cause it to be executed. In the language of OS 360, one task "attaches" another task. Conway and others[8] [3] have used the word 'fork'. The concept of 'forking' as seen by Conway is shown in Figure 7 He explains that the 'fork' and 'join' in flowcharts have their counterpart in the FORK and JOIN instruction

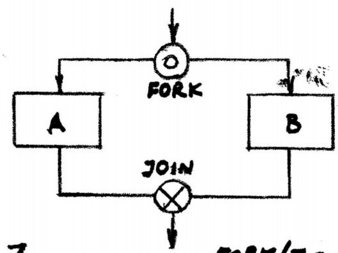


FIG. 7 FORK/JOIN

He defines as follows:

"FORK is simply an instruction with two successors. It is written and acts like a branch instruction".. The next statement will be executed as part of the current task but the location which the 'fork' branches to will be executed in parallel as part of a new task created at this point.

To implement such a scheme a new Task Control Block has to be obtained from a 'pool' set aside for this purpose and the forking point entered into the TCB as its PC. It is desirable to copy the rest of the current TCB into the newly created TCB so that the states of all the registers of the new task are the same as those of the current Task at the point of forking. RTOS which implements such a scheme also allows the specification of a priority for the new task. This is very important for creating real-time user systems.

The OSCAR System was written and working (May '70) 6 months before the preliminary specifications for RTOS had been seen by the author (Nov. '70), and 18 months before a full write up and source tapes and listings were obtained (Nov. '71). Certain similarities such as the abbreviation TCB, and the virtually identical implementation of the timed event queue (see Section 2.7.1) must stem from the common background literature and the likelihood of similar implementations of the same problem on the same computer.

The differences in implementation and overall strategy are of significance also and I would here like to justify my stand that a system using static Tasks is more useful to the average programmer for small real-time systems than dynamic tasks implemented by means of FORK instructions.

The biggest limitation of the FORK instruction is a conceptual one. Conway states that it should be conceived as a branch instruction. This is a realistic requirement in the implementation of algorithms in which parallelism may be exploited. The example usually given is one of the matrix manipulations which are obvious candidates for parallel execution. Conway[8] also specifically introduces the FORK instruction to allow the programmer access to a number of physical processors. Thus I see the FORK instruction as useful where a programmer wants to code a single problem in such a way as to exploit parallel execution or multiple processors and thus gain a speed advantage.

However, the problem usually facing the programmer in real-time systems (single processor only) is the requirement to code a number of separate sections each of which will probably run indefinitely. These sections can and should be isolated from other sections to allow them to be run and tested separately. The Static Task fulfils this requirement. Coding within the task is the same as coding for a free-standing program which has a full processor to execute. Communication with other Tasks is via well-defined Macro calls to the operating system. None of these resembles a branch into another Task.

It has been argued by Dijkstra[9] that the branch instruction in Algol, the GOTO, is unnecessary and spoils the block structure of many Algol programs. In a similar way a FORK (GOTO) into another Task is even more distressing because it tends to hide the true nature of a Task. An actual branch into another task is a meaningless concept because only the Program counter has been modified. Both before and after the branch have been executed, the same Task is still active. To appear to allow this situation in the special case of a Fork tends to dilute this fact.

The popularity of the Fork instruction probably stems from the fact that programmers are not used to thinking about their systems in a truly parallel sense. Because programs are sequential structures all the way except for the actions of peripheral devices, trained programmers tend to look at systems this way too. The present higher level languages only emphasise this situation because they take out I/O programming leaving only a single thread program. PL/I is the only well-known higher level language which recognises parallel processing but unfortunately the implementation is by the 'Attach' call which is the same as a 'Fork'. This appears to give parallel processing a slightly sequential look.

The suggestions for a parallel processing capability for Algol[5] made by Dijkstra is another approach to this problem. He suggests that a series of Algol statements be surrounded by the special statement bracket pair "parbegin" and "parend". This is to be interpreted as parallel execution of all the constituent statements. He calls the construction a "parallel compound" which is to be regarded as a statement. Initiation of a parallel compound implies simultaneous initiation of all its constituent statements. Although it is feasible to implement real-time control systems using this version of Algol it was initially thought of as a means for implementing algorithms which contain sections that can benefit from parallel execution.

1.2.11 Ideas taken from Hardware Design

My own approach to this problem has been guided largely by an education in engineering and some experience in the design of relay switching systems and later electronic switching and control systems. In this area everything is parallel. Every little building block goes about doing its small function all the time, reacting, with a finite delay, to its input signals and transmitting the result via its outputs to other blocks.

Similarly larger units can be thought of in a similar way doing their more complicated function at the same time as other units are doing theirs. It is surprising that computers which are devices of essentially this structure do not inspire programmers to emulate this structure.

It is always a great problem in switching system design to produce a system which is going to work without too many design faults or 'bugs' as computer people call them, when the system is built. A lot of work has been done in the last 20 years to provide methods of analysis and synthesis which make this job easier. Examples which come to mind are the Venn diagram or Karnaugh Map and the Huffman-Mealy method of sequential circuit analysis[1 8].

The greatest contribution to logic design has been the concept of strobing or clocking. This idea which was originated in the late 1950's simplified logic design by allowing the specification of logic in terms of state transition tables without worrying about the details of individual gate delays. Underlying the idea of strobing is the idea of an indivisible operation. The transmission of the leading edge of a strobe pulse can be regarded as a single indivisible operation and all elements which receive this strobe pulse are assumed to receive it at the same time. The leading edge of the strobe pulse marks the boundary in time between two discrete periods and any signal which develops in the second period is barred by the strobe pulse from causing any action in that period. Only after another strobe pulse may these signals cause any change in the output. This scheme would not work if the strobe pulses were divisible. Malfunction can occur in poorly designed systems if the strobe pulses are not generated by the same source or the distribution network causes different delays.

The concept of strobing can also be applied to software design. The first requirements is an indivisible operation. On most computers a machine instruction is an indivisible operation. This means that interrupts can only occur between two instructions, not halfway between an instruction. Many computers allow memory cycle stealing for data channels in the middle of instructions but this is not usually a problem. The interrupt is the only external event which directly affects program flow. An example of a common divisible operation which is often used and usually fails in an interrupt environment is the simple testing of a flag. In many computers a memory location must be loaded into an accumulator before its contents can be tested. An interrupt may occur between these two operations and before the flag is tested it may be altered, causing the wrong action when the test is finally carried out.

An example of an indivisible test for a computer is the instruction:-

```
INCREMENT MEMORY AND SKIP IF ZERO
```

If an interrupt occurs before this instruction has been executed the flag in memory has not yet been modified and execution after the return from interrupt will be correct. If the interrupt occurs just after the execution the program counter will have the value appropriate for the flag in memory before the interrupt but the interrupt routine will also see that the flag has already been tested. Unfortunately it is difficult to devise a scheme which is foolproof using this instruction only.

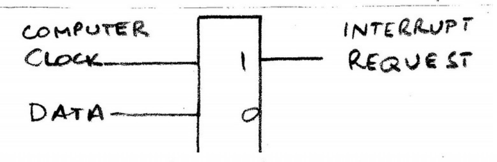
To simulate more powerful indivisible operations the interrupt flip-flop must be turned off for a number of instructions while the appropriate tests are carried out. Otherwise more elaborate instructions can be devised which means altering the computer hardware. On some machines this latter approach may be carried out with micro-programming.

In this thesis a number of indivisible operations for the purpose of software synchronisation will be developed and their correct operation will be demonstrated.

1.2.12 Events

The concept of Events is fundamental to the development of this system. Events are defined as instantaneous occurrences in the real time scale with the proviso that time is not continuous but is digitized into short intervals by a computer. The shortest such intervals usually correspond to the execution of one machine instruction. Events in the system environment always occur on the boundary between two such intervals. The Nova device interface system allows this scheme to be used even with external events. A line from the computer carries a clock pulse which is able to switch clocked flip flops after every instruction execution. Thus the data inputs to these flip flops are synchronised to the computer instruction cycle. Their outputs represent the same events as their inputs but they now fulfil the requirement that events should only occur on instruction boundaries.

This scheme is used in the Nova to synchronise Interrupts with the instruction flow.



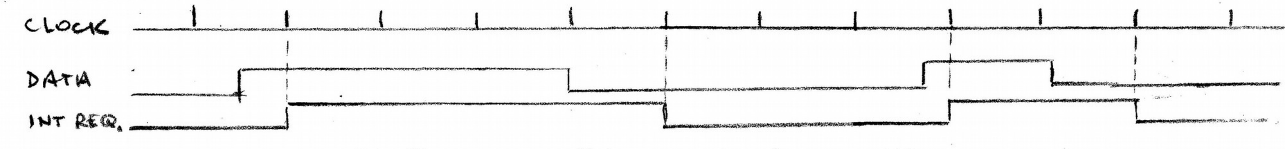


Fig. [redacted] NOVA Interrupt Synchronisation

External events can cause a computer to be interrupted or a computer program may test for the occurrence of an event. In the latter case no special hardware synchronisation is necessary. The execution of the test instruction will find the status line signalling the event either high or low, and the outcome of the test can only be one way. The outcome is available at the end of the instruction execution. Again the event is known to the process only in a discrete form. This way of looking at time is the same as in a sampled data system, except here we are mostly interested in a binary value. An event has either occurred or it has not yet occurred. The sampling process introduces quantizing errors into time estimates. The sampling rate of the computer is of the order of 1 MHz for present day machines and this would not introduce a serious error for most applications. But the number of instructions executed before an event can finally be acted upon is often quite large. This introduces further and larger quantizing errors.

In the OSCAR system every attempt has been made to reduce this time, which is usually called the response time, to a minimum. This has been done by avoiding the repetition of an operation where a single execution of that operation is sufficient to carry out an action. Thus continuous 'polling' in any form is avoided if at all possible. Secondly a conscious effort was made to optimise the code in critical routines for minimum execution time.

This practice has been found worthwhile because it had a side effect of producing a clearer structure. It is the author's view that the optimising of code can often be greatly assisted by modifications in the data structure a program has to operate on. Thus the data structures in the OSCAR system are seen as the most significant factor towards faster execution speed of the code.

Events in the system environment can also be thought of as the execution of particular instructions which are of significance to other processes. In this operating system the only instructions which show any effect in other processes are the synchronising primitives. One can then think of all the program which is executed up to a synchronising operation as part of the one event. This is the sense in which Simulation languages deal with events. Short programs are scheduled to take place at a given time. When this time arrives that program is executed. The effect of this execution is to change the state of the system and possibly schedule a new event. This scheduling takes place by sorting the new event into a queue of previously scheduled events according to the system time when the new event is due. Then the system clock is set forward to the time of the event on the head of the queue.

This system of scheduling was adopted for OSCAR with the modification that the system time now becomes real-time as measured by an external oscillator. Whenever the next event on the queue is not yet due the scheduling of events is suspended until the real-time has caught up with the time at which the next event is scheduled. This system of scheduling events has the advantage that the real-time clock oscillator need only increment one counter, yet events from many different tasks can be scheduled on the one clock queue.

1.2.13 Event Synchronization with Event Control Words

Event synchronization is the delaying of task execution until some specified event or events occur. The synchronization has two aspects:

1. The requirement for synchronization is stated explicitly by the WAIT meta-instruction or is implied by use of certain other instructions.
2. After the event has occurred, notice to the requesting task is given so it can proceed past the WAIT point.

The notification required is performed by the POST meta-instruction. When the event is known to the control program (for example, the completion of a read operation), the control program issues the POST. If the event is known only to the user's program, the user's program must issue it.

As an example, the function of both tasks A and B in Figure 9 is to compute some value, display it, and then proceed; the display to task A must precede that of B. Task A displays first, then issues the POST; task B waits for A, and then displays its results.

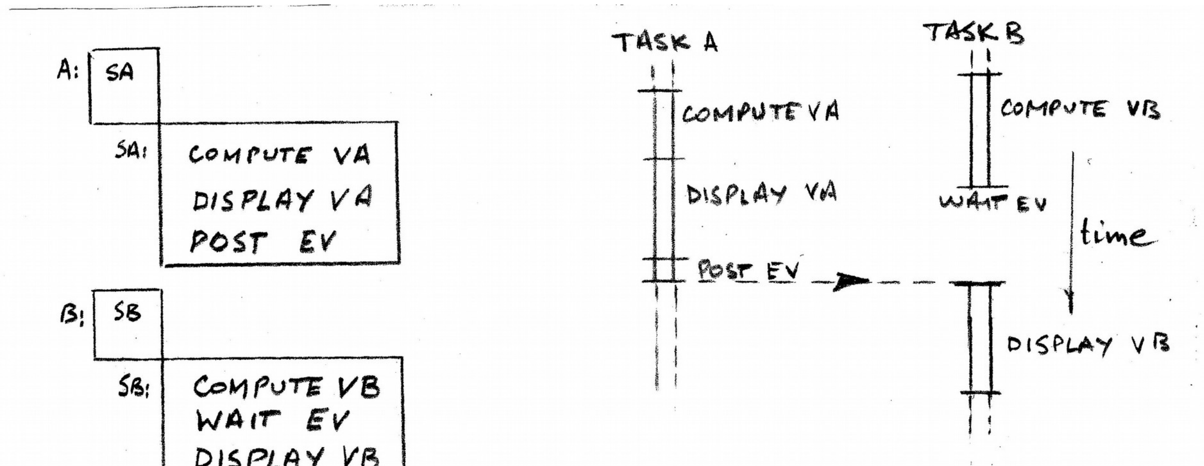


FIG. 9 INTERTASK SYNCHRONISATION

A task may make several different requests and then wait for any number of them. For example, a task may specify by READ, WRITE and DELAY meta-instructions that three asynchronous functions are to be performed. When each of these requests is made initially to the control program, the location of a one-word event control word (ECW) is also stated. The event control word provides the basic communication between the tasks issuing both the original requests and the subsequent wait, and the posting agency (in this case, the control program). When the WAIT meta-instruction is issued, the parameters supply the addresses of the event control words corresponding to the requested services. Also supplied is a wait count that specifies how many of the services (events) are required before the task is ready to continue. When an event occurs, the following takes place:

1. The completion flag in the appropriate event control word is set by the POST meta-instruction.
2. A wait count test is made to see if the number of 'completion flags' satisfies the wait condition, and hence if the task is ready.

After the task has again been given control, the programmer can determine what events did occur, and in what manner. He does this (with instructions following the WAIT meta-instruction) by testing each event control word.

Many requests for services may result in waits that are of no concern to the programmer - for example, GET and PUT subroutines to get and print a character from a Teletype (1.2.3). In these cases, event control words and wait specifications are handled entirely by the appropriate system subroutines.

The programmer is responsible for clearing event control words before each use. It is imperative that the event to which an event control word pertains has occurred before it is reused. System subroutines will do this before returning to the user. But it is important to clear all ECW's and this includes all system ECW's which are used implicitly when initialising a Task. This allows a clean re-start to take place.

Programmers intending to make use of the event synchronisation facilities will find the following example helpful.

A DELAY meta--instruction within program USER is followed by a WAIT for the completion of the input event. Figure 10 shows the situation immediately after DELAY is executed.

- The event control word required for the operation is located in a main storage area belonging to the task.
- Its address, "ECWA", was specified in the DELAY call.

- The appropriate input/ output program has queued the DELAY request and has placed the address ECWA in the queue element.

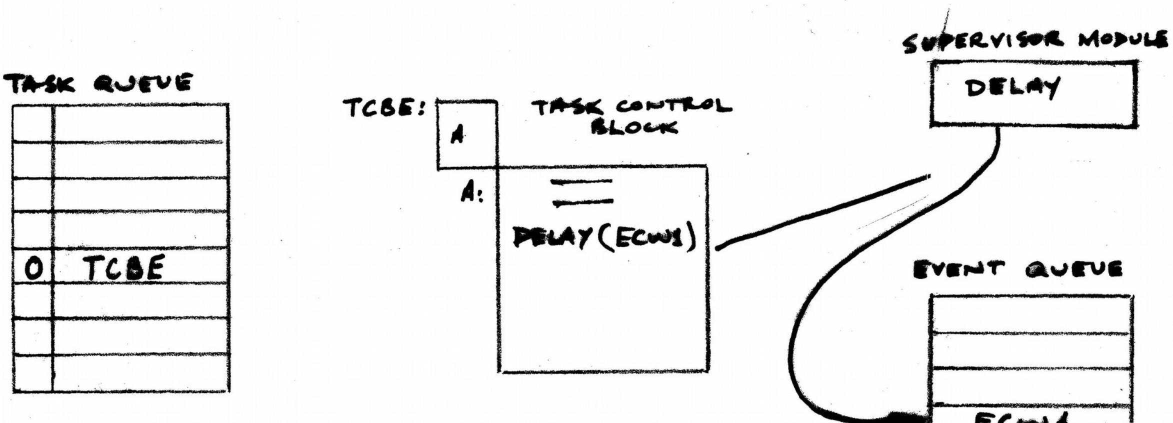


FIG. 10
AFTER EXECUTION
OF 'DELAY'

Figure 11 shows the situation at the time the WAIT meta-instruction is executed. In this example, the DELAY operation has not yet been concluded. The WAIT meta-instruction's parameters point to the event control word location, and state that only one event is needed to satisfy the WAIT. The operating system, as a result of the WAIT meta-instructions, performs these actions:

- Places the task control block address, "TCBA" in the event control word. Since this address is non zero, it means that a task is waiting for the event to take place.
- Sets a 1 in the WAIT count indicator in the task control block to show the number of events being awaited.
- Flags the task control block as being in the suspended state;
- therefore its task is no longer eligible to use the central processing unit. Passes control to the next ranking ready task on the Task queue.

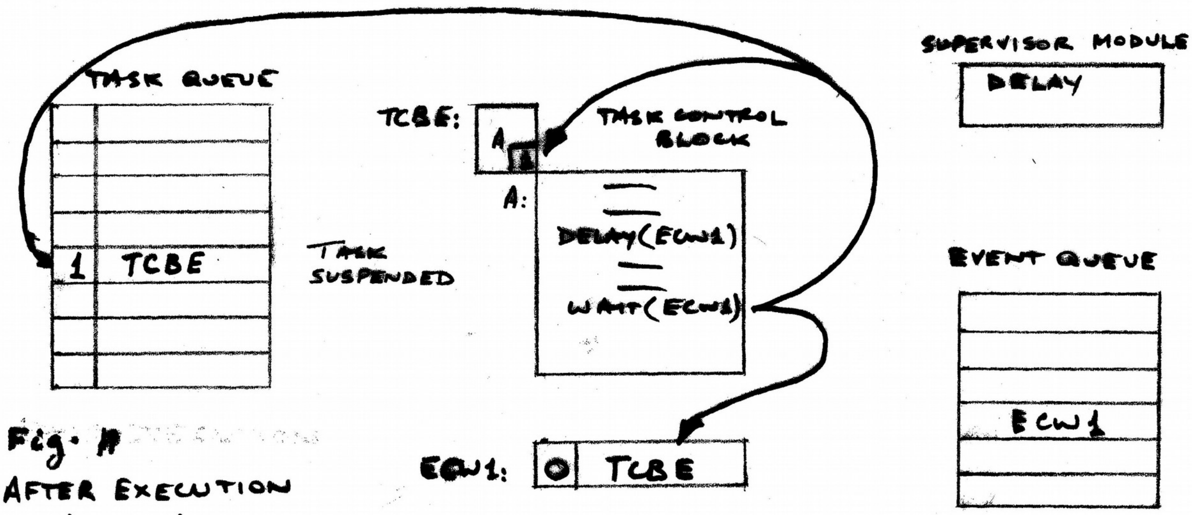


Fig. 11
AFTER EXECUTION
OF 'WAIT'

Figure 12 shows the situation at the time the Real Time clock has advanced to the time specified in the DELAY call, when the input/output supervisor performs the POST function. The following then takes place:

- The event control word is located from the address ECWA in the queue element in the input/output supervisor queue, and the completion bit of the event control word is set to 1.
- The wait indicator (Bit 1-15) in the event control word is tested to see if a task is waiting, in this case, it is, so the task control block wait count is decremented by 1.
- The wait count in the task control block is now 0, so the task is placed in the ready condition, eligible to compete on a priority basis for CPU time. As soon as there are no higher priority ready tasks, execution continues.

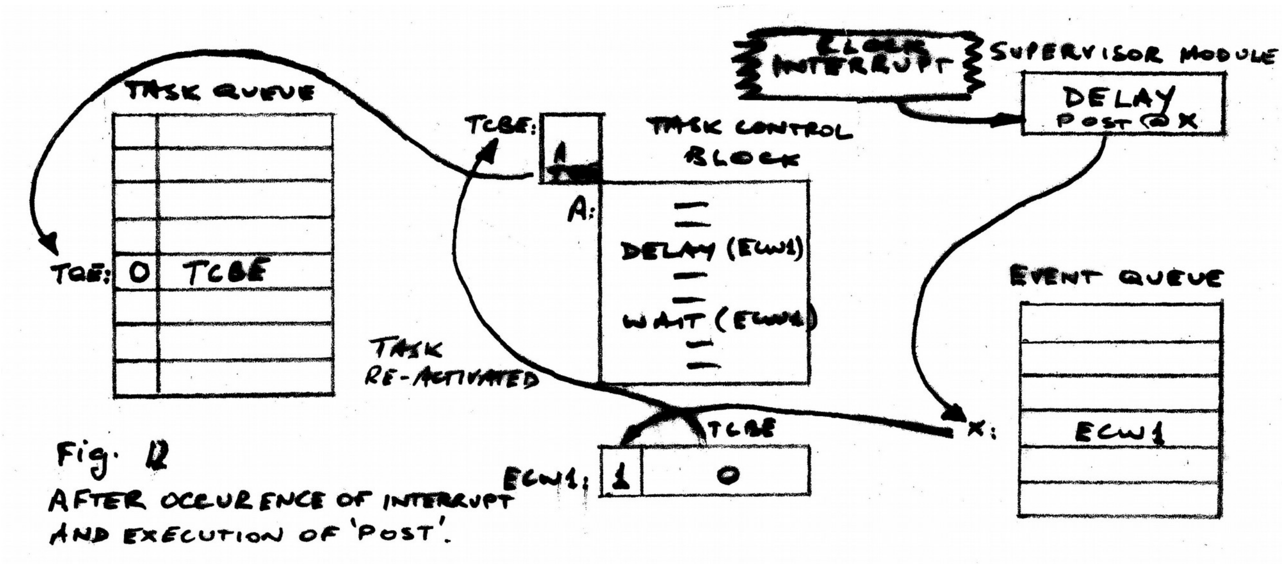


Fig. 12
AFTER OCCURENCE OF INTERRUPT AND EXECUTION OF 'POST'.

In the preceding example, the program reached the WAIT meta-instruction before the requested input/output operation was completed. If the input/output operation had been completed first, the completion bit would have been set and the program would have proceeded without any interruption when it came to the WAIT meta-instruction.

Event synchronisation which employs the WAIT and POST functions, is used mainly in the management of external resources by the Operating System. When a task requests a system resource, an event control word associated with the task is placed on the appropriate resource queue. The Task may have to wait until the resource is available. When it is, the Supervisor notifies the task by posting. It is important to note that only one task can be waiting on a particular event (as characterised by a particular ECW) at the one time. The event is unique to the posting agency and the particular task waiting for the event.

1.2.14 Critical Sections

Another form of event synchronisation is required, which allows co-operating tasks to share certain resources. The resources that can only be shared in this way are called 'critical sections'.

This property applies to a large number of facilities. Their common characteristic is that only one task may use them at one time. One example is program segments whose execution must be completed once they are started before another task may start them. These are serially re-usable programs. Another example is a table of data common to two tasks and modified by both. Care must be taken that once a modification is started, it is completed before another task picks up the modified value. If access to such a table is not made a critical section and modification is not completed the unmodified value may be picked up in another task leading to erroneous computation.

If the programmer wants to control access to such a facility, he may create a queue of all tasks requiring access, and limit access to one task at a time. Such a control action is provided in the operating system by two meta-instructions. LOWER and RAISE. These instructions operate on data items called SEMAPHORES.

The operations LOWER and RAISE are equivalent to the 'P' and 'V' operations defined by Dijkstra[5]. The mnemonic origin of the names 'P' and 'V' is obscure and the author found it so difficult to use these names that the names LOWER and RAISE were adopted. Since the origin of semaphores must have been taken from the context of mechanical railway signals, which have the function of excluding more than one train (task) from a critical section, the names LOWER and RAISE seem appropriate.

Dijkstra has thoroughly investigated the problems arising from access to common variables[5]. This is a short summary of the semaphore operations given by Wirth[1].

> *It is postulated that LOWER and RAISE be the only operations applicable to variables designated as semaphores. The observation of this postulate is crucial to the correct operation of tasks and may only be disregarded during the initialisation phase of a task. The operation*
>
> `RAISE(S)`
>
> *increments the value of the semaphore S by 1. The operation*
>
> `LOWER(S)`
>
> *can only be performed when the value of S is positive; then it is decremented by 1. From this it follows that LOWER may cause a delay of program execution until another task performs a RAISE operation on the semaphore S. Thereby a synchronisation of the two tasks is obtained.*

If semaphores are restricted to assume only the values 0 and 1, then the operators LOWER and RAISE correspond respectively to the operators LOCK and UNLOCK described by Dennis and Van Horn[14] , the TSL instruction of Lampson[6] or DEQ and ENQ in OS/360[7].

A critical section in OSCAR is coded as follows:

```
<statements before critical section>

LOWER
<SEMAPHORE ADDRESS>

<tatements in the critical section>

RAISE
<SEMAPHORE ADDRESS>

<tatements after the critical section>
```

The semaphore used to create a critical section must be a binary semaphore. The use of the general semaphore which may have other values apart from 0 and 1 is very powerful and an example may be found in the implementation of Double Ended Queues. (Section 2.3.2).

Many of the latest systems such as the VENUS operating system[13] (See also section 3.3) use semaphores for jobs for which Event Control Words are really more suitable. In particular the logical power of Event Control Words cannot be matched by Semaphores whereas the queueing action of semaphores is not available with Event Control Words. The action of Event Control Words and Semaphores is sufficiently different so that the inclusion of both in an operating system is justified. 0S/360 uses Event Control Words and Binary Semaphores. This is the only other system which uses two sets of synchronising primitives.

2 REAL TIME OPERATING SYSTEM 'OSCAR'

OSCAR is a versatile multiprogramming system which extends the hardware of the NOVA family of computers to give system programmers a flexible environment for implementing real time systems. OSCAR is a highly modular system with a hierarchical structure which allows users to access the system at a number of different levels. It is a fast system. Interrupt handling, task swapping and execution of the simple synchronising primitive is carried out in an efficient manner. High speed systems may be built using only the lower level functions. This requires more programming effort but has the advantage of producing very compact and fast systems. Otherwise formatted output and buffering is available at the cost of a bigger and slower system.

All functions are written as independent relocatable modules with global symbolic references. These are provided on a relocatable library which is compatible with the Relocatable Loader or Linker. Only modules which are required are actually loaded.

2.1 System Hierarchy

Levels of abstraction is a concept first described by Dijkstra[2]. A number of functions are loosely associated with a level. The concept of a level allows the programmer to use a number of functions at one level without being concerned about the operations at lower levels, This provides a way of thinking about a design which is clear and precise[3]. Function modules at a higher level use functions at lower levels. If all functions are specified in terms of the operations at a lower level and tested against these specifications, and if they are used correctly in the higher level functions, their use in these functions need not be tested. They can be assumed to be working correctly. This approach has made the testing of OSCAR very simple, because each function is conceptually simple and need only be tested for the small number of cases the specification allows for. If a function is more complex it is coded in terms of lower level functions which are tested independently.

An example of functions at a given level which are well specified and whose correct operation is generally accepted are the hardware operations executed by the central processor. These have been included in the levels of abstraction which shows immediately that the hardware software boundary is quite flexible. For instance floating point instructions are often implemented in hardware. The synchronising primitives in OSCAR could be implemented in Hardware. The specification and writing of the software functions has been carried out with the same care that would normally be exercised in designing a hardware facility. This is desirable for two reasons.

1. The specification must be sound to achieve correct operation. The aim of providing these functions is to give users the tools to design working systems.
2. The acceptance of a system such as OSCAR will be inversely proportional to the amount of software maintenance it requires. There is also a heavy premium on reliable operation in real-time systems.

The OSCAR functions which have been implemented to date are listed here according to levels.

2.1.1 LEVEL 0

- The Hardware Instruction Set of the Processor
- The Floating Point Instruction Set
- The Peripheral Device Interfaces

The Instruction set of the Nova is used without modification. This system runs on all Nova families of computers. The Software Floating Point package looks to the users at higher levels just like a second processor except for execution speed. The design of Device Interfaces for special devices must often be carried out by users and then the interpretation of I/O instructions for such devices depends on the design chosen. Luckily the Nova has a standard Interface design and an I/O instruction set which allows the implementation of uniform designs for a large variety of devices.

A Version of OSCAR for the PDP-8 family of computers has been partly written but not tested. The concepts of the rest of the levels are machine independent.

2.1.2 LEVEL 1

- Interrupt Handler
- Task Scheduler

This is the Operating System Nucleus. Above this level the system can be thought of as a number of Virtual processors, each having the facilities of the hardware CPU, its registers and a number of private memory locations and the Floating Point processor as an option. Interrupts are transparent to Virtual processors, just as Data Channel cycle stealing is transparent to the Hardware processor. There is as yet no means of communicating between Virtual processors and external devices.

2.1.3 LEVEL 2

- .SVC Supervisor Call
- .EXIT Exit from a Supervisor Module
- .WAIT Wait for one event
- .MWAIT Wait for a number of events
- .POST Post the occurrence of an event in a task
- .POSTI Post the occurrence of an event in an interrupt handler
- LOWER Lower a semaphore
- RAISE Raise a semaphore

These functions are the means of communication between the User who has a virtual processor (Task) and the Operating System Nucleus. The Supervisor Call allows access to System modules which are written as Tasks at higher levels. The .SVC simply provides a function which sets up the linkage between these Tasks and the User. Supervisor Task Modules have characteristics which are reminiscent of external devices. They may be started by a .SVC and they will then execute as a parallel and independent task with a separate virtual processor from the one making the call.

The other functions at this level implement the synchronising functions for Event Control Words and Semaphores described previously.

2.1.4 LEVEL 3

- Device Interrupt Service Routines
- Device Drivers
- Re-entrant Supervisor Subroutines

- .DQIN Initialise Double Ended Queue (DEQ)
- .LPUT, .RPUT Put a cell on one end of a DEQ
- .LGET,.RGET GET a cell from one end of a DEQ
- .FREQ Compute the frequency of a pulsed signal
- FLS Re-entrant interrupt service for pulsed signals
- MPY, MPYØ Unsigned Multiply
- DVD Unsigned Divide
- TIM Read elapsed time in clock increments
- FLOM Read elapsed flow in flow meter increments

These functions provide a number of services which are frequently required. They rely heavily on Level 2 and lower functions. For different applications users may develop alternative routines which will operate at this level.

Device Interrupt Service Routines and Device Drivers are usually written together. They bear a similar relationship to each other as the Data Channel Hardware of a computer and the Central Processor. They share common memory registers and their timing is interleaved in a predictable way. Device Drivers are part of the virtual Processor environment. Whereas Device Interrupt Service routines are outside this environment. But Virtual Processors may WAIT for events which are first identified in Device Interrupt Service routines. .POSTI is used to post such events. This call may not be used in a task. Device Interrupt Service Routines cannot be suspended and no calls which may have suspension as a result can be executed in them. Re-entrant Supervisor subroutines may be called from any Task. The specification of a particular routine may require the setting up of a special data area which is used by the subroutine. Such areas must be set up for each Task using the subroutine.

2.1.5 LEVEL 4

Re-entrant Supervisor Programs:

- CELLO Buffered Cell Output Program
- CES Counted Events Scheduler

These are the program part of tasks which may be implemented by providing one or more task control blocks which specify the starting point of one of these program as their initial starting point. Each task must also provide a work area whose address is part of the initialisation constants for each task.

2.1.6 LEVEL 5

Supervisor Tasks:

- TTODQ Teletype buffered output task
- INODQ 2nd Teletype (Infoton) buffered output task
- DELAY (DELEX) Schedule an event a given number of real time clock increments in the future. When the event occurs either POST it (DELAY) or execute a subroutine (DELEX)
- FLOW (FLOX) Schedule an event a given number of flow meter pulses in the future.

Planned Supervisor tasks which have not yet been implemented are:

- OPEN — Link a file or device to an input or output queue
- CLOSE — Release a file or device from its queue

These are the actual tasks whose programs are provided at Level 4. The buffered output tasks are accessed by users through Double ended queues, while the Event Schedulers are activated by a Supervisor Call.

2.1.7 LEVEL 6

Higher Level Language Interpreters

BASIC

This facility has not yet been implemented but much thought has been given to this extension of OSCAR. The plan is to implement the BASIC language in this way and to extend its instruction set to include the synchronising operations. Calls to assembly language routines will be included and all Input/Output will be carried out via the appropriate OSCAR facilities. The interpreter will be written to be re-entrant in the OSCAR environment. Interrupt service will be carried out at the appropriate OSCAR level and user written Device Service Routines will be allowed. A BASIC Task will be distinguished from other Tasks only by the fact that the Program Counter (PC) of that Task will be pointing to the code of the BASIC Interpreter. A Task may change from BASIC to assembly language programs simply by executing a call instruction.

To implement this facility a dynamic task structure with user defined priority would be appropriate. The Editing and incremental compiling facility of BASIC would exist as one task with a given priority. The RUN command would be extended to

'RUN <line number 1>, <line number 2>, <priority>'.

This would create a task which would start execution in the BASIC interpreter at line number 1 and take its Data from the first DATA statement after line number 2. The Keyboard would still respond to command input and a second Task could be started at the same line number or a different line number by the extended RUN command. Also RUN could be used as a programmed command. This would then be the same as the FORK instruction of Conway[8].

One difficulty which must be overcome is the sharing of the console between the Editing Task and the Running Task(s). This problem has been successfully solved in the implementation of Debug Task which will be described later. The output to and the input from the console is transmitted in lines via lower level function. In each Task a common semaphore ensures mutual exclusion of the console. If a BASIC Task is running and printing some values, the programmer can break in on this output by typing a special attention character - 'Escape' would be a logical choice. The Task will then complete the current output which RAISES the common semaphore and then continues running. In the meantime the BASIC Edit Task has obtained the console and Program modification may proceed. The Running Task(s) will proceed until the next output statement to the console where the Task will be suspended on the common semaphore. A suitable Proceed command or character will LOWER the common semaphore and allow the suspended Task to proceed.

Sharing of the console between a number of running Tasks must be organised by the user. This will bring out the full power as well as the difficulties of parallel processing.

A REAL TIME BASIC facility should provide a worthwhile extension of the computer for implementing On-Line systems. Many of these systems are presently being implemented with BASIC in its Uni programming form.

The same system could also be used as a multi-user BASIC facility if several consoles are available. In this case a number of consoles call up BASIC from a system Monitor. Each console will communicate with the Edit package of BASIC program which it has caused to run. Communication between consoles via BASIC would be possible.

FORTRAN and ALGOL

Fortran and Algol compilers are available for the Nova family of computers. The job of re-writing the Run-time Library to fit in a Task Structure would be quite large. Otherwise, there is no reason for not incorporating the OSCAR functions in these languages.

PL/I

PL/I already has a parallel processing capability. This very similar to the OSCAR structure and OSCAR could probably be adapted to implement the PL/I parallel operations. At present there is a compiler for PL/I which produces object code for Nova's, but which must be run on a larger computer.

2.1.8 LEVEL 7

The Keyboard Monitor

This facility has again not been implemented, but it forms a logical extension to the OSCAR system which has been planned and which would extend the system in the direction of a general purpose computer facility or time-sharing facility. This may sound ambitious, but the author feels that the functions of OSCAR are powerful enough to allow the implementation of a very versatile Multi-programming Disk Operating System, which could be accessed from a number of consoles. Currently such systems only allow Multi User Basic from a number of consoles. At best a number of User written, interrupt driven assembly language programs could be run in parallel with BASIC. There are no facilities on any mini-computer for simultaneous time-shared usage of a Disk Operating System Monitor, the higher level language compilers, the assembler and the text editor. Implementation of such a system would require the re-writing of all these programs in re-entrant form which is a formidable obstacle. But the generous provision for private register in OSCAR and the provision of tested synchronising facilities should make this job much easier than if the planning of such a system were to be started from scratch.

The Disk Operating System (DOS) for the Nova family of computers has a very good Keyboard Monitor which allows simple yet effective communication between the Operator and the System. This system has a number of implementation weaknesses which can be sheeted home to the lack of synchronising operations. The routines which fill and empty buffers on the interface between programs and interrupt service routines fail if a device empties a buffer too quickly. This was experienced when a fast Line Printer with a 132 character hardware buffer was installed. The sequence of characters in the output became mixed up because of lack of communication between the interrupt handler which emptied the buffer and the device driver which output the remaining characters.

Such a system would be very sought after for medium sized time sharing systems. Implementation would probably be difficult without some form of hardware protection. Without it system integrity could not be guaranteed to users if other users ran their own assembly language programs. Such a system would also benefit greatly from memory paging hardware. This would allow the implementation of a virtual memory structure. The present memory allocation and protection hardware for the Novas would do the job, and the

Nova on which the OSCAR system was implemented has this facility. So far it has not been used because the computers for which real-time control systems were developed are without this extra hardware.

2.2 Interrupt Handler

Interrupts on the Nova store the Program Counter in Location 0, turn interrupt off and jump to the interrupt handler whose address must be at location 1. The OSCAR interrupt handler only saves Accumulator 3 in the first word of the currently active TCB. In another 2 instructions the device code is determined and a transfer is made to a device interrupt service routine via a transfer. table on page zero. This transfer table must be set up by the user. All devices which can possibly interrupt must be represented on this table. Devices which are not required may point to a dummy device service routine which clears the offending DONE flag and returns from interrupt. A typical Device Service Table is shown in the listing in the Appendix under the title TS1. TS1 also defines a number of page zero constants and address pointers. These should-be retained.

2.2.1 Return from Interrupt

This 3 statement program is the symmetrical dual of the Interrupt Handler. It is entered from a Device Service Routine, restores Accumulator 3, turns interrupt on and jumps via location 0 to the interrupted program. The concept of symmetry has been used a great deal in coding OSCAR. Figure 13 which is a flowchart of the Task Scheduler and Interrupt handler shows how this symmetry fulfils the basic requirements of the problem of context switching.

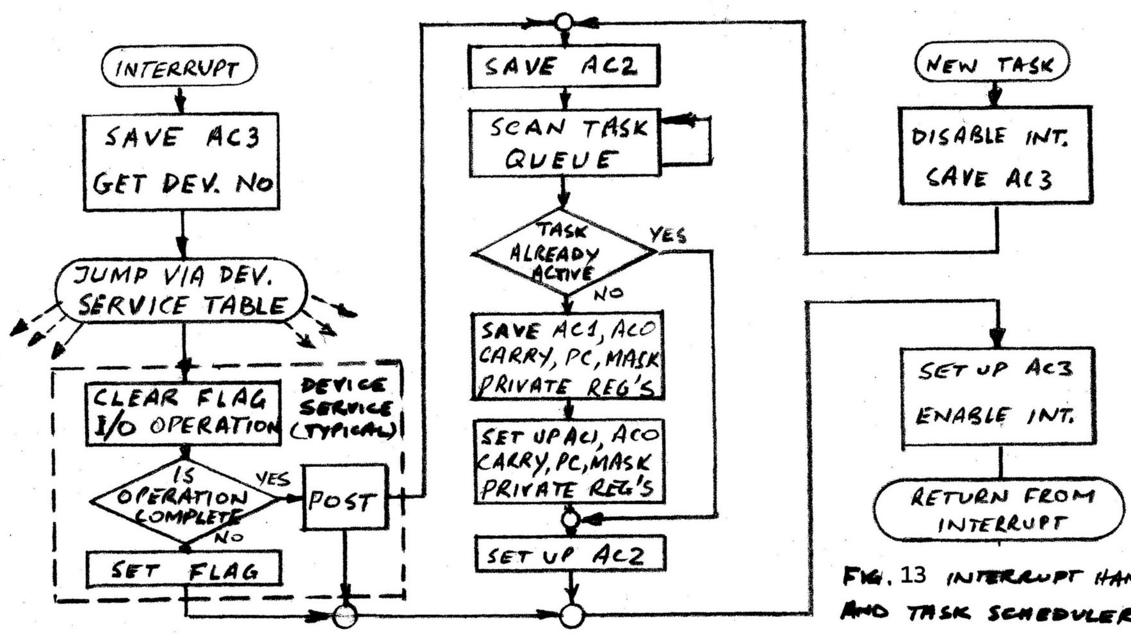


FIG. 13 INTERRUPT HANDLER AND TASK SCHEDULER

2.2.2 Task Scheduler

The Task scheduler has a number of entry points. These are only entered from other system modules, never by users. The Scheduler saves two registers in the current active TCB and then carries out a scan of the Task queue for the highest priority ready task. The structure of the task queue allows a fast scan to determine the highest priority task quickly. Each task queue entry contains the TCB address in bits 1 to 15 and a Ready-'bit in bit 0. If the Ready-bit is 1 the task is suspended, if the Ready-bit is 0 the task is ready and may be made active. In the task scan the first entry where the Ready-bit is 0 is made active. This is done by saving all relevant register of the current task in its TCB, and then setting up the registers of the TCB whose address has been found in the task queue.

A test is carried out to check that the current active task is not the highest priority task found in the task queue. If it is, time is saved by bypassing most of the task swapping code and simply restoring those registers which had already been saved. Again symmetry was exploited to do this. For more details see listing of program TS2 in the Appendix.

2.3 OSCAR Meta-Instructions

Communication between user programs and OSCAR is carried out through 'meta instructions' which are subroutine calls followed by parameters. Because of the lack of a macro-assembler, these meta-instructions are declared in their library modules as entry points (.ENT) and they must be declared as externals (.EXTN) in user modules. Although most of the meta-instructions are subroutine calls, they have been equivalenced to a single word usually beginning with a full-stop.

e.g.

```
.ENT .WAIT
.WAIT = JSR @ WAIT
```

2.3.1 Address Parameters

Parameters of OSCAR meta-instructions are frequently addresses or pointers to a certain memory location. In most instances the convention has been adopted that if bit 0 of the address parameter is zero the address defined by bits 1 to 15 is the required pointer. If on the other hand, bit 0 is a one the indirect convention of the Nova computer applies and bit 1 to 15 defines the address in which the pointer is to be found. Usually this indirect chain is emulated indefinitely until a zero is found in a bit 0. In one instance this does not apply (See note in DELAY call). Care must also be taken if auto-increment or decrement registers are used as indirect addresses. Since these indirect chains are usually emulated by software, no auto-indexing takes place. In some instances the auto-indexing will apply and the rules of each routine must be strictly observed. This problem can be serious in OSCAR because half the 'private' registers are auto-indexing. These make it possible to use the system meta-instructions re-entrantly. If the indirect feature applies to a particular parameter this is indicated in the calling sequence.

2.3.2 Supervisor Call

```
.SVC
<Supervisor Module Name>
<Other Parameters>
<Next Statement>
```

This is a linkage operation which causes scheduling of Supervisor Modules which are independent tasks rather than subroutines. The Supervisor Module Name is declared as an entry point in the module and is the address of the TCB for the task. Some supervisor modules have dual functions in which case bit 0 of the Supervisor Module Name is used to mark the second function. Users do not have to worry about this. Two separate names are declared in the Module in such cases. The other parameters depend on the individual supervisor module called. Details are found in the calling sequence of each module.

The action of the .SVC is analogous to the JSR operation except they provide a call from one task to another task, rather than from one program to another program. In the spirit of this, the main calling parameter is a TCB address, and the action of the .SVC is to make this task active. The .SVC call also stores the TCB address of the calling task in AC3 of the called task. This allows the supervisor module full access to all the accumulators and private registers of the calling task at the instant it made the call. Since the .SVC is initially also a Jump to Subroutine the value of AC3 in the calling task is a pointer to the word after the .SVC. Thus the called task has access to all the parameters following the .SVC call just like any other subroutine.

NOTE :The task priority of the supervisor module must be higher than the priority of the calling task. Otherwise the calling task will continue execution before the supervisor module, which is not intended. For this reason the 5 highest priority task names TCBA-TCBE have been reserved for supervisor use.

The .SVC details are only important if users write their own task modules which are to be accessed by the .SVC call. Also the EXIT call should never be used except in such a module.

2.3.3 Exit from Supervisor

```
.EXIT
<NEXT STATEMENT>
```

This call suspends a task unconditionally. The effect is to schedule the next ready task. This is usually the task which previously made the .SVC call to the module containing the .EXIT. It thus constitutes a return from the supervisor module to the next statement after the Supervisor call in the task which made the call. Data may have been passed through the calling tasks TCB or common locations. The latter mode is not re-entrant whereas the former is. The statement after the .EXIT call is the statement executed when the supervisor module is next activated by a .SVC. Thus the .SVC-.EXIT mechanism may also be used to implement co-routines.

2.3.4 Post an Event

```
.POST
<ECW ADDRESS> or .@<POINTER TO ECW ADDRESS>
<NEXT STATEMENT>

.POSTE                      ; ACO CONTAINS 15 BIT MESSAGE
<ECW ADDRESS> or @<POINTER TO ECW ADDRESS>
<NEXT STATEMENT>

.POSTI                      ; USE ONLY IN INTERRUPT SERVICE
<ECW ADDRESS> or @<POINTER TO ECW ADDRESS>
```

The execution of .POST or its companion instructions .POSTE or .POSTI marks the occurrence of a particular event in real time. The only parameter is the address of an Event Control Word or a pointer to such an address. The Event Control Word, which must be used as the parameter in a WAIT operation in another task to establish a communication is tested and modified by the POST operation. Because Interrupts are disabled this becomes an indivisible operation in the task environment.

All POST operations set bit 0 of the ECW which is the completion bit. Additionally .POST and .POSTI clear bit 1-15 of the ECW. .POSTE is used to also transmit a 15 bit message to the task waiting for the event (usually an error message). For this purpose bits 1-15 in ACO are stored in bit 1-15 of the ECW.

If bit 1-15 of the ECW were non-zero before modification these bits contain the address of the TCB of the task waiting for this event. This TCB has a one word register called the Wait Count register whose arithmetic value is the number of events the task is waiting for before coming to the Ready state. If a task is waiting for the event, the Wait count in its TCB is decremented and if it becomes zero, the task is taken from the Suspended to the Ready state and the task scheduler is entered. If the task is not waiting for an event or the Wait Count did not become zero, a normal subroutine return is made in the case of .POST or .POSTE. These two meta-instructions must always be used in a program which is executed on behalf of a task. The meta-instruction .POSTI must always be used in an Interrupt Service Routine while Interrupt is disabled. .POSTI returns control to the task which was interrupted by the device whose service routine contains the .POSTI call. This return may be delayed if the execution of .POSTI caused a higher priority task than the one interrupted to be made Ready. In this case the task scheduler will schedule this new task first and the interrupted task will be re-scheduled later.

A similar sequence applies in the case of .POST or .POSTE. In this case if the execution of these meta-instructions causes scheduling of a higher priority task than the one making the call .POST or .POSTE, then return to the calling task is delayed until the higher priority task suspends itself.

Multiple posting through the same ECW is allowed. Only the first posting has any effect, all subsequent postings are ignored. This means that if no task is waiting for an event that event can still occur a number of times without any ill effect. It is possible to re-write the .POSTE routine to transmit the message in ACO from the last posting rather than the first. In any case it is sometimes convenient to write a special POST macro as in-line code for maximum speed. This has been done in the DELAY module (see Listing in the Appendix).

2.3.5 Wait for a Single ' Event

```
.WAIT
<ECW ADDRESS> or @<POINTER TO ECW ADDRESS>
SUBC  3,3
STA   3,@.-'2
<NEXT STATEMENT>
```

This meta-instruction is one implementation of the WAIT operation. It suspends a task if some event which a task wants to wait for at this point in its sequence has not yet occurred. This is equivalent to saying that the ECW which is a parameter of the .WAIT meta-instruction has not yet been posted. The Wait operation tests and modifies the Event Control Word as an indivisible operation as did the Post operation. If the completion bit (bit 0) is already set (the ECW has already been posted) the next statement is executed immediately. If the completion bit is not set, the Wait Count in the TCB of the current task is set to +1 (waiting for one event) and the address of the TCB of the current task is stored in bit 1-15 of the ECW. Bit 0 is left cleared.

Also the Ready bit in the Task Queue entry for the current task is set to one. This puts the current task in the suspended state. The task scheduler is then entered to schedule the next task.

After the .WAIT call the Event Control word must be cleared before execution of the operation which will initiate the next event which finally posts the ECW. It is a good practice to do it immediately after the .WAIT call and therefore a clearing sequence has been included in the calling sequence.

Event Control Words must also be cleared during the initialisation phase of a task. Otherwise re-starting of a system is impossible. Sometimes it is advantageous to set the completion bit initially. This avoids initiating the operation which posts the event during initialisation.

2.3.6 Wait for Multiple Events

```
.MWAIT
<ADDRESS OF ECW1> or @<......>
<ADDRESS OF ECW2> or @<......>
<ADDRESS OF ECWn> or @<......>
-<m>
SUBC    3,3
STA     3,@.-<n>-2            ; CLEAR ECW1
STA     3,@.-<n>-2            ; CLEAR ECW2
```

```
STA     3,@.-<n>-2              ; CLEAR ECWn
<NEXT STATEMENT>
```

This meta-instructions endows the Wait operation with a certain amount of logical power. It is to be interpreted as:

"Wait for `m` out of the `n`. events listed".

The logic used is commonly known as majority logic. Two special cases exist which are most frequently used:

- A .MWAIT call with `m=n`.

  This produces a logical AND. It is interpreted as:

  "Suspend the current task unless or until all the events listed have occurred".

- A .MWAIT call with `m=1`.

  This produces a logical OR. It is interpreted as:

  "Suspend the current task unless or until one of the events listed has occurred".

The .MWAIT operation is not as efficient as the .WAIT operation for the special case of waiting for one event. On the other hand the logical AND of a number of events can be implemented by a sequence of .WAIT calls. This is not as efficient in space or in speed as the equivalent .MWAIT call. The logical OR case can only be implemented by the .MWAIT call.

After the .MWAIT call Event Control Words must be cleared before any other call is made which may involve suspension. There is an additional reason in the .MWAIT case. Some of the ECW's which have not yet been posted will contain the TCB address of the current task. If another WAIT operation is carried out on a different ECW, posting of the previously uncleared ECW may cause resumption of the task. Thus the wrong event would make the task Ready. Such a Wait operation may occur in a subroutine. Therefore such subroutines are included in the category of calls which may cause suspension. Therefore a clearing program has been written into the calling sequence. In the case of the .MWAIT call implementing the OR case it is frequently required that a test is made of which of the possible events has caused resumption of the task. This test must be made before the ECW's are cleared. The clearing program must then be modified but must not be forgotten.

Caution 1: `m` should not be greater than `n`, the number of ECW addresses. If it is, the task will be permanently suspended.

Caution 2: `m` should never be greater than `63`. If it is it will be interpreted as an address pointer with unpredictable results.

Caution 3: if pointers are used to indirect addresses using the indirect conventions these pointers should never have addresses which are greater than $2^{15} - 64$. If they are they will be interpreted as `-m`. This is not difficult since this is the region reserved for the binary loader in a 32K Nova computer.

2.3.7 Semaphores

Semaphores must be defined as a 2 Word Block.

e.g. `SEM1:.BLK 2`

The first word is the Semaphore Counter. The second word is the Semaphore link. It contains zero when no task is suspended on the semaphore or the TCB address of the first task suspended on the semaphore. During task initialisation semaphores must be initialised correctly. They must be initialised in the highest priority task using the semaphore. The Semaphore Counter should contain the number of LOWER operations which are to be allowed before the Semaphore suspends a task. This value is 1 for a Binary Semaphore, which is initially raised. The value is 0 for a Semaphore which is initially lowered. The Semaphore Counter should never be initialised to a negative value. The Semaphore Link should always be initialised to zero.

Apart from initialisation, Semaphores should only be operated on by the operations LOWER and RAISE. These are indivisible operations which work across task boundaries. Any other sort of test on the value of the Semaphore Counter may no longer be valid by the time the test results become known.

2.3.8 Lower a Semaphore

```
LOWER
<ADDRESS OF SEN.> or @<POINTER.....[1]>
<NEXT STATEMENT>
```

The Semaphore counter is decremented. If the counter is then positive or zero the next statement is executed immediately. Otherwise the current task is suspended until a RAISE operation on the same Semaphore makes the task Ready. When the task is suspended its TCB address is stored in the Semaphore link or in the Word Count register of the last TCB in a chain of TCB's if this was not the first task suspended on the particular Semaphore.

2.3.9 Raise a Semaphore

```
RAISE
<ADDRESS OF SEN.> or @<POINTER.....[1]>
<NEXT STATEMENT>
```

The Semaphore Counter is incremented. If the counter is then positive (not zero) the next statement is executed immediately. Otherwise a task chained to the Semaphore link is made Ready and the task scheduler is entered. Which task is executed next depends on the priority of the task which has just been made Ready.

Tasks are made Ready by RAISE operations in the order in which they were suspended. Thus no task can be suspended indefinitely at the expense of other tasks.

2.4 Simple Drivers and Interrupt Handlers

```
.GET
<NEXT STATEMENT>
```

Get a character from the Teletype Keyboard in ACØ. The Event Control word used is TTIEC.

```
.READ
<NEXT STATEMENT>
```

---

1 Do not use Auto-Indexing Registers for Pointers.

Get a character from the High Speed Paper Tape Reader. The Event Control Word used is PTREC.

```
.PUT
<NEXT STATEMENT>
```

Print a character passed in ACØ on the Teletype. The Event Control Word used is TTOE1.

```
.PUNCH
<NEXT STATEMENT>
```

Punch a character passed in ACØ on the High Speed Paper Tape Punch. The Event Control Word used is PTPE1.

These routines are all similar in structure. An interrupt from any of these devices will clear its flag and POST the Event Control Word mentioned. The Drivers WAIT on this Event Control Word on entry and then get or put the character on the device before returning.

2.4.1 Teletype Driver and Interrupt Handler

This Teletype Driver emulates a device with many more capabilities than the actual Teletype.

```
JSR @PUTB or JSR @PUTBI
<MAX NO OF BYTES IN BUFFER>
<NEXT STATEMENT>
```

AC2 must contain the word address of the first byte in the buffer

The input is passed to the driver as a byte string which must be stored in a buffer whose address is passed to the driver in AC2. This allows the routine to be used in re-entrant situations. The driver is not itself re-entrant, but its address may be stored in a private register and a re-entrant program may be shared by several tasks each of which communicate with a different driver. This is frequently required for Teletypes used as terminals.

The maximum number of bytes in the buffer is included in the call as a safety feature to prevent printout of characters which are not in the buffer. Normally a string is terminated by a null byte. The routine could easily be modified to also terminate a string by a Carriage Return or a Form Feed.

The following special character functions have been implemented.

ASCII CODE	**ACTION**
000	MARK LAST BYTE OF A STRING
001	INSERT CRLF
002	INSERT CRLF
004	SUBSTITUTE FOR TAB
005	ENQ' SUBSTITUTE FOR FORM FEED
010	CR ONLY
011	TAB TO THE NEXT COLUMN OF 8
012	LINE-FEED (THE FIRST LF AFTER CR IS IGNORED)

ASCII CODE	ACTION
014	FORM-FEED (COMPLETE THE CURRENT PAGE)
015	CARRIAGE-RETURN (INSERT LF)
017	SUBSTITUTE ‘)’
031	SUBSTITUTE SPACE
032	SUBSTITUTE ‘←’
034	SUBSTITUTE ‘↑’
035	SUBSTITUTE LF
037	SUBSTITUTE ‘(‘
177	RUB-OUT (IGNORED)

Any other control codes are not transmitted.

The Constants are correct for an Olivetti Terminal Type 308, adjusted for 80 Character lines. The page length is 60 lines with 6 extra lines to complete an 11" page.

To obtain a consistent page format all output to the Teletype must be channelled through this driver.

The driver when called by 'JSR @PUTB' returns to the calling program when the last character has been transmitted to the Teletype. This is 100 ms before the completion of printing of the last character which is the time when the buffer becomes free. Another call 'JSR @PUTB' may be made immediately because the first thing the driver does is wait for the completion of printing of the last character. Thus 100 ms are available to generate another buffer while maintaining the Teletype at its maximum speed. If this is not enough the driver may be called by 'JSR @PUTBI'. In this case the next statement is executed immediately the first character has been transmitted. Output and further computation may then proceed in parallel. Care must be taken not to disturb the buffer. To synchronise with the transmission of the last character wait for the event control word 'TTOE2' which is a global symbol.

2.4.2 Drivers and Interrupt Handlers for other Terminals

A Driver for an Infoton display and Keyboard has been written.

```
JSR @INDB or JSR @INDBI
<MAX NO. OF BYTES IN BUFFER>
<NEXT STATEMENT>
```

AC2 must contain the word address of the first byte in the buffer

This is very similar to the Teletype Driver. The only difference is that it emulates a few extra character functions which are peculiar to a display.

ASCII CODE	ACTION
000	MARK LAST BYTE OF A STRING

ASCII CODE	ACTION
001	SAVE THE POSITION OF THE CURSOR
002	RESTORE THE CURSOR TO THE POSITION LAST SAVED
010	HOME THE CURSOR WITHOUT ERASING
011	TAB TO THE NEXT COLUMN OF 8
012	LINE-FEED (THE FIRST LF AFTER CR IS IGNORED)
014	ERASE SCREEN AND HOME CURSOR
015	CARRIAGE-RETURN
017	BLINK-OFF
031	CURSOR RIGHT
032	CURSOR LEFT
034	CURSOR UP
035	CURSOR DOWN
037	BLINK-ON
177	RUB-OUT (ERASE CHAR. ON THE LEFT)

Any other Control Codes are not transmitted.

The Constants are correct for an Infoton Display with 20 lines, 64 characters per line and set to 'Roll' mode. A Cursor Count is maintained which follows the actual Cursor on the screen. The Cursor save and restore feature make use of this count.

2.4.3 Data Communications Multiplexor Driver

The Asynchronous Data Communication Multiplexor (DCM) type 4026 for the NOVA can control the transmission of asynchronous serial data on 16 output lines and can receive asynchronous serial data simultaneously over 16 input lines. The device requires periodic changes in the contents of a 16 bit output register in which each bit is connected to a separate output channel. Thus successive changes in the register contents produce bit-by-bit serial transmission over the channels. Data is received by periodically sampling the 16 input lines to pick up the bit-by-bit serial input. The sampling rate must be greater than the bit rate to allow for degradation of the signal. Satisfactory operation is achieved by sampling the input 5 times per bit time. With such a scheme a transient that in less than 3 sample times is not mistaken for a start pulse.

Because the sampling rate is 550 Hz for 110 Baud Teletypes a driver for such a device can easily use up an untoward amount of computer time. The Data Communications Multiplexor Handler program which is supplied by the manufacturers executes 343 instructions for every sampling interrupt. This means a 39% occupancy on a Nova computer. The Drivers and Interrupt Handler written to run under OSCAR can handle each sample interrupt in an average of 35 instructions. This lowers the occupancy to 4% for 16 terminals which is 0.25% per terminal. These figures are 3 to 5 times better for both handlers with the new range of Nova computers. Manufacture of this piece of hardware has ceased, probably because of the high occupancy associated with the standard handlers. The device has been replaced by a similar device which assembles full 8 bit characters by hardware.

The rest of this description should apply to both types of multiplexors.

There are 32 drivers one for each input line and one for each output line. Only those drivers which are actually required in an installation need be loaded. The DCM Interrupt handler posts an ECW associated with an input line each time a character from that line has been assembled. It posts another ECW associated with an output line each time a character has been transmitted on that line.

Each of the drivers wait for the particular ECW which is posted in the Interrupt Handler and then picks up the character from a one word Buffer for input or stores the character in a one word buffer for output. This is the simplest sort of driver, and to users its operation is identical to the driver for a conventional Teletype interface as in Section 2.4. User systems would probably be structured to be one task for a pair of drivers.

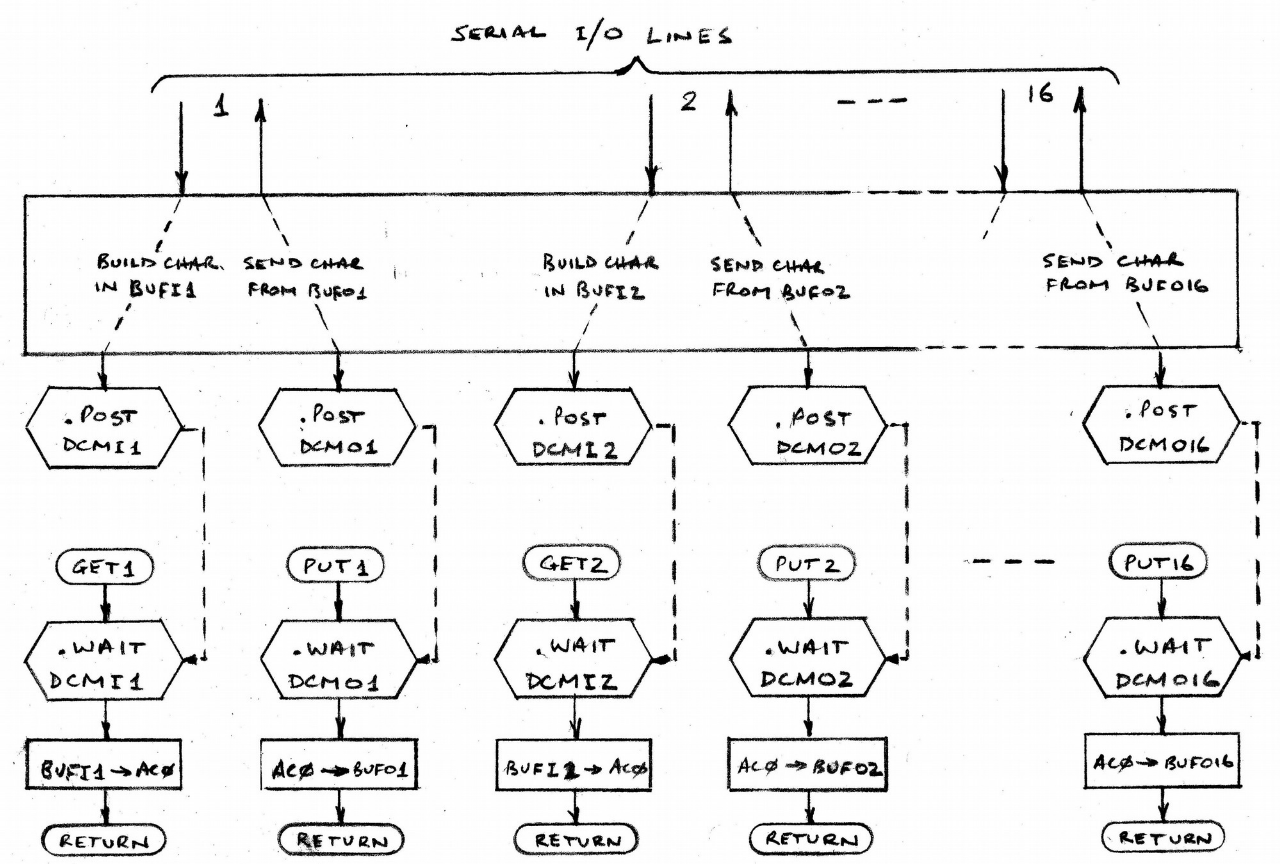


FIG. 14 DCM Interrupt Handler and Drivers

This approach contrasts strongly with the DCM Handler supplied with the machine. It has its own interrupt handler, and this is virtually the master program. Each time a character is built up it does a 'JSR' to a user supplied subroutine which must accept this character (or supply one in the case of printing). The user program is only allowed 1000 instructions times to do this. Otherwise the sequence will go astray. In other words the User program is a subroutine to the DCM handler. Contrast this with OSCAR where all functions are subroutines to the User. The worst that can happen if User programs are too slow is that a device driver may post more quickly than a task can accept events. In this case characters would be lost. But this is no different to operation of a single Teletype without interrupt. If a program cannot get around the loop in 100 ms characters will also be lost. Thus programs which run in conventional single terminal mode will run exactly the same under OSCAR through the DCM Driver.

More complex drivers along the lines of section could be written for this device. The Interrupt Handler would not be changed. It would be appropriate to write the driver re-entrantly.

2.5 Double Ended Queues

Dynamic buffering is carried out in OSCAR by the common list structure called *Double Ended Queues*. These are more useful than Single Queues because they may be accessed at both ends. Each Double Ended Queue (DEQ) consists of a control block and a variable number of cells. The number of cells may be zero in which case the DEQ is an empty DEQ. Each cell consists of two link words and a fixed number of words of storage which may be used as buffers. The address of a cell is the address of the first word of buffer storage. The Link words have a displacement of -1 and -2 with respect to the cell address. The Link words may be used as temporary storage registers while the cell is not on a DEQ. Once a cell has been put on a queue the Link Words will be overwritten.

The first two words of the Control Block (DQCB) and the two link words in each cell in the queue together form a circular linked list. The first word “L” points to the cell on the left. The second word "R" points to the cell on the right. The link words in the Control Block close the circle. Since the address of the Control Block is known, routines using the Control Block address as a parameter can manipulate cells immediately to the left and right of the control block.

DEQ's are operated on by five routines one to initialise a DEQ, two to get cells from the queue and two to put cells on the queue.

2.5.1 DEQ Initialisation Routine

```
.DQINI
<DQCB ADDRESS> or @<POINTER TO......>                   ; D
<CELL BYTE LENGTH> or @<POINTER TO ......>              ; L
<NUMBER OF CELLS> or @<POINTER TO......>                ; N
<ADDRESS OF FIRST CELL> of @<POINTER TO......>          ; S
<NEXT STATEMENT>
```

This routine initialises a DEQ Control Block and a set of cells. This is usually done in the initialisation phase of a task. Even if a DEQ is to be initially empty it is necessary to re-write the appropriate pointers when a system is re-started. As a general rule DEQ's are set up to be initially empty because they represent buffers for various facilities which are empty to start with. Only one DEQ is set up with cells and this is a source of cells for the system. This DEQ is set up by OSCAR and its DQCB has the label FREE which is defined as a global symbol. The number of cells (NC) and the cell length in bytes(CL) must be defined in TS1 which is normally assembled by users to set up their configuration of OSCAR. The labels NC and CL are entered as global symbols in TS1. If cells are going to be taken from FREE then the global symbol CL should be used as the 3rd parameter in .DQINI and as parameter of the calls 'JSR @ PUTB' and 'JSR @ INDB' discussed in Section 2.4.1 and 2.4.2

Different length cells can be handled by these routines in the one system, but only cells of the one length may be taken from or put on a particular DEQ. Since cells are normally taken from the FREE DEQ and put on a buffer DEQ and then put back on FREE, these must all have the same length cells.

NOTE 1: Cell length (L) must be specified in bytes in the 2nd parameter of the .DQINI call. It will be used to store the number of words in a cell in the last word of the DEQ Control Block. This constant does not include the two link words. Thus the actual space taken up by a cell is `(L + 5)/2`. This allows for the extra byte which is actually provided if `L` is odd.

NOTE 2 : The number of cells specified (`N`) is also the maximum number of cells allowed on the DEQ. If the address of the first cell (`S`) has the value '0', the DEQ is initialised to be empty, and '`N`' is used only to set the maximum number of cells on the DEQ.

NOTE 3 : The amount of storage which must be set aside for cells may be computed as follows. If '`N`' is the number of cells specified, '`S`' is the address of the first cell and '`L`' the byte length then the locations used go from

```
S to S + (N * ((L+5)/2) - 1)
```

The FREE DEQ which is set up by OSCAR is started at the first free memory location after OSCAR is loaded. Thus the OSCAR Library must be the last module during the relocatable linking or loading operation. Having predefined `L = CL` and `N = NC` in TS1 the last location used by the FREE DEQ will be

```
LAST = NREL + (NC * ((CL+5)/2) - 1)
```

Care must be taken that this does not interfere with the binary loader or at least does not reach past the available memory.

NOTE 4 : Even if a DEQ is to be initially empty it should be initialised with the .DQINI call so that the DEQ Control Block pointers are restored and the semaphores are reset when re-starting the system.

2.5.2 Get a Cell from a DEQ

```
.LGET or .RGET
<DEQCB ADDRESS> or @<POINTER......[2]>
<NEXT STATEMENT>
```

.LGET and .RGET will obtain the address of a cell from either the left or right of the DEQCB specified. The address of the cell is returned in AC2. The DEQ is re-linked to exclude the cell which has been taken out. A semaphore in the Control Block is LOWERed which counts the number of available cells in the DEQ and if an attempt is made to get a cell when the DEQ is empty, the task making the call is suspended. The task is re-activated when another task puts a cell on the DEQ which RAISEs the semaphore. That cell is then immediately available for the .LGET or .RGET call.

A second semaphore counts the number of cell spaces still available before reading the maximum number. This semaphore is RAISEd by .LGET or .RGET because these calls make another cell space on the DEQ available.

.LGET and .RGET also store the address of the cell returned in AC2 less 1 in private location 20. Thus location 20 can be used as an auto-incrementing pointer to the words in the cell. The word length of the cell is returned by both routines in location 30. The operation 'DSZ 30' can thus be used as a loop count when accessing words in the cell.

2 Auto-indexing registers may be used, they will not auto-index.

2.5.3 Put a Cell on a DEQ

```
.LPUT or .RPUT
<DEQCB ADDRESS> or @<POINTER......[2]>
<NEXT ADDRESS>
```

.LPUT and .RPUT insert cells into a DEQ. The address of the cell must be passed to a PUT routine in AC2. Since cells are usually taken from another DEQ with a GET operation which provides the address in AC2, the two operations are compatible. The new cell is linked into the DEQ on the side specified. The cell counting semaphore is RAISEd and if a task had previously been suspended because it tried to get a cell from where there were none it will now be re-activated. The second semaphore is LOWERed and if the present cell would make the number of cells on the queue exceed the maximum number specified at initialisation, the task making the PUT call will be suspended until a cell is taken from the DEQ by another task. This mechanism prevents all the cells from the FREE DEQ being taken by one task and put on one DEQ. This would prevent other tasks from getting cells.

2.6 Elapsed Time

```
. TIM
<NEXT STATEMENT>
```

A double precision counter is maintained by OSCAR. This counter is incremented every tick of the Real Time CLOCK. Because the interrupt service for each clock tick is only 20µs at most, a clock frequency of 1KHz is handled comfortably and the system is initialised to this value. If a different speed is required the value of RTCSP in program DELAY must be altered. If 16 bit accuracy only is required, the low order word TIME may be loaded directly. This word is on page zero and is entered as a global symbol. TIML should not be modified. If double precision is required the call .TIM will return the double precision time in ACO, AC1. This call should not be carried out when Interrupt is off.

If the time has been taken at two different points in a sequence the difference, either single or double precision will give the elapsed time between the two points in the sequence as long as the elapsed time does not exceed 2^{16} or 2^{32} respectively. A simple unsigned single precision or double precision subtraction is all that is required to obtain this difference.

This works even if the absolute value of the time for the first event is greater than the absolute time of the 2nd event. In this case overflow of the clock counter has occurred between the two events. Two's complement subtraction allows for this case.

2.7 Event Scheduling

Certain system resources may be required by more than one task. In this case queuing facilities must be provided with the routine servicing such a resource. Examples which come to mind are Read and Write requests from random access devices. These have been coded for OSCAR on an experimental basis but have not yet been incorporated in the system. A resource which is fundamental to real-time system and which has been provided with a queued service routine is the *Real Time Clock*. This routine may also be used by other devices which interrupt a computer at regular intervals and these interrupts mark the completion of a quantum of some physical measure. Devices which come to mind are displacement measuring equipment and liquid flow meters. Routines for the latter have actually been implemented for a liquid blending system described in Section 1.1.2.

To allow for this diversity of similar devices a re-entrant set of routines was written. Each physical device uses separate tasks to implement event scheduling for itself. The re-entrant programs have been called the *Counted Events Scheduler*. The action of this general program will be described by the particular implementation for the *Real Time Clock*. Implementations for other devices should be done after consulting the listing of the assembly DELAY which contains the Interrupt Service routine for the *Real Time Clock* and the Task Control Blocks for the associated tasks.

2.7.1 Time Scheduling

Enter an Event into the timed event queue.

```
.SVC
DELAY
<ECW ADDRESS> or @<POINTER......>
<DELAY> or @<ADDRESS CONTAINING DELAY>
<NEXT STATEMENT>
```

Timing starts immediately the call is made. <DELAY> must be given as an integral number of clock ticks from the time the call is made.

Any task including the one making the call (but only one) can wait on the completion of the delay which is accompanied by posting of the ECW whose address is passed in the call.

NOTE 1: If this call is repeated for the same ECW, the previous queue entry is deleted and the event will not be posted. Only the latest entry will be posted when the number of time ticks in the `<DELAY>` parameter have elapsed.

NOTE 2 : For both types of call the `<DELAY>` must be less than 2^{15} clock ticks if given directly in the call or less than 2^{16} clock ticks if pointed to by an address in the call.

NOTE 3 : The indirect chain for `<DELAY>` proceeds only 1 level whereas the chain for ECW addresses or subroutine addresses proceeds as long as @'s are encountered.

Enter a request for 'delayed execution' of a subroutine.

```
.SVC
DELEX
<ENTRY ADDRESS OF SUBROUTINE> or @<POINTER......>
<DELAY> or @<ADDRESS CONTAINING DELAY>
<NEXT STATEMENT>
```

This is an alternative of the first call which does not involve posting after completion of the delay. The request is entered into the timed event queue and control returns to the next statement immediately. When the delay time has expired the subroutine, whose entry point address is stored in the queue is executed at high priority by the supervisor.

NOTE 4: All entries for subroutine execution. are retained and finally executed even if other requests for the same subroutine are made before the first has occurred.

NOTE 2 & 3 of the previous section also apply.

Since the subroutines requested are executed by the supervisor at high priority these subroutines must satisfy a number of conditions. Otherwise the supervisor functioning will be impaired.

1. Routines should be as fast as possible.

2. Routines should contain no calls which could result in suspension.
3. All accumulators (including AC3) carry and private locations 6, 20 and 30 may be modified. Private locations 7, 21, 31, 40 and 41 must be preserved. Location 40 will contain the return point in the supervisor. Thus the return to the supervisor is 'JMP @40'.

A typical use for the delayed execution facility is the outputting of a digital signal at a certain point in time. Such a single instruction action would not warrant the setting up of a whole task.

2.8 DEBUG TASK

This is a task which may be linked in with OSCAR systems to provide an on-line debugging facility. It uses the Teletype for input and output. If the Teletype is also required by other tasks the semaphore SENDT defined in DEBUG TASK provides mutual exclusion of the Teletype as a facility in different tasks. Unless output is taking place through the Teletype in another task, the Teletype keyboard is always receptive to DEBUG TASK commands. These follow the standard pattern of Nova Debug programs. Any memory location may be inspected and/or modified. A sequence of memory locations may be searched for a particular word after it is masked. This operation also allows listing of a sequence of memory locations. A Breakpoint may be entered at any memory location. Since DEBUG TASK is always active with other tasks this may be done even when the system has been set running. When the Breakpoint instruction is executed the task mode of operation is frozen and DEBUG TASK is run as a stand-alone program. This means that the instantaneous description of all other tasks which includes all variables and also private registers in TCB's may be inspected and modified. Because of the logical processor concept of *task* this scheme makes debugging of real-time systems very tractable. The task mode may be resumed with the 'proceed' operation of DEBUG TASK.

This dual mode of DEBUG TASK makes it a very powerful debugging tool. To the user the action of most operations look the same, whether DEBUG TASK is in task mode or at a breakpoint. The inspection of variables while a system is running is particularly useful. For example, the register which stores the value from an A/D converter may be monitored at any time without first stopping at a breakpoint. This is important when a system is controlling a factory process. It is then undesirable to stop the system. DEBUG TASK allows effective debugging even in this situation. The following is a case study of a typical debugging session.

By observation of the behaviour of certain variables and by inspection of the program listing it was determined that a control algorithm was faulty. This became evident because one control loop in a system containing a number of control loops was unstable. The system was running on line and the unstable behaviour was not severe enough to warrant a shutdown. A modification to the control algorithm program was written and checked on paper to make reasonably sure that it would work. Then the modification was entered into a spare section of the computer memory as a patch. The memory modifying function of DEBUG TASK was used for this purpose. The whole patch was typed in and checked while the rest of the system operated with the old algorithm.

Then a statement in the control algorithm was overwritten with a branch instruction to the patch. The next execution of that algorithm then executed the patch. The effect of the patch may then be observed. If the action of the patch makes the system worse the branch instruction is again overwritten with the old instruction. This restores the old algorithm. If the patch causes wild operation then the system will of course crash. But this kind of modification was carried out repeatedly on an on-line system and very few mistakes were made. The final operation if a patch is successful is to list it and also to punch out a binary tape of the patch. This may again be done while the rest of the system is on-line.

2.9 Applications of OSCAR

Two major systems have been designed and implemented with OSCAR. One is a mineral processing application the other is in the continuous production process category.

2.9.1 Ore Sorter

This is a machine developed at the C.S.R. Research Laboratories for the sorting of minerals. Pieces of ore which are in a given size range are passed through the machine in single file. Two sensors and one activator are mounted adjacent to the rock stream. These units are inputs to and output from the computer controlling the whole operation. The first sensor detects the presence of a rock and also measures its outline. A task is activated by every change in outline detected by this sensor called the position detector.

This task called the OUTLINE ANALYSER assigns sections of the outlines to data structures which represent individual rocks. By means of patented pattern recognition means[17] the representations are for individual rocks even if their outlines overlap with other rock outlines or are separated from them by diagonal or horizontal clefts only (which cannot be detected by simple logic) . As soon as the OUTLINE ANALYSER tasks recognises a rock whose outline is closed, it, in a sense, casts this rock adrift. This is done by a DELAY call through an EVENT CONTROL WORD in the Work area associated with each rock. This Work area also contains a TCB. Thus each rock has associated with it a task. This task is initially suspended. It waits for the posting of the ECW in the Work area.

The delay between the completion of the rock outline and the activation of the task associated with each rock is computed to be just after complete information from the second sensor becomes available and just before the rock comes in line with the activator.

The second sensor measures a physical parameter of the rock, which can be used to make a decision on the economic value of each piece of rock. The parameter measured is usually a surface parameter. This sensor is usually connected to the computer memory via a Data Channel because transfer rates are too high for program controlled transfers. In the memory a picture of this surface parameter is built up in a data block. It is the completion of this picture that the task associated with each rock waits for. The re-entrant program which these tasks execute is called the SURFACE ANALYSER. By means of the picture of the physical parameter along the whole rock stream and the outline of the particular rock the surface of each rock outline may be analysed separately. This analysis is carried out and the result is tested against a threshold value which may be varied by an operator. Rocks above the threshold value are valuable and pass straight through the sorter. Rocks below the threshold value are considered barren and are deflected into a separate stream by the activator. The activator is energised and de-energised by a subroutine whose execution is scheduled in the SURFACE ANALYSER by a DELEX call. Thus the deflection may occur some time after the analysis has been computed. The time delays involved are computed so that the physical rock is in line with the activator when it is energised. The duration of the activator pulse is tailored to the size of the rock.

The total execution time of all the task segments associated with one rock is 15 ms, on a Nova or 5 ms, on a Nova 1200. This speed allows sorting of 70 or 200 rocks per second which corresponds to 70 or 200 tons per hour. The system as installed uses a Nova. The actual time of flight of a rock between the first sensor and the activator is 150 ms, so that each rock has an occupancy of 10%. The system allows for 12 tasks for surface analysis so that there may be 12 rocks in various stages of analysis in this system at any one time.

Inspection of the code has shown that the time spent in the supervisor and the time spent in actually processing rock data is approximately 50/50. This may seem a high ratio for the supervisor. On the other hand there seems no way of pushing the sensors and the activator closer together so that the whole job could be done as a uni-program. The supervisor functions are actually useful towards getting the job done. Since this is an extremely fast system in data processing terms the time of 7.5 ms in the supervisor is also not very high. In this time an average of 10 task swaps are carried out.

The only observation which should be made is that there is great scope for hardware implementation of context switching and some of the other supervisor functions in high speed systems as the one described.

2.9.2 Materials Blending System

This is a system developed for a factory producing a continuous product which is made by mixing a number of dry and liquid materials. These materials are mixed according to a formula which is based on a recipe for the particular product and which contains parameters which reflect the chemical reactions taking place in making the product. The computer based system replaces a system which was largely controlled either completely manually or by pneumatic controllers. The computer either sets the set-points of electric controllers or controllers are implemented in the computer by direct digital control (DDC). Either way the computer also reads many plant parameters for control purposes or for giving alarms. The control system and alarm system constitutes the lowest level of this system. At a higher level is the computing of all set-points according to the formula for the current product. At a higher level again the Operator can monitor and modify all the plant variables and vary the recipes for all the product. This is done via a Television Terminal and Keyboard. At the highest level the system collects information about the current production for a number of shift logs which are printed automatically on a system printer.

The system was justified on the basis of reducing the variation in the product made. This aim has been achieved and a significant improvement in the weight variation of the final product can be maintained. Fringe benefits are ease of changing from one product to the next and ease of winding up the total speed of the process until some physical limit is reached. This used to be a difficult process before the computer system was installed because of the extensive calculations involved.

The different levels briefly described above are implemented as independent tasks. This provides a nice breakup of the work. This system was planned and coded by a number of programmers who were not involved previously with OSCAR. Work by these different programmers could be tested independently because of the task structure.

3 OTHER OPERATING SYSTEMS

The following sections are reviews of a number of Operating Systems or Significant papers about Operating Systems which have appeared recently. The systems are looked at with a view to their suitability as real-time operating systems. Any criticism is made with this point in mind.

3.1 A Multiprogramming System developed by B. Williams

Bruce Williams first introduced me to the concept of Tasks and Task Control Blocks[19]. Prior to this I had attempted to develop a system based on a Stack only. This had been coded during the first few months of 1970 and proved to be very intractable. There was no easy way of establishing in which order execution of various sections were going to proceed because of the unpredictable nature of Interrupts. Every now and again the system would die because of a bug and then it was nearly impossible to establish what belonged where on the Stack by inspection.

Bruce Williams system was made up of two sections, an Interrupt Handler which stored machine status on a Stack and a Task Scheduler which stored machine status in Task Control Blocks. I have modified this scheme by not having a stack, but the use of the Stack does allow the implementation of Device Service Routines which are themselves interruptible.

The Interrupt Handler consists of four modules:

Module 1 is entered after every interrupt. It saves 2 accumulators carry and the Program Counter in fixed locations. It then checks that the interrupting device is valid and transfers to the appropriate Device Service Routine if it is. Interrupt remains off.

Module 2 is the converse of Module 1. It is entered after completion of Device Service if that Device Service did not call on Module 3. It restores what was saved in Module 1 and returns to the interrupted program.

Module 3 is a subroutine called from a Device Service Routine if it is going to be lengthy and requires more accumulators. The accumulators carry and PC saved in fixed locations are transferred to the stack. The remaining accumulators are also stored on the stack. An Interrupt Priority Mask for the interrupted program is stored on the stack and a new mask which is passed as a parameter of the call from the Device Service Routine is set up. Interrupt is turned on unless the stack is about to overflow in which case it is left off.

Module 4 is the converse of Module 3 and 1. It restores all the status on one stack frame including the Interrupt Priority Mask. If the stack is about to become empty, the last stack frame is transcribed to the currently active task and a scan of all tasks is carried out on the assumption that one of the Device Service Routines may have changed the status of a higher priority Task than the currently active one. If the stack is not about to become empty Module 3 returns to the interrupted program.

The Task Scheduler is entered in two ways. One way is via Module 4 of the Interrupt Handler which has just been described. The second way is from a User program running as a Task when it executes the WAIT meta instruction. The WAIT routine of the Task Scheduler saves Location 6 and 7 as well as the accumulators carry and PC. The actual Task scheduler is then entered. This consists of executing the next instruction in every Task starting at the highest priority Task. This instruction, which is always the instruction following a WAIT call should be a test for some condition for which the Task is waiting. If the test fails the Task should transfer to NO and if it succeeds the Task should transfer to YES. The NO entry continues the task scan with the next lower priority task. The YES entry terminates the scan and sets up the Task which has just been tested for further execution.

A typical calling sequence for waiting for the Teletype Done flag to set would be:

```
WAIT            ; SUSPEND TASK AND TRY ALL OTHER TASKS
LDA 0, FLAG     ; GET FLAG SET SOMEWHERE
MOV 0,0,SNR     ; TEST FLAG
NO              ; FORGET THIS TASK FOR NOW
YES             ; PROCEED WITH THIS TASK
```

The system works but it is very slow. It is difficult to introduce an efficient service routine for a real-time clock. This made me look for an improved system. In designing OSCAR the following shortcomings were overcome:

1. Avoid too much copying from one register save area to another. Bruce Williams system saves some accumulators in three different locations. In a fixed location for simple Device Service. On the stack for more involved Device Service and then in the Task Control Block when the task state is reached. For each changeover the registers must be transferred. In OSCAR a register is immediately saved in the Task Control Block,

2. To determine the occurrence of an event the Task Scheduler must test software flags over and over again which introduces a large overhead for Task scanning. In OSCAR the POSTing of an event control word which is equivalent to setting a flag marks a Ready bit in the task queue which can be tested in a 3 statement loop per Task. In Bruce Williams system the shortest scan would be 6 statements. This will often be longer.

3. The Task scan must be carried out for nearly every interrupt in case that interrupt has caused a flag to be changed which would allow some Task to proceed.

In OSCAR a task scan would only be carried out when some task has actually been made Ready and not for every interrupt. This is the most significant means of cutting down the Task Scanning overhead. Some thought has been given to not doing a task scan at all but this involves setting up a structure in which priority of a Task that has just been POSTed can be simply compared with the current Task. This problem has not yet been solved but could lead to an even more efficient solution.

3.2 THE" - Multiprogramming System

This system developed by a team under the leadership of Edsger W. Dijkstra[12] is a very early exposition of the ideas of parallel processes, semaphores and verification of design and correctness of implementation. It was developed on a Dutch machine of which little is said except that it has an interrupt system to fall in love with (A property which A.M. Turing doubted a machine could have[20]). The aims of the system are modest, it incorporates a paged virtual memory and it uses independent processes for servicing various tasks that arise in the system. The paper is notable because it introduces the concept of Levels which is seen again in the Venus Operating System. Each level takes care of a number of machine functions, which then can be ignored at higher levels. Thus testing becomes much easier, because the operations at lower levels, once tested, may be ignored at the higher levels. This is a very important paper which provides much of the foundations for later systems.

3.3 The Venus Operating System

This system is a combined software/hardware project carried out on an Interdata 3 computer. It is an experimental multiprogramming system supporting six users who operate on-line and interactively through Teletypes. Its main distinction is that the system primarily caters for users who are co-operating with each other either via common data or through co-operating processes. The system was produced to provide a machine and a software system which would make the building of co-operating structures easier and to test the difficulties encountered.

In many ways the aims of the Venus project are similar to the aims I have outlined in this thesis. They have gone a step further by implementing hardware changes to a computer, which I was unable to do. I will propose in the final section of this thesis a number of hardware operations which a real-time computer should have. Many of these have been proposed and implemented on the Venus machine.

The features implemented are:

1. Segments
2. Multiprogramming of 16 concurrent processes
3. Microprogrammed multiplexed I/O channel
4. Hardware procedure calls

Segments are named virtual memories. Segments and core memory are both divided into 256 byte pages. Information about each core page is kept in a core-resident table, used by the micro-program to map virtual addresses into real-addresses. Paging is performed on demand of a page fault routine when the microprogram cannot locate a page in core.

Multiprogramming. A process is defined to be the execution on a virtual machine[6] . This is the same interpretation as used in OSCAR. The Venus system supports 16 virtual machines. These are made up of the address space which is the same for all virtual machines. This is unusual but the authors explain that they see this as an aid to implementing common data.

Each Virtual Machine has about 150 bytes of Work Area which is permanently located in core. This corresponds to Task Control Blocks in OSCAR. In the Venus system this block is rather large. The authors mention that the general registers and program counter may be found in the Work-Area (TCB) but no mention is made how these registers are swapped from the CPU to the work area and back when processes are re-scheduled. One possibility is that the micro-program operates directly on the memory locations holding these registers for the current process. If this is the case the system would be rather slow on most machines.

Synchronisation between processes is carried out exclusively by the 'P’ and 'V' operations on semaphores as defined by Dijkstra[5]. The implementation of the semaphore linkage to waiting processes is virtually identical to the implementation in OSCAR. The only difference is that re-scheduling after a 'V' operation is done on a priority basis. This is something worth investigating for OSCAR.

The Input/Output channel implementation appears to take advantage of the fact that interrupt servicing is carried out at the micro-program level and signalling to the process is achieved at the completion of an operation by performing the 'V' operation (RAISE) on a special semaphore. This semaphore is located in the Work Area (TCB) of the process which started the transfer. This appears to be an odd way of doing it, because in the earlier description on the implementation of semaphores it is stated that the address of the Work Area (TCB) is stored in the semaphore. Maybe the Work. Area simply provides a convenient spot which is available dynamically.

Procedures are stored in unique segments and may be used re-entrantly. A calling system, a means of passing arguments and a push down stack is implemented. Not much detail is given.

One very good conceptual feature of the Venus system is the systematic use of the idea of Levels of Abstraction as defined by Dijkstra[2]. This, the authors claim should lead to a better design with greater clarity and fewer errors. I agree wholeheartedly with this, and have shown that the OSCAR facilities are similarly structured.

The following levels for virtual devices have been used:

Level 0: Micro-program without real-time constraints
Level 1: Software controllers - one for each device
Level 2: Interface between User and Controller
Several other levels are presented.

3.3.1 Critical Comments on the Venus System

It is difficult to make valid criticism of a system one has not used. But the following points are felt to be shortcomings in the system which have been improved upon in the OSCAR system.

The stated exclusive use of semaphores for synchronisation is felt to be the biggest weakness. Although semaphores are very powerful, and Dijkstra attempts to prove that they are sufficient for all synchronising functions, it is difficult to see how to implement the case where one process waits on one of a number of events. Mention is made of this case in the paper and a mechanism called 'queues' is used for this purpose. Queues are operated on by 'Send' and 'Receive' operations. Queues are held in a common segment called 'queue segment'. This sounds a very similar scheme to 'RTOS' channels. For some reason the authors exclude this mechanism from process synchronisation. I suspect that this is a second means of process synchronisation and 'Queues' perform a similar function to 'Event Control Words' in OSCAR.

The arbitrary limitation to 16 processes is probably dictated by the micro-program of which details are not available. For real-time control it is easy to visualise systems with many more than 16 concurrent processes. These need not all be available with hardware context. But the Operating System should allow scheduling of extra low priority processes which may share the lowest priority hardware Work Area (TCB). Such low priority Processes are generally not time critical and the overhead in swapping TCB's would not be high.

The so called hardware implementation of the VENUS functions is carried out by micro-program. The use of micro-programmed processors for real-time work is a problem which I have not resolved to my own satisfaction. Micro-programming makes it possible to implement operations such as described above with relative ease compared with the design of these operations in hardware without micro-program. Until the need for these operations becomes generally recognised for building operating systems, micro-programming provides the most efficient solution. But with the imminent replacement of core memory with semi-conductor main memory I foresee that the machine instructions will be executed just as fast as present day microinstructions. Delay in gating circuits is going to be the limiting factor. Speed is always important, and the cost of a mass produced central processor which is designed for minimum execution time of all machine instructions will be negligible compared with a simpler micro decoder and micro memory which has to run at least 5 times faster than main memory to be of any use. Execution speed of some micro-'programmed instructions can be very slow, and often the use of a more powerful main instruction set working at micro-instruction speed will provide a faster solution. Operations such as 'P' (LOWER) and 'V' (RAISE) should actually be included in any instruction set.

3.4 The Data General Real-Time Operating System (RTOS)

This is a system for the Nova family of computers[10] which became available in Australia towards the end of 1971. It fulfils a need similar to OSCAR and it draws on a common background. The introduction to the Manual summarises the aims of the system:

> RTOS consists primarily of a small, general-purpose multi-programming monitor designed to control a wide variety of real-time input/output devices. User programs are relieved from the details of I/O timing, data buffering, priority handling and task scheduling. In addition they are provided with a parallel processing capability plus inter-task communication and synchronisation facilities.

RTOS Tasks are organised in four states. Executing, Pending, Suspended and Dormant. Only the last state is new for OSCAR users and it simply describes the condition when the Task Control Block is on a list or pool of unused or available blocks.

The Task Control Block in RTOS only saves the PC, Accumulators and Carry. It also contains a Priority number which can be varied from the Task and a Link Word. No private memory locations are provided.

RTOS provides eight meta-instructions for users. Whenever a meta instruction is executed control is returned to the user via the RTOS task scheduler. A brief listing of these meta instructions is:

```
.IOX
<logical device#>
<device control word>
<first data item pointer>
<data item count>
<error routine address>
<normal return>
```

This call initiates Input/Output on the device specified according to a scheme encoded in the device control words and to a buffer described by the pointer and counter. Return is not made until the transfer is complete. This type of call has the following shortcomings in a control environment:

1. It allows only for data transfer not for control actions.
2. Buffer locations and size has to be determined at assembly time. There is no facility for using dynamic storage.
3. The operation is at too high a level • No lower level operations are available to users. Thus simpler operations such as transfers of single bytes are too cumbersome and time consuming.

```
.FORK
<new task priority>
<new task address>
<next statement in current task>
```

This call is the only way of generating more tasks. Because the call has the appearance of a branch instruction, there is a temptation to generate new tasks all the time. This is time consuming. The examples given in the RTOS manual do exactly this, so I feel my fears are justified.

```
.QUIT
<next statement>
```

This call places a task in the dormant state. The recommended way of executing a task in parallel with Input/Output is the following abbreviated coding sequence:

```
.FORK    ; CREATE A PARALLEL TASK
.IOX     ; DO I/O
.QUIT    ; DELETE TASK WHEN I/O COMPLETE
         ; CARRY ON PARALLEL PROCESSING
```

NOTE: For every time this code is executed the main stream is shunted to another task. It is often difficult to see in RTOS programs what code belongs to which Task and to keep track of parallel operation. The programs still look like a conventional serial program.

```
.PTY
<New Priority Level>
<next statement>
```

This operation alters the priority of a task dynamically.

```
.WAIT
<# of clock cycles>
<next statement>
```

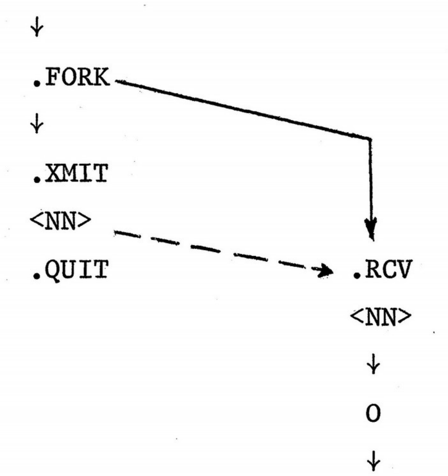
This operation is used to delay the execution of the current task for a specified time interval. It is the only way of accessing the Real Time Clock. Again I feel this operation is at too high a level and does not allow the user enough scope to do simpler things with the real-time clock. Maximum clock rate is 100 Hz.

```
.XMIT
<Channel #> or @<Channel #>
<next statement>
```

```
. RCV
<Channel #>
<next statement>
```

This is a complementary pair of operations which are provided for the purpose of Task synchronisation. The .XMIT command causes transmission of a 'synchronisation signal' over the specified channel. If an '@' sign is present in the channel number argument, the Task will be placed in the suspended state until the .RCV command on the same channel has been executed. Otherwise the Task will be allowed to continue. Upon executing the .RCV command, the current Task is placed in the suspended state until the signal is received, at which time it will be made pending and become available for execution again. A fixed number of channels (usually 8) must be set at system generation time. These are all the channels that are available to users.

The following sequence is the recommended way of implementing Conways FORK-JOIN operations:



I would regard such schemes as a waste of time in an environment in which only one processor is available. Conway created this structure for allowing the use of more than one processor on one problem. In OSCAR a structure is developed for allowing one processor to do a number of problems at the same time. RTOS does not emphasise this point enough.

The final operation is:

```
.SBRK
<Character code (ASCII)>
<return>
```

This instruction is not intended to function as a general purpose meta-command. Its use is primarily intended for the operation of a keyboard orientated executive. Its operation has to be set up at system generation time. Usually a Teletype interrupt service is enabled for the Break feature, and a special task is created which is suspended with the .BRK call until the character in the call is keyed on -he Teletype. Then all other tasks waiting for I/O on the Teletype are made dormant and the special break task is made active.

This again is a very high level and specialised operation which could be implemented very easily with the basic functions of OSCAR. Because these are not available in RTOS, this high level function becomes necessary. To illustrate the point I will show how the same facility could be implemented under OSCAR:

- In the Interrupt service routine for each Teletype Keyboard which is to be allowed to cause a break, test the character transmitted by the keyboard after the interrupt against the break character, which must be stored at some convenient location. If the character received is the break character, POST a special EVENT CONTROL WORD which may be called BRKEC for example. The character is not treated as data from then on.
- Provide a special Task which carries out the function of the Keyboard Executive. After initialisation, this task is made to WAIT on the EVENT CONTROL WORD BRKEC. Thus if the break code is typed on a keyboard this Task is activated.
- If a number of keyboards may cause a break to occur, the POSTing in the Interrupt service routine may store a device identification in this POST code section of BRKEC. The Keyboard Executive Task can test this code.

Since RTOS is a system on the same computer I am working on, actual hands on experience of the system was possible. I have coded a number of test programs and tried to compare them with OSCAR. For this reason any criticism can be much more detailed.

The following features were found to be inadequate:

1. Teletypes and Teletype like devices were not treated as separate Keyboard and Printer devices but as a composite. This introduces a number of logical difficulties which need not be there. Since this is the most complicated device, it is seen as the prototype and much of this complication is carried across to single devices such as the Line Printer.
2. The system is rather long. It is in excess of 2,000 words against 600 for OSCAR. Execution speed is slower. The basic overhead for each interrupt is 70µs as against 14µs in OSCAR.
3. The system is monolithic, despite claims to the contrary. It consists of one relocatable program. Included in the 2,000 words are the complete lOX package and one Teletype driver. Neither of these may be left out if not required. More Teletype or other device drivers require more relocatable programs and more space.

4. Space and time is wasted by saving the processor status in a block reserved in each device service routine when an interrupt occurs, and then transferring this status to the TCB if this interrupt leads to a change of Tasks.

3.5 HP 2005A Real Time Executive System

This is a system for the HP 2116 B computer. It has a task structure which is very similar to RTOS, although the word task or process is not used. The manual talks about parallel programs and the name given to the TCB is the Program Identification Segment. It again recognises four states:

1. Execution
2. Suspended
3. Scheduled
4. Dormant

I/O processing goes a step further than in RTOS by allowing the stacking of I/O requests. This is implemented in OSCAR via Double Ended Queues.

An Operator Keyboard Monitor is an intrinsic part of this system. The operator can change program status, operating environment and load, start, and stop programs. This feature is at a higher level again than the RTOS BRK feature and in my view should be a separate facility, which may be used if it is wanted.

3.6 The HP 12659A DACE System

This system is called a Data Acquisition and Control Executive (DACE) which can be run on 2114A, 2115A or 2116B computers with 8K Memory.

It is organised on a so called Task structure, but a DACE Task has a different connotation to a process. In DACE a task is a program, which is activated at regular intervals as defined by a real-time clock. An Interval time and a Phase time are task parameters. These are usually set at system generation time but can be altered dynamically or through the system keyboard. The scheduling time can only be set in seconds. No finer resolution is possible. The manual gives examples of tasks scheduled every 30 seconds. This time looks typical.

Examples of tasks that may be scheduled are Data Acquisition scans and execution of control algorithms at regular intervals. Character input and output is via a buffered interrupt driven module.

The Description of Tasks in this system emphasises the serial nature of the programs. Each Task when scheduled will start at a specified starting point, not at the place where it last stopped as in OSCAR. The Task will then run to its finishing point which is coded in some way. For testing purposes a Task may be run from start to finish once by a command from the system console.

The system is compatible with a FORTRAN II and ALGOL run time library and formatter which is very useful. But this means that the system uses 80% of memory in an 8K system leaving only 2K words for User programs. This software package cost $1000 in 1969.

There are a number of similar systems on other computers which vary in small details but which are mainly driven by a real-time clock scheduler.

Examples are CAMP for the LN5100 processor, RSX-15 for the PDP-45 and MOS for the Varian 620. There is also RTX for a SPC-42 processor.

RSX-45 and a recently released system RSX-11 for the PDP-11 are probably the most versatile systems because they do contain system calls which would allow synchronisation with arbitrary events. The presentation of these system stresses the 'scheduling at regular intervals' approach. I have seen no system except 'Venus' which incorporates semaphores, allows scheduling of tasks from user interrupt service routines and implements a general double ended queue system for data buffering. None of the systems seen incorporate a real-time clock routine which provides both interval counting and event scheduling and is fully re-entrant so that it may be used for several pulsed inputs simultaneously. Also no system has been seen which allows accurate frequency measurement from pulsed inputs.

3.7 VORTEX. Varian Omnitask Real Time Executive

VORTEX is a Real-Time executive which combines user written tasks with a Job control processor which can handle language processors as background tasks. To allow full use of foreground background processing a Varian 620 computer with 16K of memory and a rotating disc file is required.

The system is organised around a Task Scheduler and an Interrupt handler. Task Control Blocks are threaded (a word used in the VORTEX manual) on two lists. These are the busy list and the unused list. TCB's on the unused list are dormant. As Tasks are scheduled a TCB is taken from the unused list and threaded onto the busy list. The threading operation is carried out according to a priority number which is supplied as a Scheduling parameter. There are two system tasks which may be scheduled under certain conditions.

1. SAL is a memory allocation module which is activated if a task is not resident in memory.
2. ERROR is activated for a number of common errors.

There is a space for a variety of non-resident foreground and background tasks. Background tasks may be pre-empted once, if the space they occupy is required by a foreground task. The resident portion of the system is comparatively large. 0.5K in low memory plus 6K in high memory.

The communication between users and the real-time executive is via system macros which, when expanded have the general form of a subroutine call to the system with a number of arguments which follow this call. The first argument gives the function which is required. This is executed interpretively by the system. The other arguments vary for the different calls. Such a scheme must be fairly slow compared with direct subroutines for each function which is used in OSCAR.

Functions which are available in VORTEX and the nearest equivalent in OSCAR or other systems are listed below:

FUNCTION	EQUIVALENT	DESCRIPTION
SCHED	FORK	Schedule a task
SUSPND	(WAIT)	Suspend a task
RESUME	(POST)	Resume a task
DELAY	DELAY-WAIT	Delay a task
PMSK	-	Store hardware priority mask register
TIME	TIM	Obtain time of day
OVLAY	-	Load and/or execute an overlay segment
ALOC	-	Allocate a re-entrant stack
DEALOC	-	De-allocate the current re-entrant stack
EXIT	EXIT	Exit from a task upon completion

ABORT	-	Abort a task
IOLINK		Link background I/O

Most of these operations are self explanatory in the context of this paper but the following comments are in order about the SUSPND and RESUME operation.

The SUSPND function suspends the execution of a task making the call. The task can be resumed only by an interrupt or a RESUME call in another task. The type of resumption which is anticipated is in an argument of the SUSPND call. If interrupt resumption is specified, there is no indication which interrupt is going to do it. This is a function of previously set up information in Interrupt event word of the TCB which links a Line Handler (Device Interrupt handler) and the task when the particular line interrupts. The Resume call in another task must nominate the task which is to be resumed. No mention is made in the write up what happens when a SUSPND call is made and the Interrupt or the RESUME call it is suspended for has already occurred. I suspect this is not allowed, and this would make the writing of tasks much more difficult under VORTEX.

The DELAY operation only allows the calling task to be suspended, and the time of resumption is computed from the time the call is made. This is similar to the RTOS Delay call. The VORTEX call does allow an interrupt to cause earlier resumption. Thus DELAY can be used to time out a device. In OSCAR this must be done with a multiple WAIT operation.

There are a number of I/O control functions in VORTEX which are coded at a higher level in OSCAR. The most important of these are:

```
OPEN, CLOSE, READ, WRITE, STAT
```

The last of these is interesting. STAT refers to the address of a READ or WRITE macro and tests the status of the transfer. If this transfer is not complete it transfers to a user nominated busy routine. The VORTEX manual warns that this function should not be used in foreground tasks because it hangs up the System. The completion of I/O functions cannot be tested in any other way, so the only way to allow parallel processing of I/O and other computations is to generate parallel tasks with a SCHED (FORK) function. A general WAIT operation such as is used in OSCAR is clearly missing. Also missing are operations to mark the boundaries of critical sections such as LOWER and RAISE.

This system is extended to be a full Disc Operating system. The Job Control Language and the means for managing background tasks merit further study if an extension for OSCAR in this direction is undertaken.

3.8 The Tenex Time Sharing System

This is a paged time sharing system for a PDP-10 computer[15]. This system is unusual in that the use for this system is mainly a multi-terminal computing facility. As such it is not very different to users than a number of other time sharing systems. But the implementation is very different because it is based on a state of the art virtual machine and a multiple process capability with appropriate communication facilities.

This confirms a belief I have, that the use of systems using parallel processing is not confined to real-time control applications, but that these schemes allow the implementation of very powerful general purpose computing facilities with attributes which users would like to have. At present there is no satisfactory mini-computer system which will allow simultaneous Fortran compilation and execution of other programs, such as Basic. This problem would be solved by using a parallel process orientated system such as OSCAR as the kernel of a more general Operating System.

Tenex is such a system for a large processor. A number of changes were made to the hardware of the processor. The most important of these is a paging mechanism. Enough information is kept about each page to determine if it is in core, if it has been modified or not, (this saves copying back to disc) and if a shared page is about to be modified. If it is, a private copy will be generated. This allows the running of non-pure procedures as if they were re-entrant.

No mention is made of the implementation details of process synchronisation. All input output is carried out through the executive in a fairly standard manner. The implementation of File Handling is worth looking at. Being a system which may have many simultaneous compute bound processes a fairly sophisticated Scheduler is used. It is based on the concept of Occupancy and the paper provides many useful hints on how to implement such a system.

4 CONCLUSIONS

The time involved in planning and coding the first version of OSCAR took 4 months during the latter part of 1970. During the 2 years since then a number of modifications to the system have been made. In particular Semaphores and Double Ended Queues were added. Experience with the system has shown that it is easily picked up by programmers. The functions, particularly the synchronising primitives, are at first unfamiliar but with a little practice they are used as intended. Thus parallel programming as against uni-programming comes naturally to most programmers when a task structured system such as OSCAR is available to them. There is still a certain amount of resistance to going all the way with the parallel approach. Programmers traditionally join their real-time--routines into a string which is repeated at regular intervals (usually once per second). In the parallel approach each routine would be coded as a separate task with its own repetition time. It can be argued that this is not quite as efficient as the one second loop approach. But the increase in flexibility would often outweigh this slight decrease in efficiency.

The effort in producing this system is felt to have been worthwhile. Such a system as the Ore Sorter could never have been made fast enough with systems such as RTOS which have become available in the meantime.

The work was influenced by the academic discipline. This has resulted in a broad survey of the field before the actual implementation of a system was started. The work was also influenced by the constraints of industry. This has resulted in a system which has had to stand up to the tests of being used in a real and generally hostile environment. OSCAR hopefully contains the best of the current state of the art in the real-time operating system field and represents an implementation which is up to engineering standards. The fact that it is still up to date after two years is at least gratifying.

Some extensions of OSCAR which are useful have been outlined in the body of this thesis. The usefulness of this and similar operating systems on small computers could be made even greater if the computer architecture were modified to implement some of the lower level operations which must now be done by software. This would speed up execution significantly. Some computers already have such hardware. For a Nova the following additional hardware facilities are recommended:

1. Extended memory mapping facility allowing 3 maps and 3 sets of accumulators associated with 3 processor states (i) Interrupt Processing, (ii) Executive Processing and (iii) User Processing.
2. Allow I/O instructions only during the Interrupt Processing and Executive Processing states. In the User Processing state these instructions are replaced by a general Supervisor Call (SVC) whose execution unconditionally traps to Executive Processing Mode and three conditional trap instructions which have the format of Memory Reference Instructions without Accumulator in the Nova Instruction set. These are:

   (i) **Increment and Trap if Minus or Zero**. This is used to implement the 'RAISE' operation on a Semaphore.

   (ii) **Decrement and Trap if Minus**. This is used to implement the 'LOWER' operation of a Semaphore.

   (iii) **Increment and Trap if Positive**. This is used to implement the 'WAIT' operation on an Event Control Word. This scheme requires a slightly different format for Event Control Words.

A fourth instruction 'Decrement and Trap if Zero or Positive' would complete the set and would do the 'POST' operation. This could be implemented on other machines which have more instructions to spare. This instruction is mainly used in Interrupt Service, where a Trap does not apply.

With the aid of these extra instructions and a hardware Supervisor Call linkage all the occupancy overhead of Task synchronisation operations would be reduced drastically. Particularly semaphores could be used much more freely. In most instances critical section are not critical and there would be no hold up. It is just the odd case which happens say once in a thousand, in which a semaphore must catch another task using a critical section.

The simple Wait operation which is used frequently in I/O routines is also speeded up. A Multiple Wait must still be specially written.

It is hoped that the general acceptance of a process or task oriented way of programming will induce manufacturers to provide computers with these or similar facilities.

4.1 References

1. Wirth, N., On Multiprogramming, Machine Coding and Computer Organisation. Comm. ACM 12, 9 (Sept. 1969), pp 489-498.

2. Wegner, P., Programming Languages, Information Structures, and Machine Organisation. McGraw Hill, New York, 1968.

3. Digital Equipment Corporation, Introduction to Programming.

4. Habermann, A. N., Synchronisation of Communicating Processes. Comm. ACM 15, 3 (March 1972), pp 171-176.

5. Dijkstra, E. W., Co-operating sequential processes. In Programming Languages, F. Genuys, Ed., Academic Press, New York, 1968, pp 43-1129

6. Lampson, B W. , A Scheduling Philosophy for Multiprogramming Systems. Comm. ACM 11, 5 (May 1968),.pp 347-359.

7. IBM Operating System/360. Concepts and Facilities. In Programming,-Systems and Languages, S. Rosen, Ed., McGraw-Hill, .New York, 1967, pp 598-646.

8. Conway, M. E., A Multiprocessor System Design, Proc. FJCC 1963, pp 139-146.

9. Dijkstra, E. W., Programming considered as a human activity in Proc. IFIP Congress 65, vol. 1, 213-2170

10. Data General Real-Time Operating System, Data General Corporation, Southboro Massachusetts.

11. Benson, D. and Others, A Language for Real Time Computing Systems. British Computer Society report of On-line computing systems and . languages working party.

12. Dijkstra, E. W., The Structure of the "THE"-Multiprogramming, System. Comm. ACM 11, 5 (May, 1968), pp 341-346.

13. Liskov, B. H. , The design of the Venus operating system. Comm. ACM, 15, 3 (March 1972), pp 144-149

14. Dennis, J. B. , and Van Horn, E. C. , Programming semantics for multiprogrammed computations Comm ACM 9, 3 (March, 1966), pp 143-155

15. Borrow, D. G. , Burchfiel, J. D. , Murphy, D. L. , Tomlinson, R. S. TENEX, a paged time sharing system for the PDP-10, Comm ACM 15, 3 (March 1972) pp 135-143.

16. Varian OinniTask Real Time Executive (VORTEX), 98A9952 (Sept 1972) Varian Data Machines, Irvine, California

17. Dwyer, F. B , Thompson, R. L. , Wulff, E., High Speed Sorting, Australian Patent 425088.

18. Phister, M , Logical Design of Digital Computers, John Wiley and Sons, Inc., New York.

19. Williams, B., Private Communication.

20. Turing, A.M., Can a Machine Think, in the World of Mathematics, ed.-Newman,, T. R., Vol. 4, pp 2099-2123. George Allen and Unwin Ltd.

21. How to use the Nova Computers, Data General Corporation, Southboro Massachusetts, 1972.

5 APPENDIX

The derivation of instruction mnemonics for NOVA computers is as follows[21]:

LoaD / STore } Accumulator

Increment / Decrement } and Skip if Zero

JuMP

Jump to SubRoutine

COMplement / NEGate / MOVe / INCrement / ADd Complement / SUBtract / ADD / AND } for carry bit base value use { current carry / Zero / One / Complement of current carry } { ~ / shift Left / shift Right / Swap bytes } { ~ / # }

SKiP / Skip { on Zero / om Nonzero } { Carry / Result }; if Either is Zero; if Both are Nonzero

No IO transfer — and { ~ / Start / Clear / special Pulse }
Data { In / Out } { A / B / C } buffer — and { ~ / Start / Clear / special Pulse }

SKiP if { Busy / Done } is { Nonzero / Zero }

READ Switches

IO ReSeT

HALT

INTerrupt Acknowledge

MaSK Out

INTerrupt ENable

INTerrupt DiSable

MULtiply

DIVide

The following mnemonics have been added to make arithmetic tests easier to interpret:

IFEQ	s,d	SUB#	s,d,SNR	; next if s == d	IFZ	s,-	MOV#	s,-,SNR	; next if s == 0
IFNE	s,d	SUB#	s,d,SZR	; next if s != d	IFN	s,-	MOV#	s,-,SZR	; next if s != 0
IFGE	s,d	ADCZ#	s,d,SNC	; next if s >= d	IFZP	s,-	MOVL#	s,-,SNC	; next if s >= 0
IFLT	s,d	ADCZ#	s,d,SZC	; next if s < d	IFM	s,-	MOVL#	s,-,SZC	; next if s < 0
IFGT	s,d	SUBZ#	s,d,SNC	; next if s > d	IFP	s,-	NEGZL#	s,-,SEZ	; next if s > 0
IFLE	s,d	SUBZ#	s,d,SZC	; next if s <= d	IFZM	s,-	NEGZL#	s,-,SBN	; next if s <= 0
				; else skip	IFM1	s,-	COM#	s,-,SNR	; next if s ==-1
					IFNM1	s,-	COM#	s,-,SZR	; next if s != -1
CLA	d,d	SUBC	d,d	; clear acc d					; else skip

‘s’ is source Acc 0, 1 ,2 or 3; ‘d’ is destination Acc 0, 1, 2, or 3; ‘-’ Acc not used (usually same as s)

The following pages contain the full listings of the OSCAR system in Nova assembler code.

```
0001 .MAIN
               ; INTERRUPT HANDLER AND TASK SCHEDULER  MK. V
               ; ------------------------------------------
               ; E. WULFF      7-APR-71
               ; TASK CONTROL BLOCK DISPLACEMENT ALLOCATION
        000000 .DUSR   TAC3=   0       ;SAVE AC3
        000001 .DUSR   TAC2=   1       ;  "  AC2
        000002 .DUSR   TAC1=   2       ;  "  AC1
        000003 .DUSR   TAC0=   3       ;  "  AC0
        000004 .DUSR   TPCC=   4       ;  "  PC+CARRY
        000005 .DUSR   TPM=    5       ;  "  PRIORITY MASK
        000006 .DUSR   TL6=    6       ;  "  LOC 6  TEMP. REGISTER
        000007 .DUSR   TL7=    7       ;  "  LOC 7  WORK AREA POINTER
        000010 .DUSR   TL20=   10      ;  "  LOC 20 AUTO INC REGISTERS
        000011 .DUSR   TL21=   11      ;  "  LOC 21 OR TEMP. REGISTERS
        000012 .DUSR   TL30=   12      ;  "  LOC 30 AUTO DEC REGISTER
        000013 .DUSR   TL31=   13      ;  "  LOC 31 STACK POINTER
        000014 .DUSR   TL40=   14      ;  "  LOC 40 GET CHAR. POINTER
        000015 .DUSR   TL41=   15      ;  "  LOC 41 PUT CHAR. POINTER
        000016 .DUSR   TWC=    16      ;WAIT COUNT OR SEMAPHORE LINK
        000017 .DUSR   TBP=    17      ;BACK POINTER
        000020 .DUSR   TL=     20      ;TASK CONTROL BLOCK LENGTH
               ; THE LENGTH OF THE PRIVATE MEMORY REGISTER STORAGE
               ; IS TL-10.
               .END
```

```
0001  START

               ; SYSTEM START UP
               ; --------------

                ; TASK SCHEDULER MK. V

               ; E. WULFF      9-MAR-72

               .TITL   START

               .ENT    BEGIN
               .EXTD   FTP,ATCB,TS3

       000002 .LOC    2        ; SYSTEM START AND RESTART LOCATION

00002 002000-START:  JMP     @BEGIN

               .ZREL

00000-000000'BEGIN:  ST

               .NREL

00000'062677 ST:     IORST           ; RESET ALL I/O
00001'030001$        LDA     2,FTP   ; HEAD OF TASK QUEUE
00002'050020         STA     2,20

00003'036020 L1:     LDA     3,@20   ; NEXT TCB ADDRESS
00004'174014         IFNM1   3,3
00005'000411         JMP     L3      ; LEGITIMATE TASK ADDRESS

00006'030020         LDA     2,20    ; PRESENT ENTRY POINTER
00007'021001 L2:     LDA     0,1,2   ; NEXT ENTRY
00010'041000         STA     0,0,2   ; STORE IN THIS ENTRY
00011'151400         INC     2,2
00012'101014         IFN     0,0     ; TEST IF END OF LIST
00013'000774         JMP     L2      ; NO - RE-WRITE FURTHER

00014'014020         DSZ     20      ; MOVE POINTER BACK
00015'000766         JMP     L1      ; TRY ENTRY AGAIN

00016'165125 L3:     MOVZL   3,1,SNR
00017'000424         JMP     L5      ; END OF TASK QUEUE

00020'125220         MOVZR   1,1     ; BIT 0 CLEARED
00021'044006         STA     1,6     ; TEMPORARY SAVE
00022'030430         LDA     2,MTL   ; 1 - LENGTH OF TCB
00023'146400         SUB     2,1     ; POINT TO ICW-1
00024'044021         STA     1,21
00025'026021         LDA     1,@21   ; ICW

00026'102460 L4:     SUBC    0,0
00027'125122         MOVZL   1,1,SZC ; TEST BIT IN ICW
00030'022021         LDA     0,@21   ; PICK UP INITIAL VALUE
00031'041400         STA     0,0,3   ; STORE 0 OR INITIAL VALUE
00032'175400         INC     3,3
00033'151404         INC     2,2,SZR ; COUNT
00034'000772         JMP     L4
```

```
  0002  START
01 00035'034006          LDA      3,6      ; FIX TASK QUEUE ENTRY
02 00036'030020          LDA      2,20     ; AND BACK POINTER
03 00037'051417          STA      2,TBP,3
04 00040'137000          ADD      1,3      ; ICW BIT 15 -) BIT 0 OF
05 00041'055000          STA      3,0,2    ; TASK QUEUE ENTRY
06 00042'000741          JMP      L1
07
08 00043'030001$L5:      LDA      2,FTP    ; SEARCH FOR 1ST ACTIVE TASK
09 00044'050020          STA      2,20
10
11 00045'036020 L6:      LDA      3,@20
12 00046'175112          MOVL#    3,3,SZC
13 00047'000776          JMP      L6
14
15 00050'054002$         STA      3,ATCB   ; INITIALISE ACTIVE TCB POINTER
16 00051'002003$         JMP      @TS3     ; ENTER TASK SCEDULER
17
18 00052'177761 MTL:     1-TL     ; ASSEMBLE WITH CORRECT ASSEMBLER
19
20       000002 .END     START    ; SYSTEM START ADDRESS
  0003  START

ATCB    000002$X    2/15
BEGIN   000000-     1/16    1/20
FTP     000001$X    1/25    2/08
L1      000003'     1/28    1/40    2/06
L2      000007'     1/33    1/37
L3      000016'     1/30    1/42
L4      000026'     1/52    1/58
L5      000043'     1/43    2/08
L6      000045'     2/11    2/13
MTL     000052'     1/47    2/18
ST      000000'     1/20    1/24
START   000002      1/16    2/20
TS3     000003$X    2/16
```

```
0001  TQ

                  ; TASK QUEUE & DUMMY BACKGROUND TASK
                  ; ---------------------------------

                  ; TASK SCHEDULER  MK. V

                  ; E. WULFF       8-APR-71
                  ; MODIFIED       16-MAR-72

                  ; THIS ASSEMBLY PROVIDES THE COMPLETE TASK
                  ; QUEUE. INDIVIDUAL ENTRIES ARE LOADED BY THE
                  ; RELOCATABLE LOADER AS REQUIRED. THE LAST ENTRY
                  ; IN THE TASK QUEUE IS BACKGROUND AND THIS
                  ; WILL ALWAYS BE ACTIVE. ANY ENTRIES WHICH
                  ; ARE NOT DEFINED BY THE USER ARE LOADED AS
                  ; 177777 BY THE REL. LOADER. THESE ENTRIES
                  ; WILL BE EXTRACTED BY THE SYSTEM START PROGRAM.
                  ; THE TASK QUEUE WILL THUS BE CONSOLIDATED
                  ; TO INCLUDE ONLY TASKS ACTUALLY REQUIRED.

                  ; BACKGROUND COMPLEMENTS CARRY APPROXIMATELY
                  ;       TWICE A SECOND ON A NOVA
                  ;       10 TIMES A SECOND ON A SUPERNOVA
                  ; WHEN THE SYSTEM IS LIGHTLY LOADED.
                  ; THIS PROVIDES A HANDY VISUAL INDICATION THAT
                  ; THE SYSTEM IS RUNNING CORRECTLY.

                  .TITL   TQ

                  .ENT    FTP,TCBZZ,END
                  .EXTN   TCBA,TCBB,TCBC,TCBD,TCBE,TCBF,TCBG,TCBH,TCBI
                  .EXTN   TCBJ,TCBK,TCBL,TCBM,TCBN,TCBO,TCBP,TCBQ,TCBR
                  .EXTN   TCBS,TCBT,TCBU,TCBV,TCBW,TCBX,TCBY,TCBZ

                  .ZREL

 00000-000023'FTP:        STQ-1   ;FIRST TASK POINTER

                  .NREL

                  ; BACKGROUND TASK

                  ; TASK CONTROL BLOCK

       000020 TCBZZ:  .BLK    20

 00020'005400 1B4+1B6+1B7     ; INITIALISATION CONTROL WORD

 00021'000006         6       ; PC - BACKGROUND PROGRAM
 00022'101400         INC 0,0 ; L6 - EXECUTES AT LOC 6
 00023'000006         JMP 6   ; L7 - AND LOC 7

                  ; TASK QUEUE

 00024'177777 STQ:    TCBA    ; TASKS IN ALPHABETICAL ORDER
 00025'177777         TCBB    ; TO ESTABLISH PRIORITY
 00026'177777         TCBC
 00027'177777         TCBD
 00030'177777         TCBE
```

```
  0002  TQ
01 00031'177777          TCBF
02 00032'177777          TCBG
03 00033'177777          TCBH
04 00034'177777          TCBI
05 00035'177777          TCBJ
06 00036'177777          TCBK
07 00037'177777          TCBL
08 00040'177777          TCBM
09 00041'177777          TCBN
10 00042'177777          TCBO
11 00043'177777          TCBP
12 00044'177777          TCBQ
13 00045'177777          TCBR
14 00046'177777          TCBS
15 00047'177777          TCBT
16 00050'177777          TCBU
17 00051'177777          TCBV
18 00052'177777          TCBW
19 00053'177777          TCBX
20 00054'177777          TCBY
21 00055'177777          TCBZ
22 00056'000000'         TCBZZ    ; BACKGROUND
23 00057'000000          0        ; MARKER FOR START
24
25               END:             ; 1ST FREE LOCATION
26
27               .END

0003  TQ

END      000060'    2/25
FTP      000000-    1/37
STQ      000024'    1/37    1/55
TCBA     000024'X   1/55
TCBB     000025'X   1/56
TCBC     000026'X   1/57
TCBD     000027'X   1/58
TCBE     000030'X   1/59
TCBF     000031'X   2/01
TCBG     000032'X   2/02
TCBH     000033'X   2/03
TCBI     000034'X   2/04
TCBJ     000035'X   2/05
TCBK     000036'X   2/06
TCBL     000037'X   2/07
TCBM     000040'X   2/08
TCBN     000041'X   2/09
TCBO     000042'X   2/10
TCBP     000043'X   2/11
TCBQ     000044'X   2/12
TCBR     000045'X   2/13
TCBS     000046'X   2/14
TCBT     000047'X   2/15
TCBU     000050'X   2/16
TCBV     000051'X   2/17
TCBW     000052'X   2/18
TCBX     000053'X   2/19
TCBY     000054'X   2/20
TCBZ     000055'X   2/21
TCBZZ    000000'    1/45    2/22
```

```
0001  TS1

                ; INTERRUPT HANDLER AND TASK SCHEDULER
                ; ---------------------------------

                ; PART 1 - THIS SECTION MUST BE LOADED EARLY
                ; WHILE ZREL IS LESS THAN 200

                ; MK. V

                ; E. WULFF       8-APR-71
                ; MODIFIED       8-NOV-71         16-MAR-72
                ; INCLUDE DEFI   14-JUL-72

                ; VERSION B FOR THE GYPROCK SYSTEM.

                 ; PROVIDES FOR THE FOLLOWING DEVICES:

                ;       00      ;POWER FAIL
                ;       10-11   ;TTI, TTO
                ;       13-14   ;PTP, RTC
                ;       24-27   ;FL1-4  FLOW METER 1 TO 4
                ;       50-51   ;INI, INO

                .TITL   TS1

                .ENT    ATCB,RTI,PMASK,COMP,C1,C2,C3,C4,C5
                .ENT    C6,C7,C10,C11,C12,C14,C15,C40,C77,C177,C377
                .ENT    CFR1,SAV0,SAV1,SAV2,SAV3,SAVC,SAVR
                .ENT    TS,TS0,TS1,TS2,TS3,TS4,DST
                .ENT    CL,NC
                 .EXTD   BEGIN
                .EXTN   PFLS,TTIS,TTOS,PTPS,RTCS,FL1S,FL2S,FL3S,FL4S,INIS
                .EXTN   INOS,RINT,TSCH,TSCH0,TSCH1,TSCH2,TSCH3,TSCH4

        000100 CL=      100     ; BYTE LENGTH OF A DQ. CELL
        000010 NC=      10      ; NUMBER OF DQ. CELLS

                .ZREL

                ;DEVICE SERVICE TABLE

 00000-177777 DST:     PFLS    ;POWER FAIL SERVICE

 00001-177777  PMASK:  -1       ;INTERRUPT PRIORITY MASK
                                ;DEVICE 1 (MDV) CANNOT INTERRUPT.

                ;SOME TEMPORARY SAVE REGISTERS

 00002-000000 SAV0:    0
 00003-000000 SAV1:    0
 00004-000000 SAV2:    0
 00005-000000 SAV3:    0
 00006-000000 SAVC:    0
 00007-000000 SAVR:    0

                ;STANDARD I/O DEVICES

 00010-177777          TTIS    ;TELETYPE IN SERVICE
 00011-177777          TTOS    ;TELETYPE OUT SERVICE
```

```
  0002  TS1
01
02               ;ACTIVE TASK CONTROL BLOCK POINTER
03
04 00012-000000 ATCB:   0       ;INITIALISED IN SYSTEM START UP
05
06               ;DEVICE 13 & 14
07
08 00013-177777         PTPS    ;PAPER TAPE PUNCH SERVICE
09 00014-177777         RTCS    ;REAL TIME CLOCK SERVICE
10
11               ;ADDRESS CONSTANTS
12
13 00015-177777 RTI:    RINT    ;RETURN FROM INTERRUPT
14
15               ;MORE ADDRESS CONSTANTS
16
17 00016-177777 TS:     TSCH    ;TASK SCHEDULER MAIN ENTRY
18 00017-177777 TS0:    TSCH0   ; "       " SUBROUTINE ENTRY
19 00020-177777 TS1:    TSCH1   ; "       " AUXILIARY ENTRY
20 00021-177777 TS2:    TSCH2   ; "       " SVC ENTRY
21 00022-177777 TS3:    TSCH3   ; "       " POWER RESTORE ENTRY
22 00023-177777 TS4:    TSCH4   ; "       " RESTORE AC2 ENTRY
23
24               ;DEVICES 24 TO 27
25
26 00024-177777         FL1S    ;FLOW METER 1 SERVICE
27 00025-177777         FL2S    ; "    "   2    "
28 00026-177777         FL3S    ; "    "   3    "
29 00027-177777         FL4S    ; "    "   4    "
30
31               ;USEFUL CONSTANTS
32
33 00030-000001 C1:     1
34 00031-000002 C2:     2
35 00032-000003 C3:     3
36 00033-000004 C4:     4
37 00034-000005 C5:     5
38 00035-000006 C6:     6
39 00036-000007 C7:     7
40 00037-000010 C10:    10
41 00040-000011 C11:    11      ;TAB
42 00041-000012 C12:    12      ;LINE-FEED
43 00042-000014 C14:    14      ;FORM-FEED
44 00043-000015 C15:    15      ;CARRIAGE-RETURN
45 00044-000040 C40:    40      ;SPACE
46 00045-000077 C77:    77
47 00046-000177 C177:   177
48 00047-000377 C377:   377
49
50               ;DEVICES 50 & 51
51
52 00050-177777         INIS    ;INFOTON IN SERVICE
53 00051-177777         INOS    ;INFOTON OUT SERVICE
54
55 00052-077777 CFR1:   77777   ;FRACTION 1
56 00053-100000 COMP:   100000  ;COMPLETION FLAG
57
58               .END
  0003  TS1

ATCB     000012-      2/04
```

BEGIN	000001$X	
C1	000030-	2/33
C10	000037-	2/40
C11	000040-	2/41
C12	000041-	2/42
C14	000042-	2/43
C15	000043-	2/44
C177	000046-	2/47
C2	000031-	2/34
C3	000032-	2/35
C377	000047-	2/48
C4	000033-	2/36
C40	000044-	2/45
C5	000034-	2/37
C6	000035-	2/38
C7	000036-	2/39
C77	000045-	2/46
CFR1	000052-	2/55
CL	000100	1/35
COMP	000053-	2/56
DST	000000-	1/42
FL1S	000024-X	2/26
FL2S	000025-X	2/27
FL3S	000026-X	2/28
FL4S	000027-X	2/29
INIS	000050-X	2/52
INOS	000051-X	2/53
NC	000010	1/36
PFLS	000000-X	1/42
PMASK	000001-	1/44
PTPS	000013-X	2/08
RINT	000015-X	2/13
RTCS	000014-X	2/09
RTI	000015-	2/13
SAV0	000002-	1/49
SAV1	000003-	1/50
SAV2	000004-	1/51
SAV3	000005-	1/52
SAVC	000006-	1/53
SAVR	000007-	1/54
TS	000016-	2/17
TS0	000017-	2/18
TS1	000020-	2/19
TS2	000021-	2/20
TS3	000022-	2/21
TS4	000023-	2/22
TSCH	000016-X	2/17
TSCH0	000017-X	2/18
TSCH1	000020-X	2/19
TSCH2	000021-X	2/20
TSCH3	000022-X	2/21
TSCH4	000023-X	2/22
TTIS	000010-X	1/58
TTOS	000011-X	1/59

```
0001  TS

              ; INTERRUPT HANDLER AND TASK SCHEDULER
              ; ----------------------------------

              ; XXX

              ; MK. V
              ; E. WULFF      8-APR-71

              ;VERSION B
              ;PROVIDES FOR THE FOLLOWING DEVICES:
              ;        00      ;POWER FAIL
              ;        10-24   ;MAIN GROUP OF DEVICES
              ;        50-51   ;2ND GROUP OF DEVICES
              ;IF ANY OTHER DEVICES ARE CONNECTED
              ;USE VERSION A.

              .TITL   TS

              .ENT    ATCB,RT,RTI,PMASK,COMP,C1,C12,C15,C77,C177,C377
              .ENT    SAV0,SAV1,SAV2,SAV3,SAVC,SAVR
              .ENT    TS,TS0,TS1,TS2,TS3,TS4,DST
              .ENT    TTIDS,TTODS,PTRDS,PTPDS,RTCDS,PLTDS,CDRDS,LPTDS
              .ENT    DSKDS,ADVDS,MTADS,DAVDS,DCMDS,INIDS,INODS
              .EXTD   FTP,PFLI,PFR
              .EXTN   PFLS,TTIS,TTOS,PTRS,PTPS,RTCS,PLTS,CDRS,LPTS
              .EXTN   DSKS,ADCVS,MTAS,DACVS,DCMS,INIS,INOS

       000000 .LOC    0       ;START SYSTEM AT LOCATION 0

00000 002003$L0:      JMP     @PFR    ;ENTER POWER FAIL RESTORE
00001 000000'         INTS            ;INTERRUPT VECTOR

              .NREL

              ;INTERRUPT SERVICE ROUTINE

              ;FOR SHORT INTERRUPT SERVICE STORE AC3
              ;IN ITS TASK CONTROL BLOCK , WHICH IS THE
              ;LOCATION POINTED TO BY THE ACTIVE TASK CONTROL
              ;BLOCK POINTER

00000'056005-INTS:    STA     3,@ATCB ;SAVE AC3 IN ACTIVE TCB
00001'075477          INTA    3       ;GET DEVICE NUMBER
00002'003400-         JMP     @DST,3  ;JMP VIA DEVICE SERVICE TABLE
```

```
0002  TS

              .ZREL

              ;DEVICE SERVICE TABLE

00000-177777 DST:    PFLS    ;POWER FAIL SERVICE

00001-177777 PMASK:  -1      ;INTERRUPT PRIORITY MASK
                             ;DEVICE 1 (MDV) CANNOT INTERRUPT.

      000003 .BLK 3  ;MEMORY ALLOCATION AND
                     ;PROTECTION SERVICE

              ;ACTIVE TASK CONTROL BLOCK POINTER

00005-000000 ATCB:   0       ;INITIALISED IN POWER FAIL ROUTINE

              ;ADDRESS CONSTANTS

00006-000103'RT:     RET
00007-000102'RTI:    RINT    ;RETURN FROM INTERRUPT

              ;MORE OF DEVICE SERVICE TABLE

00010-177777 TTIDS:  TTIS    ;TELETYPE IN SERVICE
00011-177777 TTODS:  TTOS    ;TELETYPE OUT SERVICE
00012-177777 PTRDS:  PTRS    ;PAPER TAPE READER SERVICE
00013-177777 PTPDS:  PTPS    ;PAPER TAPE PUNCH SERVICE
00014-177777 RTCDS:  RTCS    ;REAL TIME CLOCK SERVICE
00015-177777 PLTDS:  PLTS    ;INCREMENTAL PLOTTER SERVICE
00016-177777 CDRDS:  CDRS    ;CARD READER SERVICE
00017-177777 LPTDS:  LPTS    ;LINE PRINTER SERVICE

00020-177777 DSKDS:  DSKS    ;DISK SERVICE
00021-177777 ADVDS:  ADCVS   ;A/D CONVERTER SERVICE
00022-177777 MTADS:  MTAS    ;MAGNETIC TAPE SERVICE
00023-177777 DAVDS:  DACVS   ;D/A CONVERTER SERVICE
00024-177777 DCMDS:  DCMS    ;TTY DATA MULTIPLEXER SERV.

              ;USEFUL CONSTANTS

00025-000001 C1:     1
00026-000012 C12:    12      ;LINE-FEED
00027-000015 C15:    15      ;CARRIAGE RETURN
00030-000077 C77:    77
00031-000177 C177:   177
00032-000377 C377:   377
00033-100000 COMP:   100000  ;COMPLETION FLAG

              ;SOME TEMPORARY REGISTERS

00034-000000 SAV0:   0
00035-000000 SAV1:   0
00036-000000 SAV2:   0
00037-000000 SAV3:   0
00040-000000 SAVC:   0
00041-000000 SAVR:   0

              ;MORE ADDRESS CONSTANTS
```

```
0003  TS

00042-000005'TS:      TSCH    ;TASK SCHEDULER MAIN ENTRY
00043-000003'TS0:     TSCH0   ;  "      "  SUBROUTINE ENTRY
00044-000007'TS1:     TSCH1   ;  "      "  AUXILLIARY ENTRY
00045-000020'TS2:     TSCH2   ;  "      "  SUPERVISOR CALL ENTRY
00046-000050'TS3:     TSCH3   ;  "      "  POWER RESTORE ENTRY
00047-000101'TS4:     TSCH4   ;  "      "  RESTORE AC2 ENTRY

              ;2ND GROUP OF DEVICES

00050-177777 INIDS:   INIS    ;INFOTON KEYPOARD SERVICE
00051-177777 INODS:   INOS    ;INFOTON DISPLAY SEVICE
```

```
0004  TS
                ;TASK SCHEDULER
                .NREL
                ;SUBROUTINE ENTRY TO THE TASK SCHEDULER
                ;THIS ENTRY CAN BE USED FROM POSTING ROUTINES.
00003'060277 TSCH0:  INTDS
00004'054000         STA     3,0      ;SAVE RETURN
                ;MAIN TASK SCHEDULER ENTRY FROM INTERRUPT SERVICE
00005'034005-TSCH:   LDA     3,ATCB  ;GET ACTIVE CONTROL BLOCK
00006'051401         STA     2,TAC2,3;SAVE AC2
                ;DO TASK QUEUE SCAN NOW IN CASE THE PRESENT
                ;TASK IS OF HIGHEST PRIORITY.
                ;AUXILIARY ENTRY FROM WAIT ROUTINE
00007'030020 TSCH1:  LDA     2,20
00010'051410         STA     2,TL20,3;SAVE LOCATION 20
00011'030001$        LDA     2,FTP    ;POINTER TO HEAD OF QUEUE-1
00012'050020         STA     2,20     ;USE AUTO-INCREMENT
00013'032020 L1:     LDA     2,@20    ;GET QUEUE ENTRY AND INCREMENT
00014'151112         MOVL#   2,2,SZC ;IS IT READY
00015'000776         JMP     L1       ;NO , GET NEXT
                ;AC2 CONTAINS ADDRESS OF TCB THAT IS ACTIVATED NEXT
00016'156415         IFEQ    2,3      ;TEST IF ALREADY ACTIVE
00017'000460         JMP     TAA      ;YES
                ;ENTRY WITH NEW TCB ADDRESS IN AC2. MAINLY FROM
                ;SUPERVISOR CALL.
00020'045402 TSCH2:  STA     1,TAC1,3;SAVE REST OF STATUS
00021'041403         STA     0,TAC0,3;AC1 , 0
00022'020000         LDA     0,0      ;GET RETURN ADDRESS FROM LOC 0
00023'103200         ADDR    0,0      ;SUPERIMPOSE CARRY
00024'041404         STA     0,TPCC,3;SAVE RETURN ADDRESS AND CARRY
00025'020001-        LDA     0,PMASK
00026'041405         STA     0,TPM,3 ;SAVE PRIORITY MASK
00027'020006         LDA     0,6
00030'041406         STA     0,TL6,3 ;SAVE LOCATION 6
00031'020007         LDA     0,7
00032'041407         STA     0,TL7,3 ;SAVE LOCATION 7
00033'020021         LDA     0,21
00034'041411         STA     0,TL21,3;SAVE LOCATION 21
00035'020030         LDA     0,30
00036'041412         STA     0,TL30,3;SAVE LOCATION 30
00037'020031         LDA     0,31
```

```
 0005  TS
00040'041413           STA     0,TL31,3;SAVE LOCATION 31
00041'020040           LDA     0,40
00042'041414           STA     0,TL40,3;SAVE LOCATION 40
00043'020041           LDA     0,41
00044'041415           STA     0,TL41,3;SAVE LOCATION 41
00045'155005           MOV     2,3,SNR ;IS TCB POINTER ZERO
00046'002002$          JMP     @PFLI   ;YES- CONTINUE POWER FAIL
                                       ;INTERRUPT SERVICE
              ;SET UP STATUS OF NEW TASK
00047'054005-          STA     3,ATCB  ;STORE ACTIVE TASK CONTROL BLOCK
00050'021415 TSCH3:    LDA     0,TL41,3
00051'040041           STA     0,41    ;RESTORE LOCATION 41
00052'021414           LDA     0,TL40,3
00053'040040           STA     0,40    ;RESTORE LOCATION 40
00054'021413           LDA     0,TL31,3
00055'040031           STA     0,31    ;RESTORE LOCATION 31
00056'021412           LDA     0,TL30,3
00057'040030           STA     0,30    ;RESTORE LOCATION 30
00060'021411           LDA     0,TL21,3
00061'040021           STA     0,21    ;RESTORE LOCATION 21
00062'021407           LDA     0,TL7,3
00063'040007           STA     0,7     ;RESTORE LOCATION 7
00064'021406           LDA     0,TL6,3
00065'040006           STA     0,6     ;RESTORE LOCATION 6
00066'021405           LDA     0,TPM,3 ;RESTORE PRIORITY MASK
00067'040001-          STA     0,PMASK
00070'062077           MSKO    0       ;SET UP MASK
00071'021404           LDA     0,TPCC,3
00072'105142           MOVOL   0,1,SZC ;RESTORE CARRY, SET 1 IN BIT 15
00073'121220           MOVZR   1,0     ;C=1, USE BIT 15 TO LEAVE C=1
00074'040000           STA     0,0     ;RESTORE PC IN LOC 0
00075'021403           LDA     0,TAC0,3;RESTORE ACCUMULATORS
00076'025402           LDA     1,TAC1,3
00077'031410 TAA:      LDA     2,TL20,3
00100'050020           STA     2,20    ;RESTORE LOCATION 20
00101'031401 TSCH4:    LDA     2,TAC2,3
00102'036005-RINT:     LDA     3,@ATCB ;RETURN FROM INTERRUPT
00103'060177 RET:      INTEN
00104'002000           JMP     @0      ;RETURN TO USER PROGRAM
       000000 .END     L0      ;START SYSTEM AT LOCATION 0
```

0006 TS

ADCVS	000021-X	2/35				
ADVDS	000021-	2/35				
ATCB	000005-	1/43	2/16	4/14	5/13	5/48
C1	000025-	2/42				
C12	000026-	2/43				
C15	000027-	2/44				
C177	000031-	2/46				
C377	000032-	2/47				
C77	000030-	2/45				
CDRDS	000016-	2/31				
CDRS	000016-X	2/31				
COMP	000033-	2/48				
DACVS	000023-X	2/37				
DAVDS	000023-	2/37				
DCMDS	000024-	2/38				
DCMS	000024-X	2/38				
DSKDS	000020-	2/34				
DSKS	000020-X	2/34				
DST	000000-	1/45	2/06			
FTP	000001$X	4/25				
INIDS	000050-	3/11				
INIS	000050-X	3/11				
INODS	000051-	3/12				
INOS	000051-X	3/12				
INTS	000000'	1/32	1/43			
L0	000000	1/31	5/53			
L1	000013'	4/28	4/30			
LPTDS	000017-	2/32				
LPTS	000017-X	2/32				
MTADS	000022-	2/36				
MTAS	000022-X	2/36				
PFLI	000002$X	5/08				
PFLS	000000-X	2/06				
PFR	000003$X	1/31				
PLTDS	000015-	2/30				
PLTS	000015-X	2/30				
PMASK	000001-	2/08	4/47	5/32		
PTPDS	000013-	2/28				
PTPS	000013-X	2/28				
PTRDS	000012-	2/27				
PTRS	000012-X	2/27				
RET	000103'	2/20	5/50			
RINT	000102'	2/21	5/48			
RT	000006-	2/20				
RTCDS	000014-	2/29				
RTCS	000014-X	2/29				
RTI	000007-	2/21				
SAV0	000034-	2/52				
SAV1	000035-	2/53				
SAV2	000036-	2/54				
SAV3	000037-	2/55				
SAVC	000040-	2/56				
SAVR	000041-	2/57				
TAA	000077'	4/35	5/43			
TS	000042-	3/02				
TS0	000043-	3/03				
TS1	000044-	3/04				
TS2	000045-	3/05				
TS3	000046-	3/06				

0007 TS

TS4	000047-	3/07	
TSCH	000005'	3/02	4/14
TSCH0	000003'	3/03	4/09
TSCH1	000007'	3/04	4/22
TSCH2	000020'	3/05	4/40
TSCH3	000050'	3/06	5/15
TSCH4	000101'	3/07	5/46
TTIDS	000010-	2/25	
TTIS	000010-X	2/25	
TTODS	000011-	2/26	
TTOS	000011-X	2/26	

```
 0001  PFAIL

            ; POWER FAIL AND RESTART SERVICE
            ; ----------------------------

            ; TASK SCHEDULER MK. V

            ; E. WULFF      8-APR-71
            ; MODIFIED      17-MAR-72

            .TITL   PFAIL

            .ENT    PFLS,PFR,PFLI,DSW
            .EXTD   RTI,ATCB,TS2,TS3
            .EXTN   RTCW

            .ZREL

00000-000034'PFR:       PFRST   ;POWER FAIL RESTART ADDRESS
00001-000006'PFLI:      PFLIN   ;POWER FAIL INTERRUPT SERVICE
00002-000067'DSW:       DSWT    ;ADDRESS OF 4 DEVICE STATUS WORDS.

            .NREL

            ;ENTER FROM INTERRUPT HANDLER WITH
            ;INTERRUPT DISENABLED

00000'063677 PFLS:      SKPDN   CPU       ;TEST POWER FAIL FLAG
00001'002001$           JMP     @RTI      ;SPURIOUS INTERRUPT

00002'034002$           LDA     3,ATCB
00003'051401            STA     2,TAC2,3;SAVE AC2
00004'152460            CLA     2,2       ;DUMMY TCB ADDRESS FOR T.S.
00005'002003$           JMP     @TS2      ;STORE STATUS OF ACTIVE TASK
                                          ;AND RETURN TO PFLIN

00006'020454 PFLIN:     LDA     0,INST+1;SET UP SKP INSTRUCTIONS
00007'040406            STA     0,L3+1
00010'020453            LDA     0,INST+2
00011'040405            STA     0,L3+2
00012'030002-           LDA     2,DSW
00013'034453            LDA     3,M4      ;-4

00014'102620 L3:        SUBZR   0,0       ;INITIALISE 100000
00015'063400            SKPBN   0
00016'063700            SKPDZ   0
00017'101241            MOVOR   0,0,SKP ;SET BIT FOR BUSY OR DONE =1
00020'101220            MOVZR   0,0       ;CLEAR BIT FOR BOTH 0
00021'010774            ISZ     .-4
00022'010774            ISZ     .-4
00023'101003            MOV     0,0,SNC ;TEST FOR END OF WORD
00024'000771            JMP     L3+1
00025'041000            STA     0,0,2     ;SAVE IN DSW
00026'151400            INC     2,2       ;NEXT DSW
00027'175404            INC     3,3,SZR ;IS IT THE LAST?
00030'000764            JMP     L3        ;NO

00031'020433            LDA     0,PFRI    ;PICK UP JMP INSTRUCTION
00032'040000            STA     0,0       ;POWER RESTART STARTS AT LOC 0
00033'063077            HALT              ;SHUTDOWN
```

```
 0002  PFAIL
0003  PFAIL

                      ; POWER FAIL RESTART.

                      ; ENTER HERE FROM LOC. 0

 00034'062677 PFRST:   IORST             ;CLEAR ALL DEVICE FLAGS
 00035'020424          LDA      0,INST   ;SET UP NIOS INSTRUCTION
 00036'040412          STA      0,L2+1
 00037'022426          LDA      0,@RTCP
 00040'061014          DOA      0,RTC    ;SET CLOCK FREQUENCY
 00041'030002-         LDA      2,DSW
 00042'034424          LDA      3,M4

 00043'021000 L1:      LDA      0,0,2    ;DEVICE STATUS WORD
 00044'025004          LDA      1,4,2    ;DEVICES ALLOWED
 00045'123400          AND      1,0      ;ELIMINATE UNWANTED DEVICES
 00046'126620          SUBZR    1,1
 00047'101222 L2:      MOVZR    0,0,SZC  ;TEST DEVICE STATUS WORD
 00050'060100          NIOS     0        ;START DEVICE
 00051'010777          ISZ      .-1      ;NEXT DEVICE
 00052'125223          MOVZR    1,1,SNC  ;COUNT
 00053'000774          JMP      L2
 00054'151400          INC      2,2
 00055'175404          INC      3,3,SZR
 00056'000765          JMP      L1

 00057'034002$         LDA      3,ATCB   ;START TASK THAT WAS INTERRUPTED
 00060'002004$         JMP      @TS3     ;BY POWER FAIL

 00061'060100 INST:    NIOS     0
 00062'063400          SKPBN    0
 00063'063700          SKPDZ    0
 00064'002000-PFRI:    JMP      @PFR
 00065'177777 RTCP:    RTCW

 00066'177774 M4:      -4

 00067'000000 DSWT:    0        ;DEVICE STATUS WORDS
 00070'000000          0        ;INITIALIZED TO ZERO
 00071'000000          0
 00072'000000          0
 00073'177400          177400   ;UNWANTED DEVICES - 0 TO 7
 00074'177777          177777
 00075'177777          177777
 00076'077777          077777   ; AND 77

                      .END
```

0004 PFAIL

ATCB	000002$X	1/30	3/27			
DSW	000002-	1/20	1/40	3/11		
DSWT	000067'	1/20	3/38			
INST	000061'	1/36	1/38	3/07	3/30	
L1	000043'	3/14	3/25			
L2	000047'	3/08	3/18	3/22		
L3	000014'	1/37	1/39	1/43	1/51	1/55
M4	000066'	1/41	3/12	3/36		
PFLI	000001-	1/19				
PFLIN	000006'	1/19	1/36			
PFLS	000000'	1/27				
PFR	000000-	1/18	3/33			
PFRI	000064'	1/57	3/33			
PFRST	000034'	1/18	3/06			
RTCP	000065'	3/09	3/34			
RTCW	000065'X	3/34				
RTI	000001$X	1/28				
TS2	000003$X	1/33				
TS3	000004$X	3/28				

```
0001  SVC

               ;SUPERVISOR CALL
               ;--------------

               ;TASK SCHEDULER  MK. V

               ;E. WULFF        11-APR-71
               ;MODIFIED        4-MAR-72

               .TITL   SVC

               ;THIS SEGMENT OF CODE IS ENTERED FROM APPLICATION
               ;PROGRAMS WHEN THEY REQUIRE SUPERVISOR SERVICE.

               ;THE MODULES OF THE SUPERVISOR ARE ARRANGED AS
               ;SEPERATE TASKS. AS SUCH THEY CAN ONLY BE ACTIVATED
               ;BY THE TASK SCHEDULER.

               ;CALLING SEQUENCE:
               ;       .SVC
               ;       (TCB ADDRESS OF SUPERVISOR MODULE) OR @(...)
               ;       (ERROR RETURN (IF REQUIRED))
               ;       (PARAMETERS REQUIRED (IF ANY))
               ;       (NEXT STATEMENT)

               ;THE FOLLOWING CODE WILL STORE THE RETURN LOCATION
               ;OF THE CALLING PROGRAM, SO THAT IT IS RESTARTED
               ;AFTER SERVICE IS COMPLETE. ALSO THE ADDRESS OF THE
               ;TCB OF THE CALLING PROGRAM IS STORED IN AC3 OF THE
               ;CALLED SUPERVISOR MODULE. THIS ALLOWS THE SUPERVISOR
               ;MODULE FULL ACCESS TO THE STATUS OF THE CALLING PROGRAM.
               ;THE SUPERVISOR MODULE IS MADE ACTIVE BY USING THE
               ;TCB ADDRESS IN THE WORD AFTER THE CALL. IF "@" IS USED
               ;BEFORE THE TCB ADDRESS, THE SAME TCB IS MADE ACTIVE,
               ;BUT THE SUPERVISOR MODULE MAY TEST THE BIT AND USE
               ;IT AS A SWITCH.

               .ENT    .SVC,.EXIT
               .EXTD   TS,TS2,ATCB

               .ZREL

 00000-000000'SVC:     RSVC
        006000-.SVC=   JSR     @SVC     ;DEFINE CALLING MNEMONIC

               .NREL

 00000'060277 RSVC:    INTDS
 00001'056003$         STA     3,@ATCB ;SAVE CALLING ADDRESS
 00002'054000          STA     3,0      ;ALSO IN LOC 0
 00003'034003$         LDA     3,ATCB  ;GET CALLING TCB ADDRESS
 00004'051401          STA     2,TAC2,3;SAVE AC2 IN CALLING TCB
 00005'032000          LDA     2,@0     ;GET .SVC PARAMETER, NEW TCB
 00006'010000          ISZ     0        ;INCREMENT RETURN
 00007'151100          MOVL    2,2      ;CLEAR BIT 0
 00010'151220          MOVZR   2,2
 00011'055000          STA     3,TAC3,2;STORE OLD TCB IN AC3 OF NEW
 00012'053017          STA     2,@TBP,2;ACTIVATE NEW TASK
 00013'002002$         JMP     @TS2     ;ENTER TASK SCHEDULER
```

```
 0003  SVC
01
02              ;EXIT FROM SUPERVISOR
03              ;-------------------
04
05              ;WHEN SUPERVISOR SERVICE IS COMPLETE THE
06               ;MODULE EXECUTES THIS CALL.
07
08              ; CALLING SEQUENCE:
09              ;       .EXIT
10              ;       (NEXT STATEMENT)
11
12              ;THIS CODE PUTS THE CALLING MODULE INTO THE
13              ;WAIT STATE AND PASSES CONTROL TO THE TASK
14              ;SCHEDULER. THE MODULE CAN ONLY BE RESTARTED
15              ;BY A '.SVC' FOR THAT MODULE. THEN IT WILL
16              ;BE STARTED AT THE LOCATION FOLLOWING THE LAST
17              ;'.EXIT' THAT WAS EXECUTED.
18
19              ;  SUPERVISOR MODULES CANNOT BE CALLED
20              ;RE-ENTRANTLY. THEY ARE TASKS AND THERE MAY
21              ;BE SEVERAL MODULES WITH THE SAME RE-ENTRANT
22              ;PROGRAM, BUT DIFFERENT TCB'S. ALL SUPERVISOR
23              ;MODULES SHOULD BE OF HIGHER PRIORITY ON THE
24              ;TASK SCHEDULER THAN PROGRAMS MAKING CALLS ON IT.
25              ;THIS ENSURES THAT THEY ARE NOT CALLED RE-ENTRANTLY.
26
27              ;DESTROYED:   AC3 ONLY
28
29              .ZREL
30
31 00001-000014'EXIT:   REXIT
32       006001-.EXIT=  JSR     @EXIT    ;DEFINE CALLING MNEMONIC
33
34              .NREL
35
36  00014'060277  REXIT:  INTDS
37 00015'054000         STA     3,0      ;STORE RETURN POINT
38 00016'034003$        LDA     3,ATCB   ;GET TCB OF CALL
39 00017'175100         MOVL    3,3
40 00020'175240         MOVOR   3,3      ;SET BIT 0
41 00021'057417         STA     3,@TBP,3;DE-ACTIVATE TASK
42 00022'002001$        JMP     @TS      ;ENTER TASK SCHEDULER
43
44              .END
  0004  SVC

ATCB     000003$X    1/49    1/51    3/38
EXIT     000001-     3/31    3/32
REXIT    000014'     3/31    3/36
RSVC     000000'     1/43    1/48
SVC      000000-     1/43    1/44
TS       000001$X    3/42
TS2      000002$X    1/59
.EXIT    006001-     3/32
.SVC     006000-     1/44
```

```
0001  WT
                   ;WAIT ON CONDITION
                   ;----------------
                   ; XXX
                   ;MK. V
                   ;E. WULFF       8-APR-71
                   ;THE CONDITION IS SIGNALLED BY AN
                   ;EVENT CONTROL WORD WHOSE ADDRESS IS IN THE
                   ;WORD AFTER THIS CALL. RETURN TO THE STATEMENT
                   ;AFTER THAT WHEN THE CONDITION IS POSTED. THIS MAY
                   ;OCCUR STRAIGHT AWAY OR DE-ACTIVATE THE TASK
                   ;UNTIL IT IS POSTED BY THE CONDITION.
                   ;CALLING SEQUENCE:
                   ;       .WAIT   OR   JSR @WAIT
                   ;       (ECW ADDRESS)
                   ;       SUBC    3,3      ;CLEAR THE
                   ;       STA     3,@.-2   ;EVENT CONTROL WORD
                   ;CLEARING OF THE EVENT CONTROL WORD IS ESSENTIAL
                   ;TO THE SUCCESSFUL OPERATION OF THIS METHOD OF TASK
                   ;SYNCHRONISATION. IT MAY ONLY BE DEFERRED UNTIL ANOTHER
                   ;WAIT STATEMENT. MANY SUBROUTINES CONTAIN
                   ;WAIT STATEMENTS AND THE CALLING SEQUENCE ABOVE
                   ;IS A SAFE AND RECOMMENDED PROCEDURE.
                   ;DESTROYED:     AC3 ONLY
                   .TITL   WT
                   .ENT    WAIT,.WAIT
                   .EXTD   ATCB,TS1,C1
                   .ZREL
 00000-000000'WAIT:   W1
       006000-.WAIT=  JSR     @WAIT    ;DEFINE CALLING MNEMONIC
                   .NREL
 00000'060277 W1:     INTDS
 00001'175400         INC     3,3
 00002'056001$        STA     3,@ATCB ;STORE RETURN IN TCB
 00003'037777         LDA     3,@-1,3 ;GET CONTENTS OF ECW
 00004'175112         MOVL#   3,3,SZC ;IS EVENT ALREADY POSTED
 00005'000414         JMP     W2      ;YES
 00006'034001$        LDA     3,ATCB  ;GET ACTIVE TCB POINTER
 00007'051401         STA     2,TAC2,3;SAVE AC2 IN TCB
 00010'031400         LDA     2,TAC3,3;GET SAVED RETURN IN AC2
 00011'050000         STA     2,0     ;SAVE RETURN IN LOCATION 0
 00012'057377         STA     3,@-1,2 ;STORE TCB ADDRESS IN ECW
                                      ;BIT 0 IS CLEARED
 00013'175100         MOVL    3,3
 00014'175240         MOVOR   3,3     ;SET BIT 0, DONT DISTURB CARRY
 00015'057417         STA     3,@TBP,3;USE BACKPOINTER TO STORE
```

```
  0002  WT
01              ;TCB ADDRESS WITH BIT 0 SET IN TASK QUEUE.
02              ;THIS DE-ACTIVATES THE TASK FOR THE TASK SCHEDULER
03
04 00016'030003$        LDA     2,C1     ;+1
05 00017'051416         STA     2,TWC,3 ;SET WAIT COUNT TO 1
06
07 00020'002002$        JMP     @TS1     ;ENTER MAIN TASK SCHEDULER
08
09 00021'036001$W2:     LDA     3,@ATCB
10 00022'060177         INTEN
11 00023'001400         JMP     0,3      ;RETURN TO WAITING TASK
12
13              ;THIS PROGRAM WILL ONLY WORK IN ASSOCIATION WITH OTHER
14              ;PROGRAMS WHICH POST THE CONDITION AND CLEAR THE ECW.
15
16
17              .END
```

```
  0003  WT

ATCB    000001$X    1/47    1/51    2/09
C1      000003$X    2/04
TS1     000002$X    2/07
W1      000000'     1/40    1/45
W2      000021'     1/50    2/09
WAIT    000000-     1/40    1/41
.WAIT   006000-     1/41
```

```
0001  MWT

              ;WAIT ON MULTIPLE CONDITIONS
              ;--------------------------

              ; XXX

              ;TASK SCHEDULER  MK. V

              ;E. WULFF        20-AUG-71

              ;THE CONDITIONS ARE SIGNALLED BY ONE OR MORE
              ;EVENT CONTROL WORDS WHOSE ADDRESSES FOLLOW
              ;THE CALL. THE LIST OF EVENT CONTROL WORDS
              ;IS TERMINATED BY A NEGATIVE NUMBER WHOSE TWO'S
              ;COMPLEMENT INDICATES HOW MANY OF THE CONDITIONS
              ;SHOULD BE POSTED BEFORE CONTROL IS RETURNED
              ;TO THE TASK CONTAINING THE CALL. EVENT CONTROL
              ;WORDS SHOULD BE CLEARED AS SOON AS CONTROL
              ;IS RETURNED.

              ;CALLING SEQUENCE:
              ;       .MWAIT   OR   JSR @MWAIT
              ;       ECW1
              ;       ECW2
              ;       ....
              ;       ECWN
              ;       -M
              ;       SUBC 3,3        ;CLEAR ECW'S
              ;       STA 3,@.-N-2
              ;       STA 3,@.-N-2
              ;       ....
              ;       STA 3,@.-N-2
              ;       NEXT STATEMENT

              ;THIS CALL IS INTERPRETED AS:
              ;WAIT FOR "M" OUT OF THE "N" EVENTS LISTED

              ;DESTROYED:     AC3 ONLY

              ;SUBROUTINES NEEDED:
              ;       SAVAC,RESAC

              .TITL   MWT

              .ENT    .MWAIT,MWAIT
              .EXTD   ATCB,TS1,SAVAC,RESAC

              .ZREL

 00000-000000'MWAIT:  MWT
       006000-.MWAIT= JSR     @MWAIT

              .NREL

 00000'060277 MWT:    INTDS           ;INTERRUPT MUST BE OFF
 00001'056001$        STA     3,@ATCB ;SAVE AC3
 00002'006003$        JSR     @SAVAC  ;SAVE AC0,AC2 AND CARRY
 00003'161000         MOV     3,0     ;MOVE TCB ADDRESS TO AC0
 00004'126460         CLA     1,1
```

```
 0002  MWT
01 00005'044433          STA     1,PCNT  ;ZERO POST COUNT
02 00006'036001$         LDA     3,@ATCB ;RETURN POINTER
03
04              ;PICK UP THE ECW'S AND TEST FOR WAIT COUNT
05
06 00007'175400 NPAR:    INC     3,3     ;INCREMENT PARAMETER POINTER
07 00010'031777          LDA     2,-1,3  ;GET NEXT PARAMETER
08 00011'151112          IFM     2,2     ;IS IT WAIT COUNT
09 00012'000410          JMP     LPCNT   ;YES
10 00013'025000          LDA     1,0,2   ;NO, GET ECW
11 00014'125112          IFM     1,1     ;IS EVENT ALREADY POSTED
12 00015'000403          JMP     .+3     ;YES
13 00016'041000          STA     0,0,2   ;NO, STORE TCB ADDR. IN ECW
14 00017'000770          JMP     NPAR
15
16 00020'014420          DSZ     PCNT    ;DECREMENT POST COUNT
17 00021'000766          JMP     NPAR
18
19              ;WAIT COUNT HAS BEEN FOUND
20
21 00022'054000 LPCNT:   STA     3,0     ;SAVE RETURN IN LOC 0
22 00023'115000          MOV     0,3     ;MOVE TCB ADDRESS TO AC3
23 00024'024414          LDA     1,PCNT  ;GET POST COUNT
24 00025'146400          SUB     2,1     ;WAIT FOR THIS NO. OF EVENTS
25 00026'124537          IFZM    1,1     ;TEST IF ALREADY ENOUGH
26 00027'002004$         JMP     @RESAC  ;YES, RESTART
27 00030'045416          STA     1,TWC,3 ;NO, STORE WAIT COUNT
28 00031'177240          ADDOR   3,3     ;SET BIT 0
29 00032'057417          STA     3,@TBP,3;DE-ACTIVATE TASK VIA BACK P.
30 00033'021404          LDA     0,TPCC,3
31 00034'101120          MOVZL   0,0     ;RESTORE CARRY
32 00035'021403          LDA     0,TAC0,3
33 00036'025402          LDA     1,TAC1,3;RESTORE AC0 AND AC1
34 00037'002002$         JMP     @TS1    ;ENTER TASK SCHEDULER.
35
36 00040'000000 PCNT:    0
37
38              .END
```

```
 0003  MWT

ATCB    000001$X   1/56   2/02
LPCNT   000022'    2/09   2/21
MWAIT   000000-    1/50   1/51
MWT     000000'    1/50   1/55
NPAR    000007'    2/06   2/14   2/17
PCNT    000040'    2/01   2/16   2/23   2/36
RESAC   000004$X   2/26
SAVAC   000003$X   1/57
TS1     000002$X   2/34
.MWAI   006000-    1/51
```

```
0001  SAC

            ;SAVE AND RESTORE STATUS
            ;----------------------

            ; XXX

            ;TASK SCHEDULER  MK. V

            ;E. WULFF        11-APR-71

            ;TWO ROUTINES THAT CAN BE USED BY SUPERVISOR
            ;MODULES TO FREE THE HARDWARE STATUS TEMPORARILY.
            ;INTERRUPT MUST BE OFF WHEN SAVAC AND RESAC
            ;ARE CALLED.
            ;SAVE AND RESTORE AC0,AC1,AC2 AND CARRY
            ;IN THE CURRENTLY ACTIVE TASK CONTROL
            ;BLOCK.

            ;CALLING SEQUENCE:
            ;       JSR @SAVAC
            ;       (NEXT STATEMENT)
            ;AC3 MUST BE SAVED INDEPENDENTLY.

            .TITL   SAC

            .ENT    SAVAC,RESAC
            .EXTD   ATCB,TS4,SAVR

            .ZREL

00000-000000'SAVAC:  RSAVA

            .NREL

00000'054003$RSAVA:  STA     3,SAVR   ;SAVE RETURN
00001'034001$        LDA     3,ATCB   ;GET ACTIVE TCB ADDRESS
00002'051401         STA     2,TAC2,3
00003'045402         STA     1,TAC1,3
00004'041403         STA     0,TAC0,3;SAVE AC0 TO AC2
00005'102660         SUBCR   0,0
00006'041404         STA     0,TPCC,3;SAVE CARRY
00007'002003$        JMP     @SAVR
```

```
 0002   SAC
01
02              ;RESTORE STATUS AND RETURN.
03
04              ;CALLING SEQUENCE:
05              ;       JMP @RESAC
06
07              ;THIS CALL RESTORES AC0 TO AC3 AND CARRY
08              ;FROM THE ACTIVE TASK CONTROL BLOCK.
09              ;IT THEN TURNS ON INTERRUPT AND RETURNS
10              ;CONTROL TO THE  ADDRESS IN LOC 0.
11
12              .ZREL
13
14 00001-000010'RESAC:  RRAC
15
16              .NREL
17
18 00010'034001$RRAC:   LDA     3,ATCB  ;ACTIVE TCB POINTER
19 00011'021404         LDA     0,TPCC,3
20 00012'101120         MOVZL   0,0     ;RESTORE CARRY
21 00013'021403         LDA     0,TAC0,3
22 00014'025402         LDA     1,TAC1,3
23 00015'002002$        JMP     @TS4    ;RESTORE AC0 TO AC3
24
25              .END
```

```
  0003   SAC

ATCB     000001$X    1/36    2/18
RESAC    000001-     2/14
RRAC     000010'     2/14    2/18
RSAVA    000000'     1/31    1/35
SAVAC    000000-     1/31
SAVR     000003$X    1/35    1/42
TS4      000002$X    2/23
```

```
  0001  POST
01
02              ; POST AN EVENT
03              ; -------------
04
05              ; XXX
06
07              ; TASK SCHEDULER  MK. V
08
09              ; E. WULFF      11-APR-71
10
11              ; THE EVENT IS POSTED TO ANOTHER TASK VIA AN EVENT
12              ; CONTROL WORD (ECW). IF THE OTHER TASK IS WAITING
13              ; ON THIS EVENT, THE TASK IS ACTIVATED AND THE TASK
14              ; SCHEDULER IS ENTERED. IF THE WAITING TASK IS OF
15              ; HIGHER PRIORITY THAN THE TASK CONTAINING THE CALL,
16              ; THE WAITING TASK WILL BE EXECUTED NEXT. OTHERWISE,
17              ; OR IF THE OTHER TASK IS NOT WAITING, THE STATEMENT
18              ; AFTER THE CALL IS EXECUTED NEXT. IN ALL CASES THE
19              ; COMPLETION BIT IN THE EVENT CONTROL WORD IS SET.
20              ; MULTIPLE POSTINGS ON THE SAME ECW ARE ALLOWED.
21              ; ALL BUT THE LAST POSTING ARE IGNORED.
22
23
24              ; CALLING SEQUENCE:
25              ;       .POST   OR   JSR @POST
26              ;       (ECW ADDRESS) OR @(POINTER TO ECW ADDRESS)
27              ;       (NEXT STATEMENT)
28
29              ; DESTROYED:   AC3 ONLY
30
31              ; OTHER ROUTINES REQUIRED:   POSTI
32
33              ;                  NOTE
34              ; THE ROUTINE "POSTI" MUST ALWAYS BE LOADED
35              ; IMMEDIATELY AFTER THIS ONE, BECAUSE THIS ROUTINE
36              ; EXPECTS TO ENTER IT AFTER THE LAST STATEMENT.
37
38              .TITL   POST
39
40              .ENT    .POST,POST
41              .EXTD   POSTI
42
43              .ZREL
44
45 00000-000000'POST:   RPOST
46        006000-.POST= JSR     @POST    ; DEFINE CALLING MNEMONIC
47
48              .NREL
49
50 00000'060277 RPOST:  INTDS            ; DISENABLE INTERRUPTS
51 00001'054000         STA     3,0      ; SET UP RETURN
52 00002'010000         ISZ     0        ; TO MISS PARAMETER
53
54              ; AT THIS POINT ENTER "POSTI"
55
56              .END
POST    000000-     1/45    1/46
POSTI   000001$X
RPOST   000000'     1/45    1/50
.POST   006000-     1/46
```

```
 0001  POSTI

                ; POST AN EVENT IN AN INTERRUPT SERVICE ROUTINE
                ; ---------------------------------------------

                ; XXX

                ; TASK SCHEDULER  MK. V

                ; E. WULFF      11-APR-71

                ; ROUTINE FOR POSTING AN EVENT IN AN INTERRUPT
                ; SERVICE ROUTINE TO A TASK.

                ; NOTE: INTERRUPT MUST BE DISENABLED.
                ;       ALL REGISTERS EXCEPT AC3 MUST BE RESTORED
                ;       TO THE VALUE THEY HAD WHEN THE INTERRUPT
                ;       OCCURRED, BEFORE THIS CALL IS MADE.
                ;       THIS ROUTINE DOES NOT RETURN TO THE CALLER.

                ; CALLING SEQUENCE:
                ;       JSR @POSTI
                ;       (ECW ADDRESS) OR @ (POINTER TO ECW ADDRESS)

                ; THE BODY OF THIS ROUTINE IS ALSO USED BY "POST"

                ; SEE "POST" FOR DESCRIPTION OF POSTING OPERATION.

                ; DESTROYED:   AC3 ONLY AND THIS IS RESTORED
                ; FROM THE TASK CONTROL BLOCK.

                .TITL   POSTI

                .ENT    POSTI
                .EXTD   TS,RTI,COMP,SAV2

                .ZREL

00000-000000'POSTI:     RPOSI

                .NREL

00000'050004$RPOSI:     STA     2,SAV2  ; SAVE AC2
00001'033400            LDA     2,@0,3  ;  CONTENTS OF ECW
00002'151134            MOVZL#  2,2,SZR ; TEST ECW
00003'015016            DSZ     TWC,2   ; TEST WAIT COUNT
00004'000406            JMP     POSI

00005'053017            STA     2,@TBP,2; STORE TCB ADDRESS IN
00006'030003$           LDA     2,COMP  ; TASK QUEUE VIA BACK POINTER
00007'053400            STA     2,@0,3  ; SET COMPLETE FLAG IN ECW
00010'030004$           LDA     2,SAV2  ; RESTORE AC2
00011'002001$           JMP     @TS     ; ENTER TASK SCHEDULER

00012'030003$POSI:      LDA     2,COMP
00013'053400            STA     2,@0,3  ; SET COMPLETE FLAG IN ECW
00014'030004$           LDA     2,SAV2  ; RESTORE AC2
00015'002002$           JMP     @RTI

                .END
```

0002 POSTI

COMP	000003$X	1/49	1/54	
POSI	000012'	1/46	1/54	
POSTI	000000-	1/38		
RPOSI	000000'	1/38	1/42	
RTI	000002$X	1/57		
SAV2	000004$X	1/42	1/51	1/56
TS	000001$X	1/52		

```
; SEMAPHORE OPERATIONS "LOWER" AND "RAISE"
; ---------------------------------------

; XXX

; E. WULFF      6-APR-71

.TITL   SEM

.ENT    LOWER,RAISE
.EXTD   RTI,TS1,ATCB,SAV2,

; SEMAPHORES MUST BE DEFINED AS A 2 WORD BLOCK.
; THE FIRST WORD IS THE ACTUAL SEMAPHORE COUNTER.
; THE SECOND WORD IS USED TO LINK TO TASK CONTROL
; BLOCKS WHEN TASKS ARE WAITING FOR A SEMAPHORE.
; CARE MUST BE TAKEN TO INITIALISE A SEMAPHORE
; CORRECTLY. THE FIRST WORD SHOULD CONTAIN THE
; NUMBER OF LOWER OPERATIONS WHICH ARE TO BE ALLOWED
; BEFORE THE SEMAPHORE SUSPENDS A TASK. THIS
; VALUE IS USUALLY 1 FOR A BINARY SEMAPHORE WHICH IS
; OPEN. THE VALUE IS 0 IF THE SEMAPHORE IS TO BE
; BLOCKING. THE VALUE SHOULD NEVER BE INITIALISED AS A
; NEGATIVE VALUE. THE SECOND WORD OF THE SEMAPHORE
; SHOULD ALWAYS BE INITIALISED TO 0.
; APART FROM INITIALISATION A SEMAPHORE SHOULD ONLY
; BE OPERATED ON BY THE "LOWER" AND "RAISE" OPERATIONS.

; REFERENCES:
;    DIJKSTRA, E. W.  COOPERATING SEQUENTIAL PROCESSES.
;       TECHNICAL U. EINDHOVEN, NETHERLANDS, 1966.

;    WIRTH, N.  ON MULTIPROGRAMMING, MACHINE CODING,
;       AND COMPUTER ORGANIZATION.
;       COMM. ACM 12, 9 (SEPT. 69), 489-498.


;               ***************
;               *             *
;               *    LOWER    *
;               *             *
;               ***************

; CALLING SEQUENCE:
;    LOWER
;    (ADDRESS OF SEM) OR @(POINTER TO ADDRESS OF SEM)
;    (NEXT STATEMENT)

; OPERATION:
;    THE SEMAPHORE IS DECREMENTED. IF THE RESULT
;    IS POSITIVE OR ZERO THE NEXT STATEMENT IS
;    EXECUTED IMMEDIATELY. OTHERWISE THE PRESENT TASK
;    (PROCESS) IS SUSPENDED UNTIL A "RAISE" ON THE
;    SAME SEMAPHORE RE-ACTIVATES THE TASK.

; DESTROYED:    AC3 ONLY

; THE "LOWER" OPERATION CORRESPONDS TO THE "P" OPERATION
; DESCRIBED BY E. W. DIJKSTRA.
```

```
0003  SEM

                 ; LOWER A SEMAPHORE

                 .ZREL

00000-000000'P:         RP
       006000-LOWER=    JSR     @P        ; DEFINE CALLING MNEMONIC

                 .NREL

00000'060277 RP:        INTDS             ; PREVENT INTERRUPTS
00001'054000            STA     3,0       ; SAVE RETURN

00002'017400            DSZ     @0,3      ; DECREMENT THE SEM. COUNTER
00003'000401            JMP     .+1       ; COULD BE 0
00004'037400            LDA     3,@0,3    ; GET COUNTER VALUE
00005'175112            IFM     3,3
00006'000403            JMP     P0        ; NEGATIVE

00007'010000            ISZ     0         ; ZERO OR POSITIVE
00010'002001$           JMP     @RTI      ; RETURN IMMEDIATELY

00011'034003$P0:        LDA     3,ATCB    ; ACTIVE TCB ADDRESS
00012'051401            STA     2,TAC2,3; SAVE AC2
00013'034000            LDA     3,0       ; GET CALLING ADDRESS
00014'010000            ISZ     0         ; INCREMENT RETURN
00015'035400            LDA     3,0,3     ; GET SEMAPHORE ADDRESS
00016'175112            IFM     3,3       ; IS IT ONLY POINTER
00017'000776            JMP     .-2       ; YES- TRY AGAIN

00020'031401            LDA     2,1,3     ; SEMAPHORE LINK WORD
00021'151015            IFZ     2,2       ; IS IT ZERO
00022'000415            JMP     P3        ; YES

00023'155000 P1:        MOV     2,3       ; NO- A TCB ADDRESS
00024'031416            LDA     2,TWC,3   ; NEXT LINK
00025'151014            IFN     2,2       ; IS IT ZERO
00026'000775            JMP     P1        ; NO

00027'030003$           LDA     2,ATCB    ; YES- STORE ATCB
00030'051416            STA     2,TWC,3   ; AS NEW LINK

00031'155100 P2:        MOVL    2,3       ; DO NOT DISTURB CARRY
00032'175240            MOVOR   3,3       ; SET BIT 0
00033'057417            STA     3,@TBP,3; IN TASK QUEUE ENTRY

00034'152460            SUBC    2,2
00035'051416            STA     2,TWC,3   ; CLEAR LAST LINK WORD
00036'002002$           JMP     @TS1      ; ENTER TASK SCHEDULER

                 ; THE FACT THAT BIT 0 IN AC3 IS SET IS OF NO CONSEQUENCE
                 ; TO THE TASK SCHEDULER, BECAUSE IT REPRESENTS THE
                 ; TCB ADDRESS OF A TASK WHICH HAS JUST BEEN SUSPENDED

00037'030003$P3:        LDA     2,ATCB    ; STORE ATCB
00040'051401            STA     2,1,3     ; IN SEMAPHORE LINK WORD
00041'000770            JMP     P2
```

```
0004  SEM

                ;                  ***************
                ;                  *             *
                ;                  *    RAISE    *
                ;                  *             *
                ;                  ***************

                ; CALLING SEQUENCE:
                ;     RAISE
                ;     (ADDRESS OF SEM) OR @(POINTER TO ADDRESS OF SEM)
                ;     (NEXT STATEMENT)

                ; OPERATION:
                ;     THE SEMAPHORE COUNTER IS INCREMENTED. IF THE
                ;     VALUE IS POSITIVE (NOT ZERO), THE NEXT STATEMENT
                ;     IS EXECUTED IMMEDIATELY. OTHERWISE A TASK
                ;     LINKED TO THE 2ND WORD OF THE SEMAPHORE IS
                ;     RE-ACTIVATED. IF THIS TASK IS OF LOWER PRIORITY
                ;     THAN THE TASK MAKING THE CALL THE NEXT STATEMENT
                ;     IS ALSO EXECUTED IMMEDIATELY. THE ACTIVATED TASK
                ;     IS EXECUTED IN DUE COURSE. BUT IF THE TASK IS OF
                ;     OF HIGHER PRIORITY THAN THE TASK MAKING THE CALL
                ;     THIS TASK IS EXECUTED NEXT AND THE CALLING TASK
                ;     HAS TO WAIT.

                ; DESTROYED:     AC3 ONLY

                ; THE "RAISE" OPERATION CORRESPONDS TO THE "V" OPERATION
                ; DESCRIBED BY E. W. DIJKSTRA.

                .ZREL

 00001-000042'V:        RV
        006001-RAISE=   JSR      @V        ; DEFINE CALLING MNEMONIC

                .NREL

 00042'060277 RV:       INTDS              ; DISABLE INTERRUPTS
 00043'054000           STA      3,0       ; SAVE RETURN

 00044'013400           ISZ      @0,3      ; INCREMENT SEMAPHORE COUNTER
 00045'000401           JMP      .+1       ; COULD BE ZERO
 00046'037400           LDA      3,@0,3    ; GET COUNTER VALUE
 00047'174537           IFZM     3,3
 00050'000403           JMP      V1        ; ZERO OR MINUS

 00051'010000           ISZ      0         ; POSITIVE
 00052'002001$          JMP      @RTI      ; RETURN IMMEDIATELY

 00053'034003$V1:       LDA      3,ATCB    ; ACTIVE TCB ADDRESS
 00054'051401           STA      2,TAC2,3; SAVE AC2
 00055'034000           LDA      3,0       ; ADRESS OF CALL
 00056'010000           ISZ      0
 00057'035400           LDA      3,0,3     ; GET SEMAPHORE ADDRESS
 00060'175112           IFM      3,3       ; IS IT ONLY POINTER
 00061'000776           JMP      .-2       ; YES

 00062'175400           INC      3,3       ; POINT TO LINK WORD
 00063'054004$          STA      3,SAV2
```

```
  0005  SEM
01 00064'031400         LDA     2,0,3    ; TCB ADDRESS IN LINK WORD
02 00065'035016         LDA     3,TWC,2  ; NEXT LINK
03 00066'056004$        STA     3,@SAV2  ; STORE IN SEM LINK WORD
04 00067'053017         STA     2,@TBP,2; ACTIVATE TASK
05 00070'034003$        LDA     3,ATCB
06 00071'002002$        JMP     @TS1     ; ENTER TASK SCHEDULER
07
08              .END
```

```
    0006  SEM

ATCB    000003$X   3/23   3/40   3/55   4/50   5/05
LOWER   006000-    3/07
P       000000-    3/06   3/07
P0      000011'    3/18   3/23
P1      000023'    3/35   3/38
P2      000031'    3/43   3/57
P3      000037'    3/33   3/55
RAISE   006001-    4/34
RP      000000'    3/06   3/11
RTI     000001$X   3/21   4/48
RV      000042'    4/33   4/38
SAV2    000004$X   4/59   5/03
TS1     000002$X   3/49   5/06
V       000001-    4/33   4/34
V1      000053'    4/45   4/50
```

0001 DQINI

```

        ; DOUBLE ENDED QUEUE HANDLERS
        ; -------------------------

        ; E. WULFF      2-APR-71
        ; MODIFIED      14-MAR-72

        ; MODIFIED FROM B. WILLIAMS D.Q. HANDLERS

        ; VERSION 4

        ; THESE ROUTINES HANDLE D.Q.'S AND WILL ALLOCATE
        ; CORE BETWEEN A NUMBER OF TASKS OR PROGRAMS.
        ; THEY ARE MOST USEFUL IN IMPLEMENTING DYNAMIC
        ; BUFFERS BETWEEN PROGRAMS AND I/O ROUTINES.

        ; ALGORITHM:
        ;       EACH DOUBLE ENDED QUEUE CONSISTS OF A CONTROL
        ;       BLOCK (DQCB) AND A VARIABLE NUMBER OF CELLS.
        ;       THE NUMBER OF CELLS MAY BE ZERO. EACH CELL
        ;       CONSISTS OF 2 LINK WORDS AND A NUMBER OF
        ;       WORDS OF STORAGE FOR THE USER. THE SIZE
        ;       OF ALL CELLS IN ONE QUEUE SHOULD BE THE SAME.
        ;       THE ADDRESS OF A CELL IS THE ADDRESS OF THE
        ;       FIRST WORD OF FREE STORAGE. THE 2 LINK
        ;       WORDS PRECEDE THIS ADDRESS WITH DISPLACEMENTS
        ;       OF -2 AND -1 WITH RESPECT TO THE ADDRESS
        ;       OF THE CELL. THEY SHOULD NEVER BE INTERFERED
        ;       WITH BY THE USER.

        ;       THE FIRST TWO WORDS OF THE CONTROL BLOCK AND
        ;       THE TWO LINK WORDS OF EACH CELL IN THE QUEUE
        ;       TOGETHER FORM A CIRCULAR LINKED LIST. THE
        ;       FIRST WORD "L" POINTS TO THE CELL ON THE LEFT.
        ;       THE SECOND WORD "R" POINTS TO THE CELL ON THE
        ;       RIGHT. THE LINK WORDS IN THE CONTROL BLOCK
        ;       CLOSE THE CIRCLE. SINCE THE ADDRESS OF THE
        ;       CONTROL BLOCK IS KNOWN, ROUTINES USING THE
        ;       CONTROL BLOCK ADDRESS AS A PARAMETER CAN
        ;       MANIPULATE CELLS IMMEDIATELY TO THE LEFT
        ;       AND RIGHT OF THE CONTROL BLOCK.

        ;       "LGET" AND "RGET" WILL OBTAIN THE ADDRESS
        ;       OF THE APPROPRIATE CELL AND RETURN IT IN AC2.
        ;       THE QUEUE IS RE-LINKED TO EXCLUDE THE CELL
        ;       WHICH HAS BEEN TAKEN OUT. A SEMAPHORE IN THE
        ;       CONTROL BLOCK COUNTS THE NUMBER OF AVAILABLE CELLS
        ;       IN THE QUEUE AND IF AN ATTEMPT IS MADE TO
        ;       GET A CELL WHEN THE QUEUE IS EMPTY, THE TASK
        ;       MAKING THE CALL IS SUSPENDED. A SECOND
        ;       SEMAPHORE COUNTS THE NUMBER OF EMPTY CELLS IN
        ;       THE QUEUE. IF THE VALUE OF THIS SEMAPHORE
        ;       REACHES 0, THE QUEUE IS FULL, AND AN ATTEMPT
        ;       TO PUT MORE CELLS WILL ALSO SUSPEND THE TASK
        ;       MAKING THE CALL. THE MAX. NUMBER OF CELLS
        ;       ALLOWED ON A QUEUE IS SPECIFIED IN THE
        ;       INITIALISATION OF THE D.Q.
```

```
0002 DQINI

          ;       "LGET" AND "RGET" WILL ALSO STORE THE VALUE
          ;       RETURNED IN AC2 LESS 1 IN LOCATION 20. THUS
          ;       LOCATION 20 CAN BE USED AS AN AUTO-INCREMENTING
          ;       POINTER TO THE WORDS IN THE CELL.
          ;       THE WORD LENGTH OF THE CELL IS RETURNED BY BOTH
          ;       ROUTINES IN LOCATION 30. THE OPERATION DSZ 30
          ;       CAN THUS BE USED AS A LOOP COUNT WHEN
          ;       ACCESSING WORDS IN THE CELL.

          ;        "LPUT" AND "RPUT" INSERT CELLS INTO THE QUEUE.
          ;       THE ADDRESS OF THE CELL IS PASSED TO THE
          ;       ROUTINE IN AC2. IF THE SEMAPHORE IN THE CONTROL
          ;       BLOCK HAD PREVIOUSLY SUSPENDED A TASK BECAUSE
          ;       OF A LACK OF CELLS IN THIS QUEUE, THEN "LPUT"
          ;       OR "RPUT" WILL CAUSE RE-ACTIVATION OF ONE TASK
          ;       WAITING FOR A CELL IN THIS QUEUE.

          ;       DOUBLE ENDED QUEUES ARE INITIALISED WITH
          ;       ROUTINE "DQINI". THE CONTROL BLOCK ADDRESS,
          ;       CELL LENGTH,NUMBER OF CELLS AND THE ADDRESS
          ;       OF THE FIRST CELL MUST BE SPECIFIED.

          ; NOTE: THE NUMBER OF CELLS SPECIFIED IS ALSO THE MAX.
          ;       NO. OF CELLS ALLOWED OM THE D.Q. IF A VALID ADDRESS
          ;       IS GIVEN FOR THE FIRST CELL, THE SPACE IN
          ;       CORE WILL BE LINKED INTO A CHAIN OF CELLS.
          ;       IF THE ADDRESS GIVEN IS 0, A ZERO LENGTH D.Q.
          ;       ONLY WILL BE INITIALISED. THE NUMBER OF CELLS
          ;       PARAMETER WILL STILL GIVE THE MAX. NO. OF CELLS
          ;       ALLOWED ON THE D.Q. LATER.
          ; NOTE: CELL LENGTH MUST BE SPECIFIED IN BYTES FOR
          ;       INITIALISATION. IT WILL BE RETURNED IN WORDS
          ;       IN THE LAST WORD OF THE D.Q. CONTROL BLOCK.

          ;       "DQINI" WILL SET UP THE 7 WORD CONTROL BLOCK.
          ; NOTE: 2 OF THESE PRECEDE THE ADDRESS OF THE
          ;       CONTROL BLOCK.
          ;       LET "N" BE THE NUMBER OF CELLS SPECIFIED,
          ;       "S" THE ADDRESS OF THE FIRST CELL SPECIFIED
          ;       AND "L" THE LENGTH OF THE CELL SPECIFIED IN
          ;       A CALL ON "DQINI", THEN THE AREA OF CORE WHICH
          ;       IS SET UP EXTENDS FROM (S) TO (S+N*((L+5)/2)-1).

          ; NOTE: EVEN IF A D.Q. IS OF ZERO LENGTH, IT SHOULD
          ;       BE INITIALISED WITH THE "DQINI" CALL, SO THAT
          ;       THE D.Q. IS RE-INITIALISED WHEN THE SYSTEM
          ;       IS RESTARTED.

          ;       THE FORMAT OF THE ZERO LENGTH D.Q. CONTROL BLOCK
          ;       AFTER INITIALISATION IS:

          ;               (DQCB ADDRESS)   ; LEFT LINK
          ;               (DQCB ADDRESS)   ; RIGHT LINK
          ;(DQCB ADDRESS): 0               ; SEMAPHORE 1 COUNTER
          ;                0               ; SEMAPHORE 1 LINK
          ;               (MAX. NO. CELLS); SEMAPHORE 2 COUNTER
          ;                0               ; SEMAPHORE 2 LINK
          ;               (CELL LENGTH/2)  ; CELL LENGTH
```

```
0004  DQINI

                ; ROUTINE 1

                ; D.Q. INITIALISATION ROUTINE

                ; CALLING SEQUENCE:
                ;       .DQINI
                ;       (DQCB ADDRESS) OR @(POINTER TO ...)
                ;       (CELL BYTE LENGTH) OR @(POINTER TO ...)
                ;       (NUMBER OF CELLS) OR @(POINTER TO ...)
                ;       (ADDRESS OF FIRST CELL) OR @(POINTER TO ...)
                ;       (NEXT STATEMENT)

                ; DESTROYED:    ALL ACC'S,CARRY,L6,L20,L21 AND L30

                .TITL   DQINI

                .ENT    .DQINI,DQINI,L,R,C
                .EXTN   .LPUT

                ; CONTROL BLOCK FORMAT

        177776   L=     -2       ; LEFT LINK
        177777   R=     -1       ; RIGHT LINK
        000000   S1=    0        ; SEMAPHORE 1 COUNT
                ; S1+1= 1        ; SEMAPHORE 1 LINK
        000002   S2=    2        ; SEMAPHORE 2 COUNT
                ; S2+1= 3        ; SEMAPHORE 2 LINK
        000004   C=     4        ; CELL LENGTH

                .ZREL

 00000-000000'DQINI:  DQINT
       006000-.DQINI= JSR     @DQINI

                .NREL

                ; FIRST CREATE A ZERO LENGTH D.Q.

 00000'054021 DQINT:  STA     3,21     ; SAVE RETURN
 00001'035400         LDA     3,0,3    ; DQCB ADDRESS
 00002'175112         IFM     3,3
 00003'000776         JMP     .-2      ; POINTER. TRY AGAIN

 00004'054030         STA     3,30     ; STORE FOR LPUT CALL
 00005'055776         STA     3,L,3    ; INITIALISE LEFT END
 00006'055777         STA     3,R,3    ; INITIALISE RIGHT END
 00007'032021         LDA     2,@21    ; BYTE LENGTH
 00010'102461         SUBC    0,0,SKP
 00011'031000         LDA     2,0,2    ; EMULATE INDIRECT
 00012'151112         IFM     2,2
 00013'000776         JMP     .-2      ; POINTER. TRY AGAIN

 00014'041400         STA     0,S1,3   ; CLEAR AVAILABLE CELLS
 00015'041401         STA     0,S1+1,3; SEMAPHORE
 00016'041403         STA     0,S2+1,3; CLEAR EMPTY CELLS
                                       ; SEMAPHORE. (LINK ONLY)
 00017'141620         INCZR   2,0      ; CONVERT BYTE LENGTH TO WORD LENGTH
 00020'041404         STA     0,C,3    ; STORE IN CONTROL BLOCK
```

```
 0006  DQINI
01
02              ; SET UP THE OTHER PARAMETERS
03
04 00021'032021         LDA     2,@21    ; NUMBER OF CELLS
05  00022'000402        JMP     .+2
06 00023'031000         LDA     2,0,2
07 00024'151112         IFM     2,2
08  00025'000776         JMP     .-2
09
10 00026'051402         STA     2,S2,3   ; NO OF EMPTY CELLS
11 00027'144400         NEG     2,1
12
13 00030'032021         LDA     2,@21    ; ADDRESS OF FIRST CELL
14 00031'000402         JMP     .+2
15 00032'031000         LDA     2,0,2
16 00033'151112         IFM     2,2
17 00034'000776         JMP     .-2
18
19 00035'151015         IFZ     2,2
20 00036'002021         JMP     @21      ; RETURN FOR 0 LENGTH D.Q.
21
22 00037'034411         LDA     3,C2
23  00040'173000         ADD     3,2
24 00041'163000         ADD     3,0      ; ADJUST TO ALLOW FOR LINK WORDS
25
26 00042'177777 DQI1:   .LPUT            ; PUT ON D.Q.
27 00043'100030         @30              ; SPECIFIED IN THE CALL
28 00044'113000         ADD     0,2      ; COMPUTE NEXT CELL ADDRESS
29 00045'125404         INC     1,1,SZR  ; ANY MORE CELLS
30 00046'000774         JMP     DQI1     ; YES
31
32  00047'002021        JMP     @21      ; RETURN
33
34  00050'000002 C2:    2
35
36              .END
  0007  DQINI

C       000004      4/29    4/59
C2      000050'     6/22    6/34
DQI1    000042'     6/26    6/30
DQINI   000000-     4/33    4/34
DQINT   000000'     4/33    4/40
L        177776      4/23    4/46
R       177777      4/24    4/47
S1      000000      4/25    4/54    4/55
S2      000002      4/27    4/56    6/10
.DQIN   006000-     4/34
.LPUT   000042'X    6/26
```

0001 LPUT

```

        ; DOUBLE ENDED QUEUE HANDLERS
        ; -------------------------

        ; E. WULFF        2-APR-71
        ; MODIFIED        14-MAR-72

        ; ROUTINE 2

        ; PUT A CELL ON THE LEFT OF THE D.Q.

        ; INPUT:
        ;       ADDRESS OF THE NEW CELL IS PASSED IN AC2

        ; CALLING SEQUENCE:
        ;       .LPUT
        ;       (DQCB ADDRESS) OR @(POINTER TO DQCB ADDRESS)
        ;       (NEXT STATEMENT)

        ; SEQUENCING:
        ;       IF THE NO. OF CELLS ALLOWED FOR THE D.Q. IS
        ;       EXCEEDED, THE TASK MAKING THE CALL IS SUSPENDED
        ;       UNTIL A CELL IS TAKEN AWAY BY ANOTHER TASK.

        ; DESTROYED:      AC3,L6 AND L20 (ALSO SAV2)
        ; UNCHANGED:      AC0,AC1,AC2,CARRY,L7,L21 AND L30
```

```
 0002  LPUT
01
02               .TITL   LPUT
03
04               .ENT    .LPUT,LPUT
05               .EXTD   L,R,SAV2
06               .EXTN   LOWER,RAISE
07
08               .ZREL
09
10 00000-000000'LPUT:    L.PUT
11       006000-.LPUT=   JSR     @LPUT    ; DEFINE CALLING MNEMONIC
12
13               .NREL
14
15 00000'054020 L.PUT:   STA     3,20     ; SAVE RETURN
16 00001'035400          LDA     3,0,3    ; DQCB ADDRESS
17 00002'175112          IFM     3,3      ; TEST IF POINTER
18 00003'000776          JMP     .-2      ; YES - TRY AGAIN
19 00004'054006          STA     3,6
20 00005'010006          ISZ     6
21 00006'010006          ISZ     6        ; POINT TO 2ND SEMAPHORE
22
23 00007'177777          LOWER            ; NO. OF EMPTY CELLS
24 00010'100006          @6
25
26 00011'014006          DSZ     6
27 00012'014006          DSZ     6        ; POINT TO 1ST SEMAPHORE
28 00013'034006          LDA     3,6      ; RESTORE AC3
29
30 00014'060277          INTDS            ; SECURE DURING RE-LINKING
31 00015'044003$         STA     1,SAV2   ; SAVE AC1 IN TEMP. REG.
32 00016'025401$         LDA     1,L,3    ; GET OLD LEFT END
33 00017'051401$         STA     2,L,3    ; MAKE NEW CELL LEFT END
34 00020'135000          MOV     1,3
35 00021'025402$         LDA     1,R,3    ; MOVE RIGHT LINK
36 00022'045002$         STA     1,R,2    ; TO NEW CELL
37 00023'055001$         STA     3,L,2    ; LINK OLD TO NEW
38 00024'051402$         STA     2,R,3    ; LINK NEW TO OLD
39 00025'024003$         LDA     1,SAV2   ; RESTORE AC1
40 00026'060177          INTEN            ; RELEASE
41
42 00027'177777          RAISE            ; NO. OF AVAILABLE CELLS
43 00030'100006          @6               ; ACTIVATE A TASK
44                                        ; IF WAITING FOR A CELL
45 00031'002020          JMP     @20      ; RETURN SKIPPING PARAMETER
46
47               .END
  0003  LPUT

L        000001$X    2/32    2/33    2/37
LOWER    000007'X    2/23
LPUT     000000-     2/10    2/11
L.PUT    000000'     2/10    2/15
R        000002$X    2/35    2/36    2/38
RAISE    000027'X    2/42
SAV2     000003$X    2/31    2/39
.LPUT    006000-     2/11
```

```
0001 RPUT

                ; DOUBLE ENDED QUEUE HANDLERS
                ; --------------------------

                ; E. WULFF        2-APR-71
                ; MODIFIED        14-MAR-72

                ; ROUTINE 3

                ; PUT A CELL ON THE RIGHT OF THE D.Q.

                ; INPUT:
                ;       ADDRESS OF THE NEW CELL IS PASSED IN AC2

                ; CALLING SEQUENCE:
                ;       .RPUT
                ;       (DQCB ADDRESS) OR @(POINTER TO DQCB ADDRESS)
                ;       (NEXT STATEMENT)

                ; SEQUENCING:
                ;       IF THE NO. OF CELLS ALLOWED FOR THE D.Q. IS
                ;       EXCEEDED, THE TASK MAKING THE CALL IS SUSPENDED
                ;       UNTIL A CELL IS TAKEN AWAY BY ANOTHER TASK.

                ; DESTROYED:    AC3,L6 AND L20 (ALSO SAV2)
                ; UNCHANGED:    AC0,AC1,AC2,CARRY,L7,L21 AND L30
```

```
 0002  RPUT
01
02                  .TITL   RPUT
03
04                  .ENT    .RPUT,RPUT
05                  .EXTD   L,R,SAV2
06                  .EXTN   LOWER,RAISE
07
08                  .ZREL
09
10 00000-000000'RPUT:       R.PUT
11       006000-.RPUT=      JSR     @RPUT     ; DEFINE CALLING MNEMONIC
12
13                  .NREL
14
15  00000'054020 R.PUT:     STA     3,20      ; SAVE RETURN
16 00001'035400             LDA     3,0,3     ; DQCB ADDRESS
17 00002'175112             IFM     3,3       ; TEST IF POINTER
18  00003'000776             JMP     .-2       ; YES - TRY AGAIN
19 00004'054006             STA     3,6
20 00005'010006             ISZ     6
21 00006'010006             ISZ     6         ; POINT TO 2ND SEMAPHORE
22
23 00007'177777             LOWER             ; NO. OF EMPTY CELLS
24 00010'100006             @6
25
26 00011'014006             DSZ     6
27 00012'014006             DSZ     6         ; POINT TO 1ST SEMAPHORE
28 00013'034006             LDA     3,6       ; RESTORE AC3
29
30 00014'060277             INTDS             ; SECURE DURING RE-LINKING
31 00015'044003$            STA     1,SAV2    ; SAVE AC1 IN TEMP. REG.
32 00016'025402$            LDA     1,R,3     ; GET OLD RIGHT END
33 00017'051402$            STA     2,R,3     ; MAKE NEW CELL RIGHT END
34  00020'135000             MOV     1,3
35 00021'025401$            LDA     1,L,3     ; MOVE LEFT LINK
36 00022'045001$            STA     1,L,2     ; TO NEW CELL
37 00023'055002$            STA     3,R,2     ; LINK OLD TO NEW
38 00024'051401$            STA     2,L,3     ; LINK NEW TO OLD
39 00025'024003$            LDA     1,SAV2    ; RESTORE AC1
40 00026'060177             INTEN             ; RELEASE
41
42 00027'177777             RAISE             ; NO. OF AVAILABLE CELLS
43 00030'100006             @6                ; ACTIVATE A TASK
44                                            ; IF WAITING FOR A CELL
45 00031'002020             JMP     @20       ; RETURN SKIPPING PARAMETER
46
47                  .END
  0003  RPUT

L        000001$X    2/35    2/36    2/38
LOWER    000007'X    2/23
R        000002$X    2/32    2/33    2/37
RAISE    000027'X    2/42
RPUT     000000-     2/10    2/11
R.PUT    000000'     2/10    2/15
SAV2     000003$X    2/31    2/39
.RPUT    006000-     2/11
```

```
LGET

                ; DOUBLE ENDED QUEUE HANDLERS
                ; --------------------------

                ; E. WULFF        2-APR-71
                ; MODIFIED        14-MAR-72

                ; ROUTINE 4

                ; GET A CELL FROM THE LEFT OF A D.Q.

                ; CALLING SEQUENCE:
                ;       .LGET
                ;       (DQCB ADDRESS) OR @(POINTER TO DQCB ADDRESS)
                ;       (NEXT STATEMENT)

                ; OUTPUT:
                ;       THE ADDRESS OF THE NEW CELL IS PASSED IN AC2
                ;       THE ADDRESS OF THE NEW CELL - 1 IS PASSED
                ;       IN LOCATION 20. LOCATION 20 CAN BE USED AS AN
                ;       AUTO-INCREMENTING INDEX FOR ACCESSING WORDS IN
                ;       THE CELL.
                ;       THE WORD LENGTH OF THE NEW CELL IS PASSED IN
                ;       LOCATION 30. LOCATION 30 CAN BE USED AS
                ;       A WORD COUNTER WHEN ACCESSING THE CELL.

                ; SEQUENCING:
                ;       IF NO CELL IS AVAILABLE FROM THE D.Q.
                ;       THE TASK MAKING THE CALL IS SUSPENDED
                ;       UNTIL A CELL BECOMES AVAILABLE. THEN
                ;       THE TASK MAKING THE CALL IS RE-ACTIVATED

                ; DESTROYED:    AC2,AC3,L6,L20 AND L30 (ALSO SAV2)
                ; UNCHANGED:    AC0,AC1,CARRY,L7 AND L21
```

```
 0002  LGET
01
02                .TITL   LGET
03
04                .ENT    .LGET,LGET
05                .EXTD   L,R,C,SAV2
06                .EXTN   LOWER,RAISE
07
08                 .ZREL
09
10 00000-000000'LGET:   L.GET
11       006000-.LGET=  JSR     @LGET    ; DEFINE CALLING MNEMONIC
12
13                .NREL
14
15 00000'054020 L.GET:  STA     3,20     ; SAVE RETURN
16 00001'035400         LDA     3,0,3    ; DQCB ADDRESS
17 00002'175112         IFM     3,3      ; TEST IF POINTER
18 00003'000776         JMP     .-2      ; YES - TRY AGAIN
19 00004'054006         STA     3,6      ; POINT TO 1ST SEMAPHORE
20
21 00005'177777         LOWER            ; NO. OF AVAILABLE CELLS
22 00006'100006         @6
23 00007'034006         LDA     3,6      ; DQCB ADDRESS AGAIN
24
25 00010'060277         INTDS            ; SECURE DURING RE-LINKING
26 00011'044004$        STA     1,SAV2   ; SAVE AC1 IN TEMP. REG.
27 00012'025403$        LDA     1,C,3    ; CELL LENGTH FROM DQCB
28 00013'044030         STA     1,30     ; PASS TO CALLER IN L30
29 00014'031401$        LDA     2,L,3    ; NEW CELL FROM LEFT OF DQCB
30 00015'025001$        LDA     1,L,2    ; MOVE ADDRESS OF NEXT
31 00016'045401$        STA     1,L,3    ; LEFT CELL IN DQCB
32 00017'135000         MOV     1,3
33 00020'025002$        LDA     1,R,2    ; MOVE RIGHT LINK
34 00021'045402$        STA     1,R,3    ; TO NEXT CELL
35 00022'024004$        LDA     1,SAV2   ; RESTORE AC1
36 00023'060177         INTEN            ; RELEASE
37
38 00024'010006         ISZ     6
39 00025'010006         ISZ     6        ; POINT TO 2ND SEMAPHORE
40
41 00026'177777         RAISE            ; NO. OF EMPTY CELLS
42 00027'100006         @6
43
44 00030'034020         LDA     3,20     ; RESTORE RETURN
45 00031'050020         STA     2,20     ; SET UP L20 AS AUTO-INCREMENTING
46 00032'014020         DSZ     20       ; POINTER TO NEW CELL
47 00033'001401         JMP     1,3      ; RETURN
48
49                .END
  0003  LGET
C       000003$X    2/27
L       000001$X    2/29    2/30    2/31
LGET    000000-     2/10    2/11
LOWER   000005'X    2/21
L.GET   000000'     2/10    2/15
R       000002$X    2/33    2/34
RAISE   000026'X    2/41
SAV2    000004$X    2/26    2/35
.LGET   006000-     2/11
```

0001 RGET

```
        ; DOUBLE ENDED QUEUE HANDLERS
        ; --------------------------

        ; E. WULFF       2-APR-71
        ; MODIFIED       14-MAR-72

        ; ROUTINE 5

        ; GET A CELL FROM THE RIGHT OF A D.Q.

        ; CALLING SEQUENCE:
        ;       .RGET
        ;       (DQCB ADDRESS) OR @(POINTER TO DQCB ADDRESS)
        ;       (NEXT STATEMENT)

        ; OUTPUT:
        ;       THE ADDRESS OF THE NEW CELL IS PASSED IN AC2
        ;       THE ADDRESS OF THE NEW CELL - 1 IS PASSED
        ;       IN LOCATION 20. LOCATION 20 CAN BE USED AS AN
        ;       AUTO-INCREMENTING INDEX FOR ACCESSING WORDS IN
        ;       THE CELL.
        ;       THE WORD LENGTH OF THE NEW CELL IS PASSED IN
        ;       LOCATION 30. LOCATION 30 CAN BE USED AS
        ;       A WORD COUNTER WHEN ACCESSING THE CELL.

        ; SEQUENCING:
        ;       IF NO CELL IS AVAILABLE FROM THE D.Q.
        ;       THE TASK MAKING THE CALL IS SUSPENDED
        ;       UNTIL A CELL BECOMES AVAILABLE. THEN
        ;       THE TASK MAKING THE CALL IS RE-ACTIVATED

        ; DESTROYED:    AC2,AC3,L6,L20 AND L30 (ALSO SAV2)
        ; UNCHANGED:    AC0,AC1,CARRY,L7 AND L21
```

```
 0002  RGET
01
02              .TITL   RGET
03
04              .ENT    .RGET,RGET
05              .EXTD   L,R,C,SAV2
06              .EXTN   LOWER,RAISE
07
08               .ZREL
09
10 00000-000000'RGET:   R.GET
11       006000-.RGET=  JSR     @RGET    ; DEFINE CALLING MNEMONIC
12
13              .NREL
14
15 00000'054020 R.GET:  STA     3,20     ; SAVE RETURN
16 00001'035400         LDA     3,0,3    ; DQCB ADDRESS
17 00002'175112         IFM     3,3      ; TEST IF POINTER
18 00003'000776         JMP     .-2      ; YES - TRY AGAIN
19 00004'054006         STA     3,6      ; POINT TO 1ST SEMAPHORE
20
21 00005'177777         LOWER            ; NO. OF AVAILABLE CELLS
22 00006'100006         @6
23 00007'034006         LDA     3,6      ; DQCB ADDRESS AGAIN
24
25 00010'060277         INTDS            ; SECURE DURING RE-LINKING
26 00011'044004$        STA     1,SAV2   ; SAVE AC1 IN TEMP. REG.
27 00012'025403$        LDA     1,C,3    ; CELL LENGTH FROM DQCB
28 00013'044030         STA     1,30     ; PASS TO CALLER IN L30
29 00014'031402$        LDA     2,R,3    ; NEW CELL FROM RIGHT OF DQCB
30 00015'025002$        LDA     1,R,2    ; MOVE ADDRESS OF NEXT
31 00016'045402$        STA     1,R,3    ; RIGHT CELL IN DQCB
32 00017'135000         MOV     1,3
33 00020'025001$        LDA     1,L,2    ; MOVE LEFT LINK
34 00021'045401$        STA     1,L,3    ; TO NEXT CELL
35 00022'024004$        LDA     1,SAV2   ; RESTORE AC1
36 00023'060177         INTEN            ; RELEASE
37
38 00024'010006         ISZ     6
39 00025'010006         ISZ     6        ; POINT TO 2ND SEMAPHORE
40
41 00026'177777         RAISE            ; NO. OF EMPTY CELLS
42 00027'100006         @6
43
44 00030'034020         LDA     3,20     ; RESTORE RETURN
45 00031'050020         STA     2,20     ; SET UP L20 AS AUTO-INCREMENTING
46 00032'014020         DSZ     20       ; POINTER TO NEW CELL
47 00033'001401         JMP     1,3      ; RETURN
48
49              .END
  0003  RGET

C        000003$X    2/27
L        000001$X    2/33    2/34
LOWER    000005'X    2/21
R        000002$X    2/29    2/30    2/31
RAISE    000026'X    2/41
RGET     000000-     2/10    2/11
R.GET    000000'     2/10    2/15
SAV2     000004$X    2/26    2/35
.RGET    006000-     2/11
```

```
 0001  CELLO
01
02                 ; RE-ENTRANT CELL OUTPUT ROUTINE
03                 ; ------------------------------
04
05                 ; E. WULFF      23-JUL-71
06                 ; MODIFIED      4-JUL-72
07
08                 ; INITIALISE THE EVENT CONTROL WORDS XXXE1 & XXXE2
09                 ; TO 000000 & 100000 RESPECTIVELY. THE ADDRESS OF
10                 ; XXXE1 IS PASSED IN AC2. XXXE2 IS ASSUMED TO BE IN
11                 ; THE WORD AFTER XXXE1.
12                 ; INITIALISE A D.Q. WHOSE CONTROL BLOCK ADDRESS
13                 ; IS IN LOCATION 40. THEN OUTPUT CELLS FROM
14                 ; THAT D.Q. VIA A ROUTINE WHOSE ADDRESS IS IN
15                 ; LOCATION 41. DEFINE A TCB AND A DQCB FOR EACH
16                 ; DEVICE SHARING THIS ROUTINE.
17                 ; THE TASK WILL SUSPEND ITSELF WHENEVER IT'S D.Q.
18                 ; IS EXHAUSTED. IT WILL BE RE-ACTIVATED WHENEVER
19                 ; USERS PUT CELLS ON THE D.Q.
20                 ; IF MORE THEN THE MAX. NO. OF CELLS SPECIFIED
21                 ; IN LOC 7 OF THE TCB ARE PUT ON THE D.Q., THE
22                 ; USER TASK IS SUSPENDED UNTIL THIS TASK GETS A
23                 ; CELL FROM THE D.Q.
24
25                 .TITL   CELLO
26
27                 .ENT    CELLO
28                 .EXTN   .DQINI,.RGET,.LPUT,FREE,CL
29
30                 .NREL
31
32 00000'041001 CELLO:  STA     0,1,2    ; XXXE2 (- 0
33 00001'102620         SUBZR   0,0      ; 100000
34 00002'041000         STA     0,0,2    ; XXXE1 COMPL. BIT SET
35 00003'177777         .DQINI           ; INITIALISE THE D.Q.
36 00004'100040         @40              ; DQCB ADDRESS IN L40
37 00005'177777         CL               ; CELL LENGTH IN BYTES
38 00006'100007         @7               ; MAX. NO. OF CELLS IN L7
39 00007'000000         0                ; ZERO NO. OF CELLS INITIALLY
40
41 00010'177777 LOOP:   .RGET            ; GET NEXT CELL FOR OUTPUT
42 00011'100040         @40              ; WAIT IF NONE THERE
43
44 00012'006041         JSR     @41      ; OUTPUT THE CELL
45 00013'000005'        CL               ; MAX. NO. OF BYTES
46
47 00014'177777         .LPUT            ; RETURN THE CELL TO THE
48 00015'177777         FREE             ; FREE DOUBLE ENDED QUEUE
49
50 00016'000772         JMP     LOOP
51
52              .END
0002  CELLO

CELLO   000000'    1/32
CL      000013'X   1/37    1/45
FREE    000015'X   1/48
LOOP    000010'    1/41    1/50
.DQIN   000003'X   1/35
.LPUT   000014'X   1/47
.RGET   000010'X   1/41
```

```
0001   TTODQ

                 ; PRINT TTODQ ON THE TELETYPE
                 ; --------------------------

                 ; E. WULFF        23-JUL-71
                 ; MODIFIED        4-JUL-72

                 ; DEFINES TCB AND DQCB FOR CELLO

                 .TITL    TTODQ

                 .ENT     TTODQ,TCBO
                 .EXTN    CELLO,TTOE1
                 .IFN     T
                 .EXTN    DB1.5    ; TO FORCE DEBUG 1.5 TO BE LOADED
                 .ENT     SEMDT
                 .ENDC
                 .EXTD    PUTB

                 .NREL

                 ; TASK CONTROL BLOCK

       000020 TCBO:    .BLK     20

                 ; INITIAL VALUES FOR TCB

   00020'046414 1B1+1B4+1B5+1B7+1B12+1B13;INITIALISATION CONTROL WORD
   00021'177777        TTOE1           ; AC2
                    .IFE T
   00022'177777        CELLO           ; PC
                    .ENDC
                    .IFN T
   00023'000037'       START           ; DEBUG PC
                    .ENDC
   00024'000003        3               ; PMASK TTI, TTO, INI, INO
   00025'000001        1               ; L7 - 1 CELL ALLOWED
   00026'000032'       TTODQ           ; L40 - DQCB ADDRESS
   00027'100001$       @PUTB           ; L41 - OUTPUT ROUTINE

                 ; D.Q.. CONTROL BLOCK

       000002          .BLK     2      ; LINKS
       000005 TTODQ:   .BLK     5      ; SEMAPHORES & CONSTANT

                 ; INITIALISATION OF DEBUG SEMAPHORE
                    .IFN T
   00037'040405 START: STA      0,SEMDT
   00040'010404        ISZ      SEMDT   ; +1
   00041'040404        STA      0,SEMDT+1
   00042'002401        JMP      @.+1
   00043'000022'       CELLO

       000002 SEMDT:   .BLK     2      ; SEMAPHORE

                 .END
```

0002 TTODQ

CELLO	000043'X	1/31	1/52		
DB1.5	177777 X				
PUTB	000001$X	1/39			
SEMDT	000044'	1/48	1/49	1/50	1/54
START	000037'	1/34	1/48		
T	000000U	1/14	1/30	1/33	1/47
TCBO	000000'	1/24			
TTODQ	000032'	1/38	1/44		
TTOE1	000021'X	1/29			

```
 0002  PUTB

                ; BYTE BUFFERED PRINT CHARACTER ROUTINE
                ; -----------------------------------

                ; E. WULFF                7-JULY-71
                ; MODIFIED                4-JUL-72

                ; INPUT:
                ;       AC2 MUST CONTAIN THE WORD ADDRESS
                ;       OF THE FIRST BYTE IN THE BUFFER

                ; CALLING SEQUENCE:
                ;       JSR @PUTB    OR   JSR @PUTBI
                ;       MAX NO. OF BYTES IN THE BUFFER
                ;       NEXT STATEMENT

                ; DESTROYED:  AC0,AC3 AND L6

                .TITL   PUTB

                .ENT    PUTB,PUTBI
                .EXTD   TTOBC,SAV2,SAVC
                .EXTN   .WAIT,TTOE1,TTOE2,TTOFB,TTOPB
         000001 .IFN    T                 ; DEBUG TASK
                .EXTN   LOWER,RAISE,SEMDT
                .ENDC

                .ZREL

 00000-000000'PUTB:     RPUTB
 00001-000001'PUTBI:    RPUTI

                .NREL

 00000'102001 RPUTB:    ADC     0,0,SKP ; AC0 USED AS IMMEDIATE FLAG
 00001'102460 RPUTI:    SUBC    0,0
 00002'054006           STA     3,6     ; SAVE RETURN ADDRESS AT LOC 6
       000001     .IFN  T
 00003'177777           LOWER
 00004'177777           SEMDT           ; SEMAPHORE FOR DEB TASK
                  .ENDC
 00005'177777           .WAIT           ; WAIT FOR END OF LAST CHARACTER
 00006'177777           TTOE1           ; IN THE PREVIOUS BUFFER
 00007'176460           SUBC    3,3
 00010'056776           STA     3,@.-2

 00011'036006           LDA     3,@6    ; GET BUFFER COUNT
 00012'010006           ISZ     6
 00013'054001$          STA     3,TTOBC ; STORE IN OUTPUT ROUTINE

 00014'060277           INTDS
 00015'175200           MOVR    3,3     ; SAVE CARRY
 00016'054003$          STA     3,SAVC
 00017'050002$          STA     2,SAV2  ; SAVE BUFFER POINTER IN AC2
                                        ; STORE BYTE POINTER IN TTOBP
 00020'006414           JSR     @ATTOF  ; FETCH FIRST BYTE INTO TTOCH

 00021'060277           INTDS           ; DISPLAY FIRST BYTE
 00022'006413           JSR     @ATTOP  ; FETCH 2ND BYTE IF THERE
```

```
 0003  PUTB
01
02 00023'101015          IFZ     0,0       ; TEST RETURN MODE
03 00024'002006          JMP     @6        ; RETURN IMMEDIATELY
04
05 00025'000005'         .WAIT             ; WAIT FOR BEGINNING OF LAST
06 00026'177777          TTOE2             ; CHARACTER TRANSMITTED
07 00027'176460          SUBC    3,3
08 00030'056776          STA     3,@.-2
09       000001     .IFN T
10 00031'177777          RAISE
11 00032'000004'         SEMDT
12                  .ENDC
13 00033'002006          JMP     @6        ; RETURN
14
15 00034'177777 ATTOF:   TTOFB
16 00035'177777 ATTOP:   TTOPB
17
18             .END
```

```
 0004  PUTB

ATTOF   000034'     2/56    3/15
ATTOP   000035'     2/59    3/16
LOWER   000003'X    2/39
PUTB    000000-     2/30
PUTBI   000001-     2/31
RAISE   000031'X    3/10
RPUTB   000000'     2/30    2/35
RPUTI   000001'     2/31    2/36
SAV2    000002$X    2/54
SAVC    000003$X    2/53
SEMDT   000032'X    2/40    3/11
TTOBC   000001$X    2/49
TTOE1   000006'X    2/43
TTOE2   000026'X    3/06
TTOFB   000034'X    3/15
TTOPB   000035'X    3/16
.WAIT   000025'X    2/42    3/05
```

0001 TTODR

```
; TELETYPE BYTE ORIENTED PRINT INTERRUPT SERVICE
; ------------------------------------------

; XXX

; E. WULFF      12-JULY-71
; VERS. II

; THIS INTERRUPT SERVICE ROUTINE SERVICES THE FOLLOWING
; CONTROL CODES:

; 00    MARK LAST BYTE OF A STRING
; 01    INSERT CRLF
; 02    INSERT CRLF
; 04    'EOT' SUBSTITUTE FOR TAB
; 05    'ENQ' SUBSTITUTE FOR FORM FEED
; 10    CR ONLY
; 11    TAB TO THE NEXT COLLUMN OF 8
; 12    LINE-FEED (THE FIRST LF AFTER CR IS IGNORED)
; 14    FORM-FEED (COMPLETE THE CURRENT PAGE)
; 15    CARRIAGE-RETURN (INSERT LF)
; 17    SUBSTITURE ¢
; 31    SUBSTITUTE SPACE
; 32    SUBSTITUTE _
; 34    SUBSTITUTE ¯
; 35    SUBSTITUTE LF
; 37    SUBSTITUTE ±
; 177   RUB-OUT IS IGNORED

; ANY OTHER CONTROL CODES ARE NOT TRANSMITTED.

; THE CONSTANTS ARE CORRECT FOR AN OLIVETTI TERMINAL
; TYPE 308, ADJUSTED FOR 80 CHARACTER LINES.
; THE PAGE LENGTH IS 60 LINES WITH 6 EXTRA LINES
; TO COMPLETE AN 11" PAGE.

.TITL   TTODR

.ENT    TTOS,TTOBP,TTOBC,TTOCH,TTOFB,TTOPB,TTOE1,TTOE2
.EXTD   C177,C12,C15,RTI,POSTI,SAV2,SAVC
```

```
0002  TTODR

                ; BUFFER CONSTANTS

                .ZREL

00000-000000 TTOBP:  0
00001-000000 TTOBC:  0
00002-000000 TTOCH:  0

                .NREL

                ; TABLE OF CONTROL CHARACTER ROUTINES

00000'000151'ILS:     LBYTE    ; NULL MARKS LAST BYTE
00001'000171'         ICR      ; CURSOR SAVE
00002'000171'         ICR      ; CURSOR RESTORE
00003'000147'         NOCH
00004'000213'         TAB      ; EOT
00005'000205'         FF       ; ENQ
00006'000147'         NOCH
00007'000147'         NOCH
00010'100015          @15      ; HOME
00011'000213'         TAB      ; TAB
00012'000202'         LF       ; LINE FEED
00013'000147'         NOCH
00014'000205'         FF       ; FORM-FEED
00015'000172'         CR       ; CARRIAGE RETURN
00016'000147'         NOCH
00017'100135          @"¢      ; BLINK OFF
00020'000147'         NOCH
00021'000147'         NOCH
00022'000147'         NOCH
00023'000147'         NOCH
00024'000147'         NOCH
00025'000147'         NOCH
00026'000147'         NOCH
00027'000147'         NOCH
00030'000147'         NOCH
00031'100040          @40      ; CURSOR RIGHT
00032'100137          @"_      ;   "    LEFT
00033'000147'         NOCH
00034'100136          @"       ;   "    UP
00035'100012          @12      ;   "    DOWN
00036'000147'         NOCH
00037'100133          @"±      ; BLINK ON

                ; ENTER HERE FROM BUFFERED OUTPUT TO FETCH FIRST BYTE

00040'054000 TTOFB:   STA      3,0
00041'155020          MOVZ     2,3      ; CHANGE TO MOVR FOR BYTE ADDRESS
00042'000420          JMP      NB1

                ; ENTER HERE FROM BUFFERED OUTPUT TO TRANSMIT FIRST BYTE

00043'054000 TTOPB:   STA 3    0

                ; ENTER HERE FROM INTERRUPT HANDLER

00044'010001-TTOS:    ISZ      TTOBC
```

```
 0003  TTODR
 00045'014001-          DSZ      TTOBC
 00046'000404           JMP      TTON
 00047'060211           NIOC     TTO
 00050'006005$          JSR      @POSTI
 00051'000252'          TTOE1
                 ; OUTPUT THE PREVIOUS CHARACTER
 00052'175200 TTON:     MOVR     3,3
 00053'054007$          STA      3,SAVC
 00054'050006$          STA      2,SAV2
 00055'030002-          LDA      2,TTOCH
 00056'071111           DOAS     2,TTO
 00057'002554           JMP      @LINK    ; SET UP BY SLINK
                 ; FETCH NEXT BYTE AND ANALYSE
 00060'034000-NBYTE:    LDA      3,TTOBP
 00061'175600           INCR     3,3
 00062'031400 NB1:      LDA      2,0,3
 00063'175012           MOV#     3,3,SZC
 00064'151300           MOVS     2,2
 00065'175100           MOVL     3,3
 00066'054000-          STA      3,TTOBP
 00067'034001$          LDA      3,C177
 00070'173405           AND      3,2,SNR
 00071'050001-          STA      2,TTOBC  ; CLEAR 'BC' FOR NULL BYTE
 00072'156415           IFEQ     2,3
 00073'000454           JMP      NOCH
 00074'034541           LDA      3,C40
 00075'156032           IFLT     2,3      ; TEST FOR CONTROL CHARACTER
 00076'004465           JSR      CCH      ; YES - CONTROL CHARACTER
 00077'050002-CH:       STA      2,TTOCH
 00100'014537           DSZ      CHC      ; CHARACTER COUNT
 00101'000456           JMP      RESTO
 00102'034536           LDA      3 CLL    ; LINE LENGTH
 00103'054534           STA      3,CHC
 00104'004452           JSR      SLINK    ; OUTPUT CHARACTER
 00105'030534           LDA      2,CHI1   ; 1ST DUMMY
 00106'050002-          STA      2,TTOCH
 00107'004447           JSR      SLINK
 00110'030532           LDA      2,CHI2   ; 2ND DUMMY
 00111'050002-LF1:      STA      2,TTOCH
 00112'014531           DSZ      LIC      ; LINE COUNT
 00113'000433           JMP      NLINK    ; BACK TO NORMAL
 00114'004442           JSR      SLINK    ; OUTPUT LAST CHARACTER
 00115'034527 FF2:      LDA      3,PAGEL  ; PAGE LENGTH
 00116'054525           STA      3,LIC
 00117'034526           LDA      3,LOWC
 00120'054531           STA      3,TABC
```

```
 0004  TTODR
00121'030003$          LDA     2,C15    ; EXTRA CR

00122'050002-LOW:      STA     2,TTOCH  ; OUTPUT
00123'004433           JSR     SLINK
00124'030002$          LDA     2,C12    ; EXTRA LF'S
00125'014524           DSZ     TABC
00126'000774           JMP     LOW

00127'034517           LDA     3,DASHC  ; NO OF DASHES
00130'054521           STA     3,TABC
00131'030517           LDA     2,DASH
00132'050002-          STA     2,TTOCH

00133'004423 ACROSS:   JSR     SLINK
00134'014515           DSZ     TABC
00135'000776           JMP     ACROSS

00136'034511           LDA     3,TOPC   ; NO OF LF'S AT TOP
00137'054512           STA     3,TABC
00140'030003$          LDA     2,C15

00141'050002-TOP:      STA     2,TTOCH
00142'004414           JSR     SLINK
00143'030002$          LDA     2,C12
00144'014505           DSZ     TABC
00145'000774           JMP     TOP

                ; NORMAL LINK AGAIN

00146'004410 NLINK:    JSR     SLINK

                ; NO CHARACTER ENTRY

00147'014001-NOCH:     DSZ     TTOBC
00150'000710           JMP     NBYTE

00151'030006$LBYTE:    LDA     2,SAV2
00152'034007$          LDA     3,SAVC
00153'175100           MOVL    3,3
00154'006005$          JSR     @POSTI
00155'000253'          TTOE2

                ; SAVE LINK SUBROUTINE ENTRY

00156'054455 SLINK:    STA     3,LINK

                ; RESTORE STATUS AND RETURN

00157'030006$RESTO:    LDA     2,SAV2
00160'034007$          LDA     3,SAVC
00161'175100           MOVL    3,3
00162'002004$          JMP     @RTI

                ; CONTROL CHARACTER

00163'157000 CCH:      ADD     2,3      ; COMPUTE TABLE ENTRY
00164'035701           LDA     3,ILS-CH,3
00165'175113           IFZP    3,3      ; TEST BIT 0
00166'001400           JMP     0,3      ; ENTER SPECIAL ROUTINE
```

```
0005  TTODR

00167'171000          MOV    3,2       ; SUBSTITUTE OTHER CHARACTER
00170'000707          JMP    CH

               ; CONTROL CHARACTER ROUTINES

00171'030003$ICR:     LDA    2,C15     ; INSERT CR-LF

00172'050002-CR:      STA    2,TTOCH
00173'034445          LDA    3,CLL
00174'054443          STA    3,CHC
00175'004761          JSR    SLINK

00176'030002$         LDA    2,C12
00177'176000          ADC    3,3
00200'054436          STA    3,LFLAG
00201'000710          JMP    LF1

00202'010434 LF:      ISZ    LFLAG
00203'000706          JMP    LF1
00204'000743          JMP    NOCH      ; IGNORE FIRST LF AFTER CR

00205'030002$FF:      LDA    2,C12
00206'050002-         STA    2,TTOCH

00207'004747 FF1:     JSR    SLINK
00210'014433          DSZ    LIC
00211'000776          JMP    FF1
00212'000703          JMP    FF2

               ; THIS TAB ROUTINE WILL ONLY WORK WITH A LINE
               ; LENGTH (CLL) WHICH IS A MULTIPLE OF EIGHT

00213'034422 TAB:     LDA    3,C40     ; SPACE
00214'054002-         STA    3,TTOCH
00215'014422          DSZ    CHC
00216'000401          JMP    .+1
00217'034420          LDA    3,CHC
00220'030414          LDA    2,CC7
00221'173405          AND    3,2,SNR
00222'000747          JMP    ICR

00223'050414          STA    2,CHC
00224'156405          SUB    2,3,SNR
00225'000721          JMP    NLINK

00226'054423          STA    3,TABC

00227'004727 TA1:     JSR    SLINK
00230'014421          DSZ    TABC
00231'000776          JMP    TA1
00232'000714          JMP    NLINK
               ; CONSTANTS AND VARIABLES

00233'000147'LINK:    NOCH
00234'177770 CC7:     177770
00235'000040 C40:     40
00236'000000 LFLAG:   0
00237'000001 CHC:     1
```

```
 0006   TTODR
00240'000120 CLL:     80.      ; LINE LENGTH (72. FOR ASR 33)
00241'000015 CHI1:    15       ; (15 FOR ASR 33)
00242'000177 CHI2:    177      ; (12 FOR ASR 33)
00243'000001 LIC:     1
00244'000074 PAGEL:   60.      ; 11" PAGE (60 LINES)
00245'000005 LOWC:    5.       ; CR + 4 LF'S  LOW MARGIN
00246'000003 DASHC:   3.       ; NO OF DASHES
00247'000002 TOPC:    2.       ; CR + 2 LF'S   TOP MARGIN
00250'000055 DASH:    "-
00251'000000 TABC:    0

00252'100000 TTOE1:   100000
00253'000000 TTOE2:   0

             .END
```

0007 TTODR

ACROS	000133'	4/14	4/16						
C12	000002$X	4/05	4/24	5/14	5/23				
C15	000003$X	4/01	4/20	5/07					
C177	000001$X	3/27							
C40	000235'	3/34	5/34	5/57					
CC7	000234'	5/39	5/56						
CCH	000163'	3/36	4/56						
CH	000077'	3/38	4/57	5/03					
CHC	000237'	3/39	3/43	5/11	5/36	5/38	5/43	5/59	
CHI1	000241'	3/46	6/02						
CHI2	000242'	3/49	6/03						
CLL	000240'	3/42	5/10	6/01					
CR	000172'	2/27	5/09						
DASH	000250'	4/11	6/09						
DASHC	000246'	4/09	6/07						
FF	000205'	2/19	2/26	5/23					
FF1	000207'	5/26	5/28						
FF2	000115'	3/56	5/29						
ICR	000171'	2/15	2/16	5/07	5/41				
ILS	000000'	2/14	4/57						
LBYTE	000151'	2/14	4/37						
LF	000202'	2/24	5/19						
LF1	000111'	3/51	5/17	5/20					
LFLAG	000236'	5/16	5/19	5/58					
LIC	000243'	3/52	3/57	5/27	6/04				
LINK	000233'	3/15	4/45	5/55					
LOW	000122'	4/03	4/07						
LOWC	000245'	3/58	6/06						
NB1	000062'	2/51	3/21						
NBYTE	000060'	3/19	4/35						
NLINK	000146'	3/53	4/30	5/45	5/52				
NOCH	000147'	2/17	2/20	2/21	2/25	2/28	2/30	2/31	2/32
		2/33	2/34	2/35	2/36	2/37	2/38	2/41	2/44
		3/32	4/34	5/21	5/55				
PAGEL	000244'	3/56	6/05						
POSTI	000005$X	3/05	4/40						
RESTO	000157'	3/40	4/49						
RTI	000004$X	4/52							
SAV2	000006$X	3/12	4/37	4/49					
SAVC	000007$X	3/11	4/38	4/50					
SLINK	000156'	3/45	3/48	3/54	4/04	4/14	4/23	4/30	4/45
		5/12	5/26	5/49					
TA1	000227'	5/49	5/51						
TAB	000213'	2/18	2/23	5/34					
TABC	000251'	3/59	4/06	4/10	4/15	4/19	4/25	5/47	5/50
		6/10							
TOP	000141'	4/22	4/26						
TOPC	000247'	4/18	6/08						
TTOBC	000001-	2/07	2/59	3/01	3/29	4/34			
TTOBP	000000-	2/06	3/19	3/25					
TTOCH	000002-	2/08	3/13	3/38	3/47	3/51	4/03	4/12	4/22
		5/09	5/24	5/35					
TTOE1	000252'	3/06	6/12						
TTOE2	000253'	4/41	6/13						
TTOFB	000040'	2/49							
TTON	000052'	3/02	3/10						
TTOPB	000043'	2/55							
TTOS	000044'	2/59							

```
  0001  INODQ
01
02                ; PRINT INODQ ON THE INFOTON
03                ; --------------------------
04
05                ; E. WULFF      23-JUL-71
06                ; MODIFIED      4-JUL-72
07
08                ; DEFINES TCB AND DQCB FOR CELLO
09
10                .TITL   INODQ
11
12                .ENT    INODQ,TCBL
13                .EXTN   CELLO,INOE1
14                .EXTD   INDB
15
16                .NREL
17
18                ; TASK CONTROL BLOCK
19
20        000020 TCBL:   .BLK    20
21
22                ; INITIAL VALUES FOR TCB
23
24 00020'046414 1B1+1B4+1B5+1B7+1B12+1B13;INITIALISATION CONTROL WORD
25 00021'177777         INOE1           ; AC2
26 00022'177777         CELLO           ; PC
27 00023'000003         3               ; PMASK TTI, TTO, INI, INO
28 00024'000003         3               ; L7 - 3 CELLS ALLOWED
29 00025'000031'        INODQ           ; L40 - DQCB ADDRESS
30 00026'100001$        @INDB           ; L41 - OUTPUT ROUTINE
31
32                ; D.Q.. CONTROL BLOCK
33
34        000002         .BLK    2      ; LINKS
35        000005 INODQ:  .BLK    5      ; SEMAPHORES & CONSTANT
36
37                .END
```

```
  0002  INODQ

CELLO   000022'X    1/26
INDB    000001$X    1/30
INODQ   000031'     1/29     1/35
INOE1   000021'X    1/25
TCBL    000000'     1/20
```

```
0002  INDB

               ; BYTE BUFFERED DISPLAY CHARACTER ROUTINE (DEV.51)
               ; ------------------------------------

               ; XXX

               ; E. WULFF.          18-MAY-71

               ; INPUT:
               ;       AC2 MUST CONTAIN THE WORD ADDRESS
               ;       OF THE FIRST BYTE IN THE BUFFER

               ; CALLING SEQUENCE:
               ;       JSR @INDB   OR   JSR @INDBI
               ;       MAX NO. OF BYTES IN THE BUFFER
               ;       NEXT STATEMENT

               ; DESTROYED:  AC0,AC3 AND L6

               .TITL   INDB

               .ENT    INDB,INDBI
               .EXTD   WAIT,INOBC,SAV2,SAVC
               .EXTN   INOE1,INOE2,INOFB,INOPB

               .ZREL

00000-000000'INDB:     RINDB
00001-000001'INDBI:    RINDI

               .NREL

00000'102001 RINDB:    ADC     0,0,SKP ; AC0 USED AS IMMEDIATE FLAG
00001'102460 RINDI:    SUBC    0,0
00002'054006           STA     3,6     ; SAVE RETURN ADDRESS AT LOC 6
00003'006001$          JSR     @WAIT   ; WAIT FOR END OF LAST CHARACTER
00004'177777           INOE1           ; IN THE PREVIOUS BUFFER
00005'176460           SUBC    3,3
00006'056776           STA     3,@.-2

00007'036006           LDA     3,@6    ; GET BUFFER COUNT
00010'010006           ISZ     6
00011'054002$          STA     3,INOBC ; STORE IN OUTPUT ROUTINE

00012'060277           INTDS
00013'175200           MOVR    3,3     ; SAVE CARRY
00014'054004$          STA     3,SAVC
00015'050003$          STA     2,SAV2  ; SAVE BUFFER POINTER IN AC2
                                       ; STORE BYTE POINTER IN INOBP
00016'006412           JSR     @AINOF  ; FETCH FIRST BYTE INTO INOCH

00017'060277           INTDS           ; DISPLAY FIRST BYTE
00020'006411           JSR     @AINOP  ; FETCH 2ND BYTE IF THERE

00021'101015           IFZ     0,0     ; TEST RETURN MODE
00022'002006           JMP     @6      ; RETURN IMMEDIATELY

00023'006001$          JSR     @WAIT   ; WAIT FOR BEGINNING OF LAST
00024'177777           INOE2           ; CHARACTER TRANSMITTED
```

```
  0003  INDB
01 00025'176460         SUBC    3,3
02 00026'056776         STA     3,@.-2
03
04 00027'002006         JMP     @6       ; RETURN
05
06 00030'177777 AINOF:  INOFB
07 00031'177777 AINOP:  INOPB
08
09               .END
```

```
   0004  INDB

 AINOF   000030'    2/50    3/06
 AINOP   000031'    2/53    3/07
 INDB    000000-    2/28
 INDBI   000001-    2/29
 INOBC   000002$X   2/43
 INOE1   000004'X   2/37
 INOE2   000024'X   2/59
 INOFB   000030'X   3/06
 INOPB   000031'X   3/07
 RINDB   000000'    2/28    2/33
 RINDI   000001'    2/29    2/34
 SAV2    000003$X   2/48
 SAVC    000004$X   2/47
 WAIT    000001$X   2/36    2/58
```

0001 INODR

```
        ; INFOTON BYTE ORIENTED DISPLAY INTERRUPT SERVICE
        ; -----------------------------------------------

        ; XXX

        ; E. WULFF      4-JULY-71
        ; VERS. VII

        ; THIS INTERRUPT SERVICE ROUTINE SERVICES THE FOLLOWING
        ; CONTROL CODES:

        ; 00     MARK LAST BYTE OF A STRING
        ; 01     SAVE THE POSITION OF THE CURSOR
        ; 02     RESTORE THE CURSOR TO THE POSITION LAST SAVED
        ; 10     HOME THE CURSOR WITHOUT ERASING
        ; 11     TAB TO THE NEXT COLUMN OF 8
        ; 12     LINE-FEED (THE FIRST LF AFTER CR IS IGNORED)
        ; 14     ERASE SCREEN AND HOME CURSOR
        ; 15     CARRIAGE-RETURN
        ; 17     BLINK-OFF
        ; 31     CURSOR RIGHT
        ; 32     CURSOR LEFT
        ; 34     CURSOR UP
        ; 35     CURSOR DOWN
        ; 37     BLINK-ON
        ; 177    RUB-OUT (ERASE CHAR. ON THE LEFT)

        ; ANY OTHER CONTROL CODES ARE NOT TRANSMITTED.

        ; THE COSTANTS ARE CORRECT FOR AN INFOTON DISPLAY
        ; WITH 20 LINES, 64 CHARACTERS PER LINE AND SET TO
        ; 'ROLL' MODE. A CURSOR COUNT IS MAINTAINED WHICH
        ; FOLLOWS THE ACTUAL CURSOR ON THE SCREEN. THE CURSOR
        ; SAVE AND RESTORE FEATURE MAKE USE OF THIS COUNT.

        .TITL   INODR

        .ENT    INOS,INOBP,INOBC,INOCH,INOFB,INOPB,INOE1,INOE2
        .EXTD   C177,C12,RTI,POSTI,SAV2,SAVC
```

```
0002  INODR

                 ; BUFFER CONSTANTS

                 .ZREL

00000-000000 INOBP:   0
00001-000000 INOBC:   0
00002-000000 INOCH:   0

                 .NREL

                 ; TABLE OF CONTROL CHARACTER ROUTINES

00000'000137'ILS:     LBYTE    ; NULL MARKS LAST BYTE
00001'000146'         CSAVE    ; CURSOR SAVE
00002'000151'         CRSTR    ; CURSOR RESTORE
00003'000135'         NOCH
00004'000135'         NOCH
00005'000135'         NOCH
00006'000135'         NOCH
00007'000135'         NOCH
00010'000216'         HOME     ; HOME
00011'000200'         TAB      ; TAB
00012'000213'         LF       ; LINE FEED
00013'000135'         NOCH
00014'000216'         HOME     ; ERASE
00015'000116'         CR       ; CARRIAGE RETURN
00016'000135'         NOCH
00017'000071'         CH       ; BLINK OFF
00020'000135'         NOCH
00021'000135'         NOCH
00022'000135'         NOCH
00023'000135'         NOCH
00024'000135'         NOCH
00025'000135'         NOCH
00026'000135'         NOCH
00027'000135'         NOCH
00030'000135'         NOCH
00031'000235'         CSRRT    ; CURSOR RIGHT
00032'000232'         CSRLT    ;   "    LEFT
00033'000135'         NOCH
00034'000233'         CSRUP    ;   "    UP
00035'000236'         CSRDN    ;   "    DOWN
00036'000135'         NOCH
00037'000071'         CH       ; BLINK ON

                 ; ENTER HERE FROM BUFFERED OUTPUT TO FETCH FIRST BYTE

00040'054000 INOFB:   STA      3,0
00041'155020          MOVZ     2,3      ; CHANGE TO MOVR FOR BYTE ADDRESS
00042'000412          JMP      NB1

                 ; ENTER HERE FROM BUFFERED OUTPUT TO TRANSMIT FIRST BYTE

00043'054000 INOPB:   STA 3    0

                 ; ENTER HERE FROM INTERRUPT HANDLER

00044'010001-INOS:    ISZ      INOBC
```

```
 0003  INODR
00045'014001-          DSZ     INOBC
00046'000442           JMP     INON

00047'060251           NIOC    INO
00050'006004$          JSR     @POSTI
00051'000270'          INOE1

00052'034000-NBYTE:    LDA     3,INOBP
00053'175600           INCR    3,3
00054'031400 NB1:      LDA     2,0,3
00055'175012           MOV#    3,3,SZC
00056'151300           MOVS    2,2
00057'175100           MOVL    3,3
00060'054000-          STA     3,INOBP

00061'034001$          LDA     3,C177
00062'173405           AND     3,2,SNR
00063'050001-          STA     2,INOBC ; CLEAR 'BC' FOR NULL BYTE

00064'156415           IFEQ    2,3     ; TEST FOR RUBOUT
00065'000533           JMP     RUBOUT

00066'034571           LDA     3,C40
00067'156032           IFLT    2,3     ; TEST FOR CONTROL CHARACTER
00070'004454           JSR     CCH     ; YES - CONTROL CHARACTER

00071'010561 CH:       ISZ     CUSR    ; NORMAL CHARACTER
00072'050002-          STA     2,INOCH ; STORE

00073'034557 ACSR:     LDA     3,CUSR  ; ADJUST CURSOR
00074'030571           LDA     2,CMAX
00075'172032           IFLT    3,2
00076'000404           JMP     RESTO

00077'030557           LDA     2,C100
00100'156400           SUB     2,3     ; SIMULATE ROLL
00101'054551           STA     3,CUSR

00102'030005$RESTO:    LDA     2,SAV2
00103'034006$          LDA     3,SAVC
00104'175100           MOVL    3,3
00105'002003$          JMP     @RTI    ; RETURN FROM INTERRUPT

00106'054543 SLINK:    STA     3,LINK
00107'000764           JMP     ACSR

             ; OUTPUT LAST CHARACTER

00110'175200 INON:     MOVR    3,3
00111'054006$          STA     3,SAVC
00112'050005$          STA     2,SAV2
00113'030002-          LDA     2,INOCH
00114'071151           DOAS    2,INO
00115'002534           JMP     @LINK

             ; FOR EXAMPLE: CARRIAGE RETURN SERVICE

00116'050002-CR:       STA     2,INOCH
00117'034533           LDA     3,CUSR
```

```
0004  INODR
00120'030534           LDA     2,M100
00121'157400           AND     2,3
00122'054530           STA     3,CUSR
00123'004763           JSR     SLINK    ; OUTPUT CR
00124'030002$          LDA     2,C12
00125'176000           ADC     3,3
00126'054536           STA     3,LFLAG
00127'050002-LF1:      STA     2,INOCH
00130'034522           LDA     3,CUSR
00131'030525           LDA     2,C100
00132'157000           ADD     2,3
00133'054517           STA     3,CUSR
00134'004752 NLINK:    JSR     SLINK    ; OUTPUT LF OR ANY LAST CHAR.
                ; NORMAL LINK AGAIN
00135'014001-NOCH:     DSZ     INOBC
00136'000714           JMP     NBYTE
00137'030005$LBYTE:    LDA     2,SAV2
00140'034006$          LDA     3,SAVC
00141'175100           MOVL    3,3
00142'006004$          JSR     @POSTI
00143'000271'          INOE2
                ; CONTROL CHARACTER
00144'157000 CCH:      ADD     2,3      ; COMPUTE TABLE ENTRY
00145'003707           JMP     @ILS-CH,3
                ; CONTROL CHARACTER ROUTINES
00146'034504 CSAVE:    LDA     3,CUSR   ; SAVE CURSOR
00147'054504           STA     3,SCUSR
00150'000765           JMP     NOCH
00151'034502 CRSTR:    LDA     3,SCUSR
00152'054500           STA     3,CUSR
00153'054513           STA     3,TABC
00154'034507           LDA     3,C10    ; HOME CHAR.
00155'054002-CRS1:     STA     3,INOCH
00156'034510           LDA     3,TABC
00157'030477           LDA     2,C100
00160'156423           SUBZ    2,3,SNC
00161'000405           JMP     CRS2     ; RESULT -VE
00162'054504           STA     3,TABC
00163'004723           JSR     SLINK    ; OUTPUT CHAR.
00164'034474           LDA     3,C35    ; CURSOR DOWN CHAR.
00165'000770           JMP     CRS1
00166'034474 CRS2:     LDA     3,C31    ; CURSOR RIGHT
00167'054500           STA     3,SPCH
00170'010476           ISZ     TABC
00171'000404           JMP     CRS4
```

```
0005  INODR
00172'004714 CRS3:   JSR    SLINK
00173'034474         LDA    3,SPCH
00174'054002-        STA    3,INOCH
00175'014471 CRS4:   DSZ    TABC
00176'000774         JMP    CRS3
00177'000735         JMP    NLINK    ; OUTPUT LAST CHAR.
00200'034452 TAB:    LDA    3,CUSR
00201'030454         LDA    2,M10    ; -10
00202'157400         AND    2,3
00203'156400         SUB    2,3
00204'030446         LDA    2,CUSR
00205'054445         STA    3,CUSR
00206'156400         SUB    2,3
00207'054457         STA    3,TABC
00210'034447         LDA    3,C40
00211'054456         STA    3,SPCH
00212'000762         JMP    CRS4-1
00213'010451 LF:     ISZ    LFLAG
00214'000713         JMP    LF1
00215'000720         JMP    NOCH     ; IGNORE FIRST LF AFTER CR
00216'034434 HOME:   LDA    3,CUSR   ; RESET
00217'000420         JMP    CSRDN+1
00220'034432 RUBOUT: LDA    3,CUSR
00221'175015         IFZ    3,3
00222'000407         JMP    RO1      ; UNDERFLOW. LEFT ONLY
00223'034436         LDA    3,C32
00224'054002-        STA    3,INOCH
00225'004661         JSR    SLINK    ; OUTPUT CURSOR LEFT
00226'034431         LDA    3,C40
00227'054002-        STA    3,INOCH
00230'004656         JSR    SLINK    ; OUTPUT SPACE
00231'030430 RO1:    LDA    2,C32    ; CURSOR LEFT AGAIN
00232'176521 CSRLT:  SUBZL  3,3,SKP  ; +1.
00233'034423 CSRUP:  LDA    3,C100   ; +64.
00234'000403         JMP    CSRDN+1
00235'176001 CSRRT:  ADC    3,3,SKP  ; -1.
00236'034416 CSRDN:  LDA    3,M100   ; -64.
00237'050002-        STA    2,INOCH
00240'030412         LDA    2,CUSR
00241'172400         SUB    3,2
00242'034423         LDA    3,CMAX
00243'151112         IFM    2,2
00244'173000         ADD    3,2
00245'156033         IFGE   2,3
00246'172400         SUB    3,2
00247'050403         STA    2,CUSR
00250'000664         JMP    NLINK
```

```
 0006   INODR
00251'000135'LINK:   NOCH
00252'000000 CUSR:   0
00253'000000 SCUSR:  0
00254'177700 M100:   -100
00255'177770 M10:    -10
00256'000100 C100:   100
00257'000040 C40:    40
00260'000035 C35:    35
00261'000032 C32:    32
00262'000031 C31:    31
00263'000010 C10:    10
00264'000000 LFLAG:  0
00265'002400 CMAX:   2400
00266'000000 TABC:   0
00267'000000 SPCH:   0

00270'100000 INOE1:  100000
00271'000000 INOE2:  0

             .END
```

0007 INODR

ACSR	000073'	3/30	3/45						
C10	000263'	4/43	6/11						
C100	000256'	3/35	4/12	4/47	5/44	6/06			
C12	000002$X	4/06							
C177	000001$X	3/16							
C31	000262'	4/56	6/10						
C32	000261'	5/33	5/41	6/09					
C35	000260'	4/53	6/08						
C40	000257'	3/23	5/18	5/37	6/07				
CCH	000144'	3/25	4/31						
CH	000071'	2/29	2/45	3/27	4/32				
CMAX	000265'	3/31	5/52	6/13					
CR	000116'	2/27	3/58						
CRS1	000155'	4/45	4/54						
CRS2	000166'	4/49	4/56						
CRS3	000172'	5/02	5/06						
CRS4	000175'	4/59	5/05	5/20					
CRSTR	000151'	2/16	4/40						
CSAVE	000146'	2/15	4/36						
CSRDN	000236'	2/43	5/27	5/45	5/48				
CSRLT	000232'	2/40	5/43						
CSRRT	000235'	2/39	5/47						
CSRUP	000233'	2/42	5/44						
CUSR	000252'	3/27	3/30	3/37	3/59	4/03	4/11	4/14	4/36
		4/41	5/10	5/14	5/15	5/26	5/29	5/50	5/57
		6/02							
HOME	000216'	2/22	2/26	5/26					
ILS	000000'	2/14	4/32						
INOBC	000001-	2/07	2/59	3/01	3/18	4/20			
INOBP	000000-	2/06	3/08	3/14					
INOCH	000002-	2/08	3/28	3/52	3/58	4/10	4/45	5/04	5/34
		5/38	5/49						
INOE1	000270'	3/06	6/17						
INOE2	000271'	4/27	6/18						
INOFB	000040'	2/49							
INON	000110'	3/02	3/49						
INOPB	000043'	2/55							
INOS	000044'	2/59							
LBYTE	000137'	2/14	4/23						
LF	000213'	2/24	5/22						
LF1	000127'	4/10	5/23						
LFLAG	000264'	4/08	5/22	6/12					
LINK	000251'	3/44	3/54	6/01					
M10	000255'	5/11	6/05						
M100	000254'	4/01	5/48	6/04					
NB1	000054'	2/51	3/10						
NBYTE	000052'	3/08	4/21						
NLINK	000134'	4/16	5/08	5/58					
NOCH	000135'	2/17	2/18	2/19	2/20	2/21	2/25	2/28	2/30
		2/31	2/32	2/33	2/34	2/35	2/36	2/37	2/38
		2/41	2/44	4/20	4/38	5/24	6/01		
POSTI	000004$X	3/05	4/26						
RESTO	000102'	3/33	3/39						
RO1	000231'	5/31	5/41						
RTI	000003$X	3/42							
RUBOU	000220'	3/21	5/29						
SAV2	000005$X	3/39	3/51	4/23					
SAVC	000006$X	3/40	3/50	4/24					
SCUSR	000253'	4/37	4/40	6/03					
SLINK	000106'	3/44	4/04	4/16	4/52	5/02	5/35	5/39	
SPCH	000267'	4/57	5/03	5/19	6/15				
TAB	000200'	2/23	5/10						
TABC	000266'	4/42	4/46	4/51	4/58	5/05	5/17	6/14	

```
0001  CES

              ; COUNTED EVENTS SCHEDULER
              ; ------------------------

              ; TASK SCHEDULER MK. V

              ; E. WULFF       29-7-70
              ; MODIFIED       19-APR-71, 6-MAR-, 19-MAR-72

              ; THIS IS A RE-ENTRANT PROGRAM IN TWO SECTIONS.
              ; EACH SECTIONS FORMS A SEPERATE TASK AND MUST BE
              ; SUPPLIED WITH A SEPERATE TCB. THE TWO TASKS
              ; TOGETHER ARE CAPABLE OF SUPPORTING A NUMBER
              ; OF DEVICES SUPPLYING A SERIES OF INTERRUPTS TO
              ; THE COMPUTER. THE FIRST TASK IS CALLED BY A .SVC
               ; TO ENTER AN EVENT INTO AN EVENT QUEUE. THE
              ; SECOND TASK IS POSTED FROM THE DEVICE SERVICE
              ; ROUTINE ASSOCIATED WITH THE QUEUE WHEN THE EVENT
              ; AT THE HEAD OF THE QUEUE IS DUE. IT THEN LOADS
              ; A COUNT INTO THE DEVICE SERVICE ROUTINE
              ; AND WAITS FOR THIS COUNT TO BE DECREMENTED
              ; TO ZERO.
              ; THE TWO TASK SHARE A COMMON WORKAREA WHOSE
              ; STARTING POINT MUST BE INITIALISED IN TL7 AND TAC2
              ; OF BOTH TCB'S. THE WORKAREA IS 6 WORDS LONG.

              .TITL   CES

              .ENT    CESE,CESP,DUMMY,RETEX
              .EXTN   .WAIT,.EXIT,.RGET,FREE
              .EXTD   COMP,PMASK

              ; DEFINITIONS FOR SUPPORTING PROGRAMS

       000000 EC=     0       ;EVENT CONTROL WORD BETWEEN SECTIONS
       000001 FRE=    1       ;POINTER TO 1ST ENTRY ON FREE LIST
       000002 HD=     2       ;POINTER TO HEAD OF QUEUE
       000003 CC=     3       ;ABSOLUTE ADDRESS OF TCC
       000004 TCC=    4       ;TEMP CLOCK COUNT
       000005 CCP=    5       ;POINTS TO INSTRUCTION WHICH
                              ;COUNTS INTERRUPTS
```

```
0002   CES

        ; SECTION 1

        ; ENTER AN EVENT INTO AN EVENT QUEUE.
        ; THIS SECTION IS CALLED BY A SUPERVISOR CALL.

        ; CALLING SEQUENCE:

        ;       .SVC
        ;       (TCB ADDRESS FOR APPROPRIATE DEVICE)
        ;       (ECW ADDRESS) OR @(POINTER TO ECW ADDRESS)
        ;       (DELAY) OR @(ADDRESS CONTAINING DELAY)
        ;       (NEXT STATEMENT)

        ; TIMING STARTS IMMEDIATELY THE CALL IS MADE
        ; ANY TASK CAN WAIT ON THE COMPLETION OF THE
        ; EVENT, WHICH WILL BE POSTED BY SECTION 2 WHEN
        ; THE NUMBER OF JIFFYS CORRESPONDING TO THE
        ; DELAY ENTERED IN THIS CALL HAVE OCCURRED.

        ; NOTE:
        ; IF THIS CALL IS REPEATED FOR THE SAME ECW,
        ; THE PREVIOUS QUEUE ENTRY IS DELETED, AND THE
        ; EVENT WILL NOT BE POSTED. ONLY THE LATEST
        ; ENTRY WILL BE POSTED.

        ; ENTER A REQUEST FOR 'DELAYED EXECUTION' OF
        ; A SUBROUTINE.

        ; CALLING SEQUENCE:
        ;       .SVC
        ;       @(TCB ADDRESS FOR APPROPRIATE DEVICE)
        ;       (ENTRY ADDRESS OF SUBROUTINE) OR @(POINTER TO ...)
        ;       (DELAY) OR @(ADDRESS CONTAINING DELAY)
        ;       (NEXT STATEMENT)

        ; THE REQUEST IS ENTERED INTO THE DELAY QUEUE FOR
        ; THE APPROPRIATE DEVICE, AND CONTROL RETURNS TO
        ; THE NEXT STATEMENT IMMEDIATELY. WHEN THE DELAY TIME
        ; HAS EXPIRED, THE SUBROUTINE, WHOSE ENTRY POINT ADDRESS
        ; IS GIVEN IN THE 2ND WORD AFTER '.SVC',IS EXECUTED
        ; AT HIGH PRIORITY BY THE SUPERVISOR. THE SUBROUTINE
        ; SHOULD BE SHORT.

        ; NOTE:
        ; ALL ENTRIES FOR SUBROUTINE EXECUTION ARE RETAINED
        ; AND EXECUTED, EVEN IF A OTHER REQUESTS FOR THE
        ; SAME SUBROUTINE ARE MADE BEFORE THE FIRST HAS OCCURRED.

        ; NOTE: THE DELAY MUST BE LESS THAN 215 JIFFYS
        ;       IF PLACED IN THE CALL DIRECT, OR LESS THAN
        ;       216 JIFFYS IF POINTED TO BY AN ADDRESS IN THE
        ;       CALL. NOTE ALSO THAT THE INDIRECT CHAIN FOR THE
        ;       DELAY PROCEEDS ONLY 1 LEVEL, WHEREAS THE CHAIN
        ;       FOR THE ECW ADDRESS PROCEEDS AS LONG AS @'S ARE
        ;       ENCOUNTERED.

        ; DESTROYED:    AC3 ONLY
```

```
0003  CES

               .NREL

00000'011404 CESE:   ISZ    TPCC,3  ; INCREMENT RETURN ADRESS
00001'011404         ISZ    TPCC,3  ; IN CALLING TASK'S TCB
00002'035400         LDA    3,TAC3,3; GET CALLING ADDRESS

00003'031400         LDA    2,0,3   ; CALL IDENTIFICATION
00004'151100         MOVL   2,2     ; BIT 0 -) CARRY
00005'031401         LDA    2,1,3   ; ECW ADDRESS OR POINTER
00006'102461         CLA    0,0,SKP ; CLEAR AC0 FOR TEMP. CLOCK

00007'031000         LDA    2,0,2   ; EMULATE INDIRECT CHAIN
00010'151112         IFM    2,2     ; TEST IF POINTER
00011'000776         JMP    .-2     ; YES - TRY AGAIN
00012'145000         MOV    2,1     ; PRESERVE BIT 0 IN AC2
00013'127200         ADDR   1,1     ; MARK QUEUE ENTRY BIT 0
00014'044040         STA    1,40    ; SAVE ECW ADDRESS OR
                                    ; SUBROUTINE ENTRY ADDRESS

00015'035402         LDA    3,2,3   ; DELAY OR ADDRESS OF DELAY
00016'175113         IFZP   3,3     ; TEST IF ADDRESS
00017'165001         MOV    3,1,SKP ; NO - MOVE DELAY TO AC1
00020'025400         LDA    1,0,3   ; YES - GET DELAY (ONLY 1 LEVEL
                                    ; OF INDIRECT EMULATION)
00021'034007         LDA    3,7     ; WORK AREA ADDRESS IN AC3
00022'041404         STA    0,TCC,3 ; CLEAR TEMP. CLOCK
00023'013405         ISZ    @CCP,3  ; DISCONNECT CLOCK ROUTINE

               ; CLOCK INTERRUPTS NO LONGER DISTURB THE QUEUE ENTRIES.
               ; THEY ARE DIVERTED TO LOCATION TCC IN THE WORK AREA.

               ; START SCAN WITH PSEUDO EVENT IN WORK AREA.

00024'054031 SCAN:   STA    3,31    ; SAVE PREVIOUS ENTRY ADDRESS
00025'035402         LDA    3,2,3   ; GET THIS ENTRY ADDRESS
00026'021400         LDA    0,0,3   ; GET INCREMENT IN ENTRY
00027'106423         SUBZ   0,1,SNC ; SUBTRACT FROM DELAY
00030'000424         JMP    INSRT   ; AC0 ) AC1
                                    ; AC0(= AC1
00031'021401         LDA    0,1,3   ; GET ECW ADDRESS IN ENTRY
00032'112414         IFNE   0,2     ; IS THIS THE ECW TO BE INSERTED
00033'000771         JMP    SCAN    ; NO - PROCEED
```

```
0004  CES

                ; DELETE THIS ENTRY FROM THE QUEUE

00034'021400            LDA     0,0,3    ; GET INCREMENT AGAIN
00035'107000            ADD     0,1      ; RESTORE DELAY TO PREV. ENTRY
00036'045401            STA     1,1,3    ; SAVE IN 2ND WORD OF ENTRY
00037'027402            LDA     1,@2,3   ; GET NEXT INCREMENT
00040'107000            ADD     0,1      ; LENGTHEN TO SPAN DEL. ENTRY
00041'047402            STA     1,@2,3   ; STORE IN NEXT ENTRY
00042'025401            LDA     1,1,3    ; RESTORE DELAY TO PREV. ENTRY

00043'030007            LDA     2,7      ; PUT DELETED ENTRY ON FREE LIST
00044'021001            LDA     0,FRE,2
00045'041400            STA     0,0,3
00046'055001            STA     3,FRE,2

00047'021402            LDA     0,2,3    ; MOVE POINTER TO THIS ENTRY
00050'034031            LDA     3,31     ; FROM THIS ENTRY
00051'041402            STA     0,2,3    ; TO PREVIOUS ENTRY
00052'030040            LDA     2,40     ; ECW ADDRESS AGAIN
00053'000752            JMP     SCAN+1   ; CONTINUE SCAN

                ; INSERT A NEW ENTRY INTO THE QUEUE. IF ENTRIES ARE
                ; COINCIDENT, NEW ENTRY IS INSERTED AFTER THE OLD ENTRY.

00054'124400 INSRT:     NEG     1,1      ; INCREMENT FROM NEW TO NEXT
00055'045400            STA     1,0,3    ; STORE IN NEXT ENTRY
00056'122400            SUB     1,0      ; INCREMENT FROM PREV. ENTRY
00057'054021            STA     3,21     ; SAVE POINTER TO NEXT ENTRY

00060'030007 TRY:       LDA     2,7      ; WORK AREA ADDRESS
00061'025001            LDA     1,FRE,2  ; TRY TO GET ENTRY FROM LOCAL
                                         ; FREE LIST
00062'125015            IFZ     1,1      ; IS IT EMPTY?
00063'000433            JMP     MORE     ; YES - GET MORE CORE
                                         ; NO-PROCEED WITH ENTRY OBTAINED
00064'034031            LDA     3,31     ; POINTER TO PREV. ENTRY
00065'045402            STA     1,2,3    ; LINK NEW ENTRY TO PREV. ENTRY
00066'135000            MOV     1,3      ; POINTER TO NEW ENTRY
00067'027001            LDA     1,@FRE,2; RE-LINK FREE LIST
00070'045001            STA     1,FRE,2

00071'041400            STA     0,0,3    ; STORE INCREMENT IN NEW ENTRY
00072'024040            LDA     1,40     ; ECW FOR NEW ENTRY
00073'045401            STA     1,1,3
00074'024021            LDA     1,21     ; POINTER TO NEXT ENTRY
00075'045402            STA     1,2,3

00076'060 277           INTDS            ; SECURE THIS SHORT SECTION
00077'017005            DSZ     @CCP,2   ; CCP BACK TO QUEUE
00100'023002            LDA     0,@HD,2  ; INCREMENT TO 1ST ENTRY
00101'025004            LDA     1,TCC,2  ; CLOCK COUNT DURING SERVICE
00102'123000            ADD     1,0      ; ADD TO OBTAIN DIFFERENCE
00103'043002            STA     0,@HD,2  ; UPDATE 1ST ENTRY

00104'100532            NEGZL#  0,0,SZC  ; IS NEXT EVENT DUE
00105'000406            JMP     FIN      ; NO - POSITIVE OR 215
                                         ; YES - 0,MINUS BUT NOT 215
```

```
0005  CES

                ; POST ECW IN FIRST WORD OF WORK AREA TO ACTIVATE SECT.2

00106'035000            LDA      3,EC,2   ; NO NEED TO TEST WAIT COUNT
00107'175134            MOVZL#   3,3,SZR  ; TEST IF WAITING
00110'057417            STA      3,@TBP,3; YES - ACTIVATE TASK
00111'034001$           LDA      3,COMP   ; EITHER WAY SET COMPLETION
00112'055000            STA      3,EC,2   ; BIT IN ECW

00113'060177 FIN:       INTEN             ; ENABLE MOMENTARILY TO IMPROVE
                                          ; LATENCY
00114'177777            .EXIT             ; SUSPEND THIS TASK UNTIL CALLED
00115'000663            JMP      CESE     ; AGAIN

                ; GET MORE CORE FROM THE MAIN FREE LIST (D.Q.)

00116'177777 MORE:      .RGET             ; GET 1 CELL FROM FREE D.Q.
00117'177777            FREE              ; AC2 CONTAINS ADDRESS
                                          ; LOC 20 IS DESTROYED
00120'024030            LDA      1,30     ; LOC 30 CONTAINS CELL LENGTH
00121'034414            LDA      3,M2     ; -2
00122'157000            ADD      2,3      ; INCREASE CELL SIZE BY USING
                                          ; 2 LINK WORDS. CELL WILL NEVER
00123'030007            LDA      2,7      ; BE RETURNED TO FREE D.Q.
00124'055001            STA      3,FRE,2  ; LINK TO LOCAL FREE LIST
00125'030411            LDA      2,C3     ; +3

00126'157000 LINKF:     ADD      2,3      ; NEXT ENTRY
00127'055775            STA      3,-3,3   ; STORE POINTER IN PREV. ENTRY
00130'146426            SUBZ     2,1,SEZ  ; IS CELL EXHAUSTED
00131'000775            JMP      LINKF    ; NO - MAKE ANOTHER

00132'126460            CLA      1,1      ; TERMINATE LOCAL FREE LIST
00133'045775            STA      1,-3,3   ; WITH 0 IN POINTER POSITION
                                          ; OF LAST ENTRY
00134'000724            JMP      TRY      ; TRY AGAIN, NOW WITH SUCCESS

00135'177776 M2:        -2                ; CONSTANTS
00136'000003 C3:        3
```

```
0006  CES

                  ; SECTION 2

                  ; THE EVENT AT THE HEAD OF THE QUEUE IS NOW
                  ; DUE. POST IT, IF THE QUEUE ENTRY IS AN ECW ADDRESS,
                  ; OR EXECUTE A SUBROUTINE, IF THE QUEUE ENTRY POINTS
                  ; TO A SUBROUTINE. DELETE IT FROM THE QUEUE.
                  ; TEST IF THE NEXT EVENT IS DUE AND REPEAT.
                  ; IF NOT SET UP A VALUE FOR THE APPROPRIATE DEVICE
                  ; TO DECREMENT AND WAIT FOR THE DEVICE TO DECREMENT
                  ; THIS VALUE TO ZERO. THE TASK ALSO CIRCULATES
                  ; 3 DUMMY EVENTS IN THE EVENT QUEUE. THESE ARE
                  ; RE-INITIALISED BY CALLING AN EXTRA ROUTINE VIA
                  ; THE POINTER 'DUMMY'. THIS RESETS THE COUNT
                  ; TO 215 AND PUTS THE ENTRY ON THE VERY END OF THE
                  ; EVENT QUEUE, USING A CONSTANT WHICH IS ASSEMBLED
                  ; INTO THE 4TH WORD OF DUMMY ENTRIES. DUMMY ENTRIES
                  ; ARE NOT RETURNED TO A FREE LIST. THE 3 DUMMY ENTRIES
                  ; ENSURE THAT THERE IS AT LEAST 1 EVENT WHICH
                  ; IS DUE MORE THAN 216 JIFFYS FROM NOW.

                  ; THIS TASK MASKS INTERRUPTS FROM ALL DEVICES NORMALLY.
                  ; (THIS MAY BE RELAXED IN SPECIAL CIRCUMSTANCES BY
                  ; SETTING UP A DIFFERENT MASK THAN 177777 IN THE TCB)
                  ; IF THIS TASK LOOPS INTERNALLY A NUMBER OF TIMES,
                  ; CLOCK INTERRUPTS MIGHT BE MISSED. THUS SELECTED
                  ; INTERRUPTS ARE ALLOWED MOMENTARILY. SET UP AN APPRO-
                  ; PRIATE MASK IN TL6 OF THE WORK AREA.

00137'020021 MASK:  LDA     0,21     ; DEVICE MASK
00140'062077        MSKO    0        ; ALLOW MORE INTERRUPTS
00141'020002$       LDA     0,PMASK  ; GO BACK TO ORIGINAL MASK
00142'062077        MSKO    0        ; USUALLY 177777

00143'034007 NEXT:  LDA     3,7      ; RESTORE POINTER
00144'031402        LDA     2,HD,3   ; ENTRY ABOUT TO BE FREED
00145'023002        LDA     0,@2,2   ; NEXT INCREMENT
00146'025000        LDA     1,0,2    ; THIS INCREMENT (0 OR -VE)
00147'123000        ADD     1,0      ; ACTUAL INCREMENT FROM NOW
00150'043002        STA     0,@2,2   ; STORE IN NEXT ENTRY

00151'025002        LDA     1,2,2    ; MAKE NEXT ENTRY HEAD OF QUEUE
00152'045402        STA     1,HD,3   ; ENTRY CUT OFF

00153'035001        LDA     3,1,2    ; ADDRESS IN THIS ENTRY
00154'175112        IFM     3,3      ; TEST IF ECW
00155'000411        JMP     EXEC     ; NO - ECECUTE SUBROUTINE

                  ; POST EVENT CONTROL WORD IN THIS QUEUE ENTRY

00156'035400        LDA     3,0,3    ; EVENT CONTROL WORD
00157'174536        NEGZL#  3,3,SEZ  ; TEST ECW
00160'015416        DSZ     TWC,3    ; TEST WAIT COUNT
00161'000402        JMP     .+2      ; NOT WAITING. ONLY POST
00162'057417        STA     3,@TBP,3; ACTIVATE TASK IN ECW

00163'176620        SUBZR   3,3      ; 100000
00164'057001        STA     3,@1,2   ; SET COMPLETION BIT IN ECW
00165'000406        JMP     ATT
```

```
 0007  CES


                ; EXECUTE AN EXTRA SUBROUTINE

 00166'040031  EXEC:  STA     0,31     ; SAVE ACC'S
 00167'050041         STA     2,41
 00170'005400         JSR     0,3      ; ENTER EXTRA ROUTINE

 00171'030041 RETEX:  LDA     2,41     ; RETURN HERE
 00172'020031         LDA     0,31     ; RESTORE ACC'S

 00173'034007 ATT:    LDA     3,7
 00174'025401         LDA     1,FRE,3  ; ATTACH ENTRY TO
 00175'045000         STA     1,0,2    ; LOCAL FREE LIST
 00176'051401         STA     2,FRE,3

 00177'100533 TEST2:  NEGZL#  0,0,SNC  ; IS NEXT EVENT DUE
 00200'000737         JMP     MASK     ; YES - 0, -VE BUT NOT 215

 00201'177777         .WAIT            ; NO - WAIT FOR CLOCK ROUTINE
 00202'100007         @7               ; TO DECREMENT THE HEAD OF THE
 00203'176460 CESP:   SUBC    3,3      ; QUEUE TO ZERO.
 00204'056007         STA     3,@7     ; CLEAR ECW
 00205'000736         JMP     NEXT

               ; RE-INSERT DUMMY EVENT WITH AN INCREMENT OF
               ; 215 (100000) JIFFYS AFTER PREVIOUS DUMMY.

 00206'025003 RDUM:   LDA     1,3,2    ; POINTER IN 4TH WORD
 00207'045002         STA     1,2,2    ; TO CLOSE CIRCLE
 00210'126620         SUBZR   1,1      ; 215
 00211'045000         STA     1,0,2    ; SET INCREMENT
 00212'000765         JMP     TEST2    ; DO NOT PUT ON FREE

        100206'DUMMY= @RDUM

               ; FORMAT OF DUMMY ENTRIES:
               ;      .ENT    DUMMY

               ; DUM1: 100000
               ;       DUMMY
               ;       DUM2
               ;       DUM2

               ; DUM2: 100000
               ;       DUMMY
               ;       DUM3
               ;       DUM3

               ; DUM3: 100000
               ;       DUMMY
               ;       DUM1
               ;       DUM1

               .END
 0009  CES

ATT     000173'    6/59    8/11
```

C3	000136'	5/26	5/39						
CC	000003	1/38							
CCP	000005	1/40	3/28	4/50					
CESE	000000'	3/04	5/13						
CESP	000203'	8/21							
COMP	000001$X	5/07							
DUMMY	100206'	8/34							
EC	000000	1/35	5/04	5/08					
EXEC	000166'	6/47	8/04						
FIN	000113'	4/57	5/10						
FRE	000001	1/36	4/13	4/15	4/32	4/40	4/41	5/25	8/12
		8/14							
FREE	000117'X	5/18							
HD	000002	1/37	4/51	4/54	6/36	6/43			
INSRT	000054'	3/39	4/26						
LINKF	000126'	5/28	5/31						
M2	000135'	5/21	5/38						
MASK	000137'	6/30	8/17						
MORE	000116'	4/35	5/17						
NEXT	000143'	6/35	8/23						
PMASK	000002$X	6/32							
RDUM	000206'	8/28	8/34						
RETEX	000171'	8/08							
SCAN	000024'	3/35	3/43	4/21					
TCC	000004	1/39	3/27	4/52					
TEST2	000177'	8/16	8/32						
TRY	000060'	4/31	5/36						
.EXIT	000114'X	5/12							
.RGET	000116'X	5/17							
.WAIT	000201'X	8/19							

```
0001  TIM

            ; ABSOLUTE TIME IN JIFFIES
            ; ----------------------

            ; XXX

            ; E. WULFF      23-APR-71

            ; SUBROUTINE WHICH RETURNS THE DOUBLE PRECISION CLOCK-
            ; COUNT IN AC0 & AC1. IT IS ESSENTIAL TO USE THIS
            ; ROUTINE TO FETCH THE HIGH ORDER TIME, BECAUSE CLOCK
            ; INTERRUPTS ARE INHIBITED WHILE THE VALUES ARE LOADED.

            ; IF THE LOW ORDER WORD ONLY IS REQUIRED, THIS MAY BE
            ; FETCHED WITH A 'LDA  X,TIML'.

            ; CALLING SEQUENCE:
            ;       .TIM   OR   JSR @TIM
            ;       (NEXT STATEMENT)

            ; PRECAUTION:   CALL ONLY IN USER PROGRAMS WHEN
            ;               INTERRUPT IS ON.

            ; OUTPUT:
            ;       AC0 = HIGH ORDER TIME
            ;       AC1 = LOW ORDER TIME
            ;       TOGETHER THESE ARE AN UNSIGNED DOUBLE PREC.
            ;       VALUE WHICH REPRESENT THE TIME MOD. 232
            ;       SINCE INITIALISATION. THE VALUE IS IN JIFFIES
            ;       AT THE CURRENT CLOCK SPEED. THE OUTPUT IS
            ;       MAINLY USED TO COMPUTE THE TIME DIFFERENCE
            ;       BETWEEN TWO EVENTS.

            ; DESTROYED:    AC0,AC1 AND AC3

            .TITL   TIM

            .ENT    TIM,.TIM
            .EXTD   TIML

             .ZREL

00000-000000'TIM:   RTIM
      006000-.TIM=  JSR     @TIM    ; DEFINE CALLING MNEMONIC

            .NREL

00000'060277 RTIM:  INTDS           ; SECURE NEXT 2 STATEMENTS
 00001'020404        LDA     0,TIMH  ; HIGH ORDER
00002'024001$       LDA     1,TIML  ; LOW ORDER
00003'060177        INTEN           ; RELEASE
00004'001400        JMP     0,3     ; RETURN

            TIMH:   ; DEFINE PLACE OCCUPIED IN 'DT'

            ;               NOTE
            ; THIS PROGRAM MUST ALWAYS BE LOADED IMMEDIATELY
            ; BEFORE 'DT', SO THAT TIMH IS CORRECTLY DEFINED.
```

```
  0002  TIM
01               .END
  0003  TIM

RTIM    000000'      1/43     1/48
TIM     000000-      1/43     1/44
TIMH    000005'      1/49     1/54
TIML    000001$X     1/50
.TIM    006000-      1/44
```

```
0001  DELAY

                 ; DELAY TIMER
                 ; -----------

                 ; TASK SCHEDULER MK. V

                 ; E. WULFF       21-APR-71
                 ; MODIFIED       14-MAR-72        4-OCT-72

                 .TITL   DELAY

                 .ENT    RTCS,TCBC,TCBD,DELAY,DELEX,TIMH,TIML
                 .ENT    TCBE,TDTIM,TDSEM,RTCW,FREE
                 .EXTD   RTI,POSTI,C377,COPY
                 .EXTN   CESE,CESP,DUMMY,.SVC,LOWER,RAISE
                 .EXTN   .WAIT,.DQIN,CL,RETEX,NC,END


                 ; REAL TIME CLOCK INTERRUPT SERVICE.

                 ; MAINTAIN A DOUBLE PRECISION UNSIGNED CLOCK COUNT. THE
                 ; LOW ORDER WORD IS IN LOCATION "TIML" ON PAGE ZERO.
                 ; THE HIGH ORDER WORD IS IN LOCATION "TIMH" ON THIS PAGE.
                 ; IT SHOULD ONLY BE OBTAINED BY A CALL TO SUBROUTINE
                 ; "TIM" WHICH RETURNS LOW ORDER TIME IN AC0 AND HIGH
                 ; ORDER IN AC1. (SEE SEPERATE ASSEMBLY)
                 ; NEVER MODIFY EITHER LOCATION, SINCE OTHER TASKS MAY
                 ; ALSO WANT TO USE THEM.
                 ; TIME IS ACCURATE TO THE NEAREST JIFFY.

                 ; ALSO SUPPORT THE CURRENT EVENT SCHEDULER FOR THE REAL
                 ; TIME CLOCK. A CLOCK WORK AREA (CWA), 3 DUMMY EVENTS
                 ; AND TASK CONTROL BLOCKS FOR SECTIONS 1 & 2 OF THE
                 ; CURRENT EVENT SCHEDULER ARE SET UP IN THIS ASSEMBLY.

                 .ZREL

 00000-000000 TIML:   0

                  .NREL

 00000'000000 TIMH:   0

 00001'060114 RTCS:   NIOS    RTC      ; SET BUSY FLAG FOR NEXT CYCLE
 00002'010000-        ISZ     TIML     ; ABSOLUTE TIME (LOW ORDER)
 00003'000403         JMP     C.INS
 00004'010774         ISZ     TIMH     ; HIGH ORDER
 00005'000401         JMP     C.INS    ; IGNORE FURTHER OVERFLOW

                 ; ACCUMULATE COUNT ON THE HEAD OF THE EVENT QUEUE

 00006'016430 C.INS:  DSZ     @CWA+2   ; THIS INSTRUCTION IS MODIFIED
 00007'002001$        JMP     @RTI     ; COUNT NOT ZERO

 00010'006002$        JSR     @POSTI   ; COUNT IS ZERO
 00011'000034'        CWA              ; POST EVENT DESPATCHER
```

```
0002  DELAY

                ; ORIGINAL OF CLOCK WORK AREA

00012'000102'OCWA:   CPOST    ; SECTION 2 EVENT CONTROL WORD (WAITING)
00013'000000         0        ; FREE - INITIALLY EMPTY
00014'000042'        DUM1     ; HD - POINTS TO FIRST DUMMY EVENT
00015'000040'        CWA+4    ; CC - POINTS TO TCC
00016'000000         0        ; TCC - ACCUMULATES CLOCK COUNT WHILE
                              ; SECTION 2 IS ACTIVE
00017'000006'        C.INS    ; CCP - POINTS TO DSZ INSTRUCTION

                ; ORIGINAL OF 3 DUMMY EVENT ENTRIES.

00020'100000         100000   ; INCREMENT
00021'177777         DUMMY    ; CALL TO SPECIAL ROUTINE
00022'000046'        DUM2     ; POINT TO NEXT DUMMY INITIALLY
00023'000046'        DUM2     ; CONSTANT VALUE

00024'100000         100000
00025'000021'        DUMMY
00026'000052'        DUM3
00027'000052'        DUM3

00030'100000         100000
00031'000025'        DUMMY
00032'000042'        DUM1     ; CLOSE THE CIRCLE
00033'000042'        DUM1

                ; WORKING COPIES

      000006 CWA:    .BLK    6
      000004 DUM1:   .BLK    4
      000004 DUM2:   .BLK    4
      000004 DUM3:   .BLK    4

                ; TASK CONTROL BLOCK FOR SECTION 1 OF CES

      000020 TCBC:   .BLK    20
00076'006401 1B4+1B5+1B7+1B15; TASK INITIALLY SUSPENDED (B15)
00077'177777         CESE     ; PC POINTS TO START OF SECTION 1
00100'000003         3        ; MASK TTI, TTO
00101'000034'        CWA      ; L7 - CLOCK WORK AREA

      000056'DELAY=  TCBC     ; NAME FOR ENTERING EVENT IN TIME QUEUE
      100056'DELEX=  @DELAY   ; NAME FOR ENTERING REQEST FOR EXEC.

                ; TASK CONTROL BLOCK FOR SECTION 2 OF CES

      000020 TCBD:   .BLK    20
00122'007413 1B4+1B5+1B6+1B7+1B12+1B14+1B15  ; INITIALLY SUSPENDED
00123'177777         CESP     ; POINTS TO START OF SECTION 2
00124'177777         -1       ; MASK ALL DEVICES
00125'000003         3        ; L6 - TEMP. MASK WHEN LOOPING
00126'000034'        CWA      ; L7 - CLOCK WORK AREA
00127'177777         RETEX    ; L40 - RETURN AFTER EXEC.
00130'000001         1        ; WC

      000102'CPOST=  TCBD
```

```
0003  DELAY

                 ; TASK CONTROL BLOCK FOR 1 MIN. LOOP

         000020 TCBE:   .BLK 20

   00151'006000         1B4+1B5 ; ICW
   00152'000163'        STCLK   ; PC
   00153'177773         -1-1B13 ; MASK ALL DEVICES EXCEPT RTC

                 ; FREE LIST D.Q. CONTROL BLOCK

         000002         .BLK 2  ; LEFT & RIGHT LINK
         000005 FREE:   .BLK 5  ; SEMAPHORES & CELL LENGTH

         000003 RTCSP=  3       ; 1 KHZ CLOCK SPEED

                 ; INITIALISATION PROGRAM

   00163'006004$STCLK:  JSR     @COPY    ; COPY CLOCK WORK AREA
   00164'000012'        OCWA             ; AND DUMMY ENTRIES
   00165'000034'        CWA              ; ADDRESS OF COPY
   00166'000022         6+4+4+4          ; NO. OF WORDS

   00167'177777         .DQIN   ; SET UP THE FREE D.Q.
   00170'000156'        FREE
   00171'177777         CL      ; CELL LENGTH IN BYTES
   00172'177777         NC      ; NUMBER OF CELLS
   00173'177777         END     ; BEGINNING OF UNUSED STORAGE

   00174'020451         LDA     0,RTCW   ; SET CLOCK SPEED
   00175'061114         DOAS    0,RTC    ; START CLOCK

   00176'102520         SUBZL   0,0
   00177'040441         STA     0,TDSEM ; INITIALISE THE SEMAPHORE
   00200'102460         SUBC    0,0
   00201'040440         STA     0,TDSEM+1
```

```
0004  DELAY

                     ; ONE MINUTE LOOP

00202'176460 TDLOO:  SUBC    3,3
00203'054437         STA     3,TDECW ; CLEAR THE ECW

00204'177777         .SVC    ; ENTER A 1 MIN. DELAY
00205'000056'        DELAY
00206'000242'        TDECW
00207'100246'        @TCNT

00210'177777         LOWER   ; SECURE THE TIME LOCATIONS
00211'000240'        TDSEM   ; FROM MODIFICATION DURING INCREMENT

00212'020431         LDA     0,TDADR ; ADDRESS OF TIME LOC. TABLE
00213'040431         STA     0,TDPNT ; INITIALISE POINTER

00214'004447         JSR     TDINC   ; INCREMENT MINUTES
00215'000073         0.B7+59.

00216'004445         JSR     TDINC   ; INCREMENT HOURS
00217'000027         0.B7+23.

00220'004443         JSR     TDINC   ; INCREMENT DAYS
00221'100247'        @DAY            ; VIA A TABLE

00222'004441         JSR     TDINC   ; INCREMENT MONTHS
00223'000414         1.B7+12.

00224'004437         JSR     TDINC   ; INCREMENT YEARS
00225'000143         0.B7+99.

00226'177777 COMPL:  RAISE   ; RELEASE TIME LOCATIONS
00227'000240'        TDSEM

00230'177777         .WAIT   ; WAIT FOR NEXT 1 MIN.
00231'000242'        TDECW
00232'000750         JMP     TDLOO
```

```
0005  DELAY

   00233'000000 TDTIM:  0.       ; TIME LOCATIONS - MIN.
   00234'000000         0.       ; HOUR
   00235'000001         1.       ; DAY
   00236'000006         6.       ; MONTH
   00237'000110         72.      ; YEAR

   00240'000001 TDSEM:  1        ; SEMAPHORE
   00241'000000         0
   00242'000000 TDECW:  0        ; EVENT CONTROL WORD
   00243'000232'TDADR:  TDTIM-1
         000001 TDPNT:  .BLK 1
   00245'000003 RTCW:   RTCSP
                TCNT:            ; JIFFIES IN 1 MIN.
                .IFE    RTCSP    ; 50 HZ
                        3000.
                .ENDC
                .IFE    RTCSP-1  ; 10 HZ
                        600.
                .ENDC
                .IFE    RTCSP-2  ; 100 HZ
                        6000.
                .ENDC
         000000 .IFE    RTCSP-3  ; 1 KHZ
   00246'165140         60000.
                .ENDC

                ; TABLE OF DAYS IN THE MONTH'S OF THE YEAR

                DAY:
   00247'000437 1.B7+31.         ; JANUARY
   00250'100434 @1.B7+28.        ; FEBRUARY (ALLOWS FOR LEAP YEAR)
   00251'000437 1.B7+31.         ; MARCH
   00252'000436 1.B7+30.         ; APRIL
   00253'000437 1.B7+31.         ; MAY
   00254'000436 1.B7+30.         ; JUNE
   00255'000437 1.B7+31.         ; JULY
   00256'000437 1.B7+31.         ; AUGUST
   00257'000436 1.B7+30.         ; SEPTEMBER
   00260'000437 1.B7+31.         ; OCTOBER
   00261'000436 1.B7+30.         ; NOVEMBER
   00262'000437 1.B7+31.         ; DECEMBER
```

```
 0006  DELAY
01
02              ; SUBROUTINE TO INCREMENT A WORD POINTED
03              ; TO BY "TDPNT"+1 ACCORDING TO A MODULUS STORED
04              ; IN BITS 8-15 OF THE WORD AFTER THE CALL
05              ; AND RESTORED AFTER OVERFLOW TO A VALUE
06              ; STORED IN BIT 1 TO 7 OF THE WORD AFTER THE
07              ; CALL. IF BIT 0 OF THE WORD AFTER THE CALL IS 1
08              ; BITS 1 TO 15 ARE USED AS A BASE ADDRESS
09              ; TO COMPUTE AN ADDRESS IN THE DAYS OF THE
10              ; MONTH TABLE. THIS WORD IS USED TO OBTAIN
11              ; A MODULUS AND RESTORE VALUE.
12              ; IF NO OVERFLOW OCCURS, THE ROUTINE RETURNS
13              ; TO A LOCATION "COMPL". NO INCREMENTING OF
14              ; HIGHER VALUES ARE REQUIRED. IF OVERFLOW
15              ; OCCURS THE ROUTINE RETURNS TO THE 2ND WORD
16              ; AFTER THE CALL.
17
18 00263'010761 TDINC:  ISZ     TDPNT   ; INCREMENT THE POINTER
19 00264'031400         LDA     2,0,3   ; WORD AFTER THE CALL
20 00265'151113         IFZP    2,2
21 00266'000413         JMP     TDI1    ; USE WORD AS IS
22
23 00267'024747         LDA     1,TDTIM+3; MONTH VALUE
24 00270'133000         ADD     1,2     ; COMPUTE ADDRESS IN DAY OF
25 00271'031377         LDA     2,-1,2  ; MONTH TABLE, GET WORD
26 00272'151113         IFZP    2,2     ; CHECK IF FEBRUARY
27 00273'000406         JMP     TDI1    ; NO
28
29 00274'024743         LDA     1,TDTIM+4; YEAR
30 00275'125203         MOVR    1,1,SNC
31 00276'125202         MOVR    1,1,SZC ; CHECK IF LEAP YEAR
32 00277'000402         JMP     TDI1    ; NO
33 00300'151400         INC     2,2     ; YES, MAKE FEB. 29 DAYS
34
35 00301'024003$TDI1:   LDA     1,C377
36 00302'147400         AND     2,1     ; MODULUS-1+RESET
37 00303'012741         ISZ     @TDPNT  ; INCREMENT THE VALUE
38 00304'022740         LDA     0,@TDPNT
39 00305'106432         IFLE    0,1     ; HAS IT OVERFLOWN
40 00306'000720         JMP     COMPL   ; YES - NO FURTHER WORK
41
42 00307'024404         LDA     1,C774. ; 77400 - MASK FOR BIT 1-7
43 00310'147700         ANDS    2,1     ; RESET VALUE  0 OR 1
44 00311'046733         STA     1,@TDPNT; RESET THE VALUE
45 00312'001401         JMP     1,3     ; RETURN TO ALLOW NEXT
46                                      ; VALUE TO BE INCREMENTED
47
48 00313'077400 C774.:  77400
49
50              .END
  0007  DELAY

C377    000003$X    6/35
C774.   000313'     6/42    6/48
```

CESE	000077'X	2/40						
CESP	000123'X	2/51						
CL	000171'X	3/26						
COMPL	000226'	4/33	6/40					
COPY	000004$X	3/19						
CPOST	000102'	2/04	2/58					
CWA	000034'	1/52	1/56	2/07	2/31	2/42	2/54	3/21
C.INS	000006'	1/46	1/48	1/52	2/10			
DAY	000247'	4/25	5/30					
DELAY	000056'	2/44	2/45	4/08				
DELEX	100056'	2/45						
DUM1	000042'	2/06	2/26	2/27	2/32			
DUM2	000046'	2/16	2/17	2/33				
DUM3	000052'	2/21	2/22	2/34				
DUMMY	000031'X	2/15	2/20	2/25				
END	000173'X	3/28						
FREE	000156'	3/13	3/25					
LOWER	000210'X	4/12						
NC	000172'X	3/27						
OCWA	000012'	2/04	3/20					
POSTI	000002$X	1/55						
RAISE	000226'X	4/33						
RETEX	000127'X	2/55						
RTCS	000001'	1/44						
RTCSP	000003	3/15	5/13	5/15	5/18	5/21	5/24	
RTCW	000245'	3/30	5/13					
RTI	000001$X	1/53						
STCLK	000163'	3/07	3/19					
TCBC	000056'	2/38	2/44					
TCBD	000102'	2/49	2/58					
TCBE	000131'	3/04						
TCNT	000246'	4/10	5/14					
TDADR	000243'	4/15	5/11					
TDECW	000242'	4/05	4/09	4/37	5/10			
TDI1	000301'	6/21	6/27	6/32	6/35			
TDINC	000263'	4/18	4/21	4/24	4/27	4/30	6/18	
TDLOO	000202'	4/04	4/38					
TDPNT	000244'	4/16	5/12	6/18	6/37	6/38	6/44	
TDSEM	000240'	3/34	3/36	4/13	4/34	5/08		
TDTIM	000233'	5/02	5/11	6/23	6/29			
TIMH	000000'	1/42	1/47					
TIML	000000-	1/38	1/45					
.DQIN	000167'X	3/24						
.SVC	000204'X	4/07						
.WAIT	000230'X	4/36						

```
 0001  COPY

              ; COPY AREAS OF CORE
              ; ------------------

              ; XXX

              ; E. WULFF      16-JULY-71

              ; CALLING SEQUENCE:

              ;       JSR     @COPY
              ;       (ADDRESS OF SOURCE BLOCK)
              ;       (ADDRESS OF COPY BLOCK)
              ;       (NUMBER OF WORDS TO BE COPIED)
              ;       (NEXT STATEMENT)

              ; OR

              ;       JSR     @COPA
              ;       (NUMBER OF WORDS TO BE COPIED)
              ;       (NEXT STATEMENT)
              ;       AC1 MUST CONTAIN ADDRESS OF SOURCE BLOCK
              ;       AC2 MUST CONTAIN ADDRESS OF COPY BLOCK

              ; MODIFIED:     ALL ACC., CARRY AND L6
              ; UNCHANGED:    L7

              .TITL   COPY

               .ENT    COPY,COPA

              .ZREL

 00000-000000'COPY:   RCOPY
 00001-000004'COPA:   RCOPA

              .NREL

 00000'025400 RCOPY:  LDA     1,0,3    ; ADDRESS OF SOURCE
 00001'031401         LDA     2,1,3    ; ADDRESS OF COPY
 00002'175400         INC     3,3
 00003'175400         INC     3,3

 00004'021400 RCOPA:  LDA     0,0,3
 00005'100000         COM     0,0      ; BLOCK COUNT
 00006'175400         INC     3,3
 00007'054006         STA     3,6      ; SAVE RETURN
 00010'135000         MOV     1,3

 00011'101405 LOOP:   INC     0,0,SNR  ; STEP COUNTER
 00012'002006         JMP     @6       ; RETURN

 00013'025400         LDA     1,0,3    ; SOURCE WORD
 00014'045000         STA     1,0,2    ; STORE IN COPY
 00015'175400         INC     3,3
 00016'151400         INC     2,2
 00017'000772         JMP     LOOP

              .END
```

0002 COPY

COPA	000001-	1/35	
COPY	000000-	1/34	
LOOP	000011'	1/50	1/57
RCOPA	000004'	1/35	1/44
RCOPY	000000'	1/34	1/39

```
0002  DB1.5

                ;               ***************
                ;               *             *
                ;               *  DEBUG 1.5  *
                ;               *             *
                ;               ***************

                ; E. WULFF      25-JUN-72
                ; VERS. 18      3-OCT-72

                .TITL   DB1.5

                .ENT    DB1.5
         000001 .IFN    T
                .ENT    TCBW,PTPS
                .EXTN   .WAIT,TTIEC,TTOE1,LOWER,RAISE,SEMDT
                .EXTD   RTI
                .ENDC

                .NREL

                ; PUNCH INTERRUPT SERVICE FOR TASK MODE.

         000001    .IFN T

 00000'060213 PTPS:   NIOC    PTP      ; CLEAR DONE FLAG
 00001'002001$        JMP     @RTI     ; TEST BUSY IN TASK
                   .ENDC

                ; PUNCH BINARY TAPE.

 00002'000020 C20:    20

 00003'015562 PS:     DSZ     AFLAG-SAV0,3
 00004'000506         JMP     PROC     ; PROCEED

 00005'151100         MOVL    2,2
 00006'151220         MOVZR   2,2      ; CLEAR BIT 0
 00007'050525         STA     2,JFLAG ; FINAL ADDRESS
 00010'031564         LDA     2,SADR-SAV0,3
 00011'007415         JSR     @APUT-SAV0,3    ; ECHO

 00012'145000 BLL:    MOV     2,1
 00013'020521         LDA     0,JFLAG ; FINAL ADDRESS
 00014'106022         ADCZ    0,1,SZC ; REMAINDER
 00015'002516         JMP     @AGO     ; FINISHED

 00016'034764         LDA     3,C20    ; BLOCK SIZE
 00017'137033         ADDZ#   1,3,SNC
 00020'164400         NEG     3,1      ; LIMIT 20 WORDS
 00021'044510         STA     1,RELAD ; ORIGINAL START NOT NEEDED
 00022'121020         MOVZ    1,0
 00023'004425         JSR     PW       ; PUNCH WORD COUNT
 00024'141000         MOV     2,0
 00025'004423         JSR     PW       ; PUNCH ADDRESS
 00026'120400         NEG     1,0      ; INITIAL CHECKSUM
 00027'142400         SUB     2,0      ; TAKE OUT ADDRESS
```

```
0003  DB1.5


              ; COMPUTE THE CHECKSUM

00030'035000 CSL:    LDA     3,0,2
00031'162400         SUB     3,0
00032'151400         INC     2,2
00033'125444         INCO    1,1,SZR
00034'000774         JMP     CSL      ; COMPUTE THE CHECKSUM
00035'004413         JSR     PW       ; PUNCH CHECKSUM
00036'024473         LDA     1,RELAD  ; BLOCK LENGTH AGAIN
00037'133040         ADDO    1,2      ; RESTORE START OF BLOCK

00040'021000 PLP:    LDA     0,0,2
00041'004407         JSR     PW       ; PUNCH VALUE
00042'151400         INC     2,2
00043'125404         INC     1,1,SZR
00044'000774         JMP     PLP      ; AC2 POINTS TO START OF
                                      ; NEXT BLOCK
00045'102440         SUBO    0,0
00046'004402         JSR     PW       ; 2 NULL BYTES
00047'000743         JMP     BLL

              ; PUNCH A 16 BIT WORD IF CARRY = 0
              ; (8BITS IF CARRY = 1)

00050'061113 PW:     DOAS    0,PTP    ; PUNCH 1 BYTE
00051'063513         SKPBZ   PTP      ; DB1.5 MUST BE LOW PRIORITY
00052'000777         JMP     .-1
00053'101362         MOVCS   0,0,SZC
00054'000774         JMP     PW       ; PUNCH 2ND BYTE
00055'001400 C1400:  JMP     0,3      ; RETURN

00056'002017 BRINS:  JMP     @LINK
00057'000723'ABRA:   BRADR
00060'000367'ATTOF:  TTOFL
00061'000362'ASAV:   SAVO
00062'000377 C377:   377
00063'060000 C60K:   60000
00064'000000 INTFL:  0
00065'000000 BPN1:   0
       000001    .IFN T
00066'000562'ATFLG:  TFLAG
00067'000000 GFLAG:  0
                 .ENDC
```

```
0004  DB1.5


              ; PUNCH END BLOCK AND TRAILER (10")
              ; ENTER WITH CARRY = 0

00070'015563 ES:     DSZ     BFLAG-SAV0,3
00071'145102         MOVL    2,1,SZC
00072'131260         MOVCR   1,2      ; 100000 IF BFLAG = 1
00073'025720         LDA     1,M60-SAV0,3    ; TRAILER LENGTH
00074'007415         JSR     @APUT-SAV0,3
00075'102520         SUBZL   0,0      ; AC0 = 1, CARRY = 0
00076'004752         JSR     PW       ; PUNCH  1
00077'141020         MOVZ    2,0
00100'004750         JSR     PW       ; PUNCH ADDRESS OR 100000
00101'140021         COMZ    2,0,SKP  ; CHECKSUM

00102'102440 FLP:    SUBO    0,0      ; NULL
00103'004745         JSR     PW       ; PUNCH CHECKSUM OR NULL
00104'125404         INC     1,1,SZR
00105'000775         JMP     FLP
00106'002425         JMP     @AG0     ; FINISHED

              ; PUNCH 10" LEADER

00107'025720 FS:     LDA     1,M60-SAV0,3    ; LEADER LENGTH
00110'007415         JSR     @APUT-SAV0,3
00111'000771         JMP     FLP
```

```

                ; PROCEED FROM A BREAKPOINT.
                ; IF THE SYSTEM DID NOT ENTER VIA A BREAKPOINT
                ; OR IF DEBUG IS IN TASK MODE IT IS NOT
                ; POSSIBLE TO PROCEED.

00112'024571 PROC:   LDA     1,BPN
00113'125014         IFN     1,1
00114'001443         JMP     GO-SAV0,3

00115'151015         IFZ     2,2
00116'151400         INC     2,2      ; PROCEED COUNT AT LEAST 1
00117'051411         STA     2,PRCNT-SAV0,3
00120'007415         JSR     @APUT-SAV0,3    ; ECHO NOW
00121'012737         ISZ     @ATTOF
00122'060211         NIOC    TTO
00123'020733         LDA     0,BRINS ; MOST LIKELY PROCEED INSTR.
00124'032733         LDA     2,@ABRA ; BRADR
00125'024410         LDA     1,PRADR
00126'146414         IFNE    2,1
00127'022406         LDA     0,@PRADR; BREAK POINT HAS MOVED
00130'000464         JMP     EMI

                ; CONSTANTS AND ADDRESSES

      000017 LINK=   17
00131'000000 RELAD:  0       ; 'RELAD MUST 4 WORDS FROM 'PRADR'
00132'004000 C4K:    4000
00133'000425'AGO:    GO
00134'000000 JFLAG:  0
00135'000000 PRADR:  0

                ; RUN SERVICE

             GS:
      000001    .IFN T
00136'026730         LDA     1,@ATFLG
00137'044730         STA     1,GFLAG ; STORE TASK STATE
                .ENDC
00140'024772         LDA     1,C4K   ; 'JSR'
00141'015563 RS:     DSZ     BFLAG-SAV0,3
00142'000402         JMP     .+2     ; AC1 = 0 FOR 'R'  'JMP'
00143'031406         LDA     2,LOC-SAV0,3    ; STORED START LOCATION
00144'007415         JSR     @APUT-SAV0,3
00145'006537         JSR     @ACRLF
00146'012712         ISZ     @ATTOF
00147'060211         NIOC    TTO
00150'004472         JSR     R1

                ; RETURN AFTER SUBROUTINE EXECUTION

      000001    .IFN T
00151'054760         STA     3,RELAD
00152'034715         LDA     3,GFLAG ; RESTORE TASK STATE
00153'056713         STA     3,@ATFLG
00154'054760         STA     3,JFLAG
00155'034754         LDA     3,RELAD ; KEEP AC3
                .ENDC
```

```
0006  DB1.5

                  ; USE THIS ORGANISATION OF BREAK POINT ENTRY.
                  ; A BREAK MAY OCCUR IN A TASK AND THE TASK MAY
                  ; BE INTERRUPTED BEFORE 'INTDS' IS EXECUTED. THE
                  ; SAME BREAK POINT MAY THEN BE EXECUTED IN ANOTHER
                  ; TASK WHICH HAS DIFFERENT STATUS. THUS IT IS
                  ; IMPORTANT NOT TO MODIFY ANY WORDS IF INTERRUPT
                  ; IS ON, BEFORE IT IS DISENABLED. THE SAME APPLIES
                  ; TO MULTIPLE BREAKPOINTS.

00156'063477 DB1.5:   SKPBN   CPU     ; START HERE MANUALLY
00157'010705          ISZ     INTFL   ; INTERRUPT FLAG
00160'060277          INTDS
00161'010704          ISZ     BPN1    ; TEMP. BREAK COUNTER

00162'063477 BRK:     SKPBN   CPU     ; BREAK ENTRY
00163'010701          ISZ     INTFL
00164'060277          INTDS

00165'040575          STA     0,SAV0  ; SAVE ACCUMULATORS
00166'044575          STA     1,SAV1
00167'050575          STA     2,SAV2
00170'054575          STA     3,SAV3
00171'176660          CLAR    3,3     ; CARRY -) BIT 0
00172'020672          LDA     0,INTFL
00173'030672          LDA     2,BPN1
00174'142000          ADC     2,0
00175'116400          SUB     0,3     ; 1 IF ION, 0 IF IOF
00176'054570          STA     3,SAVCI ; BIT 0 = CARRY, BIT 15 = INT
00177'050504          STA     2,BPN   ; 0 = BREAK ENTRY, 1 = START
00200'176460          CLA     3,3
00201'054663          STA     3,INTFL
00202'054663          STA     3,BPN1

       000001     .IFN T
00203'010731          ISZ     JFLAG   ; TEST IF SUBR. RETURN
00204'000402          JMP     .+2
00205'002726          JMP     @AGO    ; TASK MODE
                  .ENDC

00206'026651          LDA     1,@ABRA ; CURRENT ADDRESS
00207'020647          LDA     0,BRINS ; AND INSTRUCTION
00210'044725          STA     1,PRADR ; SAVE FOR PROCEED

00211'014562          DSZ     PRCNT   ; PROCEED COUNTER
00212'151014          IFN     2,2     ; TEST IF START ENTRY
00213'000567          JMP     BREAK   ; START OR PRCNT = 0
```

```

                        ; EMULATED EXECUTION OF THE BREAK INSTRUCTION.
                        ; PROCEED ADDRESS IN AC1, PROCEED INSTRUCTION IN AC0

   00214'010721 EMI:     ISZ     PRADR     ; NEXT LOCATION
   00215'034646          LDA     3,C60K    ; 60000
   00216'030637          LDA     2,C1400   ; 1400
   00217'116032          IFLT    0,3       ; SKIPS IF IO OR ALC

   00220'113705          ANDS    0,2,SNR   ; MEM REF SKIPS IF X () 0
   00221'000425          JMP     NOTX

   00222'034637          LDA     3,ASAV    ; ADDRESS OF REG SAVE BLOCK
   00223'157000          ADD     2,3
   00224'151234          MOVZR#  2,2,SZR   ; SKIPS IF X = 1
   00225'025400          LDA     1,0,3     ; INDEX VALUE
   00226'030634          LDA     2,C377    ; 377
   00227'155620          INCZR   2,3       ; 200
   00230'113400          AND     0,2       ; DISPLACEMENT
   00231'157524          ANDZL   2,3,SZR   ; SIGN -) BIT 7
   00232'172400          SUB     3,2       ; EXTEND SIGN
   00233'133100          ADDL    1,2       ; (X + D) * 2
   00234'024552          LDA     1,C176K   ; 176000
   00235'134400          NEG     1,3       ; 2000
   00236'107420          ANDZ    0,1       ; STRIP INDEX AND DISPL
   00237'137414          AND#    1,3,SZR   ; TAKE OUT @ BIT IF
   00240'166420          SUBZ    3,1       ; INSTRUCTION IS INDIRECT
   00241'151201          MOVR    2,2,SKP   ; AND INSERT @ BIT IN ADDRESS.
   00242'054673 R1:      STA     3,PRADR
   00243'050666          STA     2,RELAD   ; ADDRESS
   00244'020435          LDA     0,CDEX    ; DISPLACEMENT FOR EXECUTION
   00245'123000          ADD     1,0       ; MODIFIED INSTRUCTION

   00246'024664 NOTX:    LDA     1,C4K     ; 4000
   00247'134520          NEGZL   1,3       ; 170000
   00250'117424          ANDZ    0,3,SZR
   00251'000405          JMP     NOTJ      ; NOT JMP/JSR

   00252'034663          LDA     3,PRADR   ; NEXT WORD AFTER JSR
   00253'107404          AND     0,1,SZR   ; SKIP IF JMP
   00254'054511          STA     3,SAV3    ; JSR
   00255'122420          SUBZ    1,0       ; TURN JSR TO JMP. C (- 1

   00256'010657 NOTJ:    ISZ     PRADR     ; PRE-INCREMENT FOR SKIPS
   00257'176200          ADCR    3,3       ; JFLAG (- 77777 IF NOT JMP/JSR
   00260'054654          STA     3,JFLAG   ; JFLAG (- 177777 IF JMP/JSR
   00261'040414          STA     0,PRINS   ; STORE FOR EXECUTION
```

```
0008  DB1.5

00262'020500         LDA     0,SAV0   ; RESTORE AC'S
00263'024500         LDA     1,SAV1
00264'030500         LDA     2,SAV2
00265'034501         LDA     3,SAVCI  ; CARRY AND INTERRUPT
00266'175140         MOVOL   3,3
00267'054414         STA     3,BPN    ; TEMPORARY INTERRUPT FLAG
00270'034475         LDA     3,SAV3
00271'010643         ISZ     JFLAG
00272'000403         JMP     PRINS    ; NOT JMP/JSR

00273'014410         DSZ     BPN      ; JMP/JSR
00274'060177         INTEN
00275'000000 PRINS:  0                ; EXECUTE INSTRUCTION
00276'014637         DSZ     PRADR    ; DID NOT SKIP
00277'014404         DSZ     BPN
00300'060177         INTEN
00301'002634 CDEX:   JMP     @PRADR   ; 'PRADR' 4 WORDS AFTER 'RELAD'
```

```
0009  DB1.5

                    ; CONSTANTS AND FLAGS

   00302'177720 M60:     -60
   00303'000000 BPN:     0
   00304'000715'ACRLF:   CRLF

                    ; COMMAND TABLE

   00305'000510'AT:      LFS
   00306'000511'         CRS
   00307'000565'         PLS
   00310'000663'         CS
   00311'000564'         MNS
   00312'000574'         DOTS
   00313'000550'         SLSHS
   00314'000555'         EQLS
   00315'000526'         AS
   00316'000604'         BS
   00317'000070'         ES
   00320'000107'         FS
   00321'000136'         GS
   00322'000003'         PS
   00323'000141'         RS
   00324'000623'         SS
   00325'000521'         ROS

   00326'000012 CT:      12
   00327'000015          15
   00330'000053          "+
   00331'000054          ",
   00332'000055          "-
   00333'000056          ".
   00334'000057 C57:     "/
   00335'000075          "=
   00336'000101          "A
   00337'000102          "B
   00340'000105          "E
   00341'000106          "F
   00342'000107          "G
   00343'000120          "P
   00344'000122          "R
   00345'000123          "S
   00346'000177 C177:    177
```

```
0010  DB1.5


                ; SCAN THE COMMAND TABLE.
                ; ENTER WITH CARRY = 0

00347'031624 SCAN:   LDA     2,CT-J,3
00350'175402         INC     3,3,SZC
00351'000504         JMP     GETC+1  ; IGNORE CHARACTER
00352'112424         SUBZ    0,2,SZR
00353'000774         JMP     SCAN    ; CARRY = 1 IF AC2 ) AC0

00354'030567         LDA     2,TOTAL
00355'010565         ISZ     SFLAG   ; ADD OR SUBTRACT
00356'133001         ADD     1,2,SKP
00357'132400         SUB     1,2     ; AC2 IS SUBTOTAL
00360'126440         SUBO    1,1     ; AC1 = 0, CARRY = 0
00361'007602         JSR     @AT-J-1,3

                ; AC3 CONTAINS POINTER TO AC SAVE BLOCK

00362'000000 SAV0:   0       ; 0A  OR  A
00363'000000 SAV1:   0       ; 1A
00364'000000 SAV2:   0       ; 2A
00365'000000 SAV3:   0       ; 3A
00366'000000 SAVCI:  0       ; 4A
00367'000000 TTOFL:  0       ; 5A
00370'000156'LOC:    DB1.5   ; 6A
00371'000000 OFFST:  0       ; 7A
00372'000000 BOFL:   0       ; 10A
00373'000001 PRCNT:  1       ; 11A
00374'000000 MSK:    0       ; 12A
00375'000000 WRD:    0       ; 13A

                ; CONSTANTS AND FLAGS

00376'000070 C70:    70
00377'000726'APUT:   PUT
00400'000710'AOCT5:  OCT5

      000001    .IFN T
00401'000756'ATCBW:  TCBW
                .ENDC
```

```
0011  DB1.5
                      ; A BREAK HAS OCCURRED.
00402'010771 BREAK:   ISZ     PRCNT    ; RESTORE TO +1
00403'063511          SKPBZ   TTO      ; AC2 IS NOT -1
00404'000777          JMP     .-1
00405'063711          SKPDZ   TTO      ; TEST IF TTO ACTIVE
00406'176000 C176K:   ADC     3,3      ; YES - SET TO -1
00407'054760          STA     3,TTOFL
      000001     .IFN T
00410'034771          LDA     3,ATCBW ; TCB POINTER
00411'057417          STA     3,@17,3 ; ACTIVATE DEBUG TASK
00412'023755          LDA     0,@ATTOE-TCBW,3 ; TTOE1 ADDRESS
00413'162405          SUB     3,0,SNR ; IS DEB TASK WAITING ON TTO
00414'043755          STA     0,@ATTOE-TCBW,3 ; STOP IT WAITING
00415'102460          CLA     0,0
00416'042443          STA     0,@ATTIE; CLEAR TTI ECW
00417'041403          STA     0,3,3    ; CLEAR AC0 IN TCBW FOR AFLAG
00420'021421          LDA     0,ARENT-TCBW,3  ; RE-ENTRY ADDRESS
00421'041404          STA     0,4,3    ; OVERWRITE PC IN TCBW
00422'054540          STA     3,TFLAG ; STAND ALONE MODE
                 .ENDC
00423'151015          IFZ     2,2      ; TEST IF BREAK OR START
00424'006754          JSR     @AOCT5   ; BREAK - TYPE ADDRESS
00425'006657 GO:      JSR     @ACRLF
      000001     .IFN T
00426'010534          ISZ     TFLAG    ; IS IT TASK MODE
00427'000420          JMP     G2
                      ; TASK RE-ENTERS HERE. STARTS WITH L41 = 0
00430'060277 REENT:   INTDS
00431'010041          ISZ     41       ; -1 -) 0 = OUTSIDE
00432'000403          JMP     .+3      ; DEBUG WAS INTERRUPTED OUTSIDE
00433'177777          RAISE
00434'177777          SEMDT   ; SYNCHRONISE OUTPUT
00435'177777          .WAIT   ; WAIT FOR A KEYSTROKE
00436'177777          TTIEC   ; DON'T CLEAR YET
00437'176000          ADC     3,3
00440'054522          STA     3,TFLAG ; TASK MODE
00441'054642          STA     3,BPN    ; NO 'P' FROM TASK
00442'054725          STA     3,TTOFL ; 'R' FROM TASK LEAVES TTO ON
00443'060277          INTDS
00444'054041          STA     3,41     ; -1 = INSIDE DEBUG TASK
00445'177777          LOWER
00446'000434'         SEMDT
                 .ENDC
```

```
0012  DB1.5

00447'040475 G2:     STA     0,AFLAG ; SET OR CLEAR AFLAG
00450'126460         CLA     1,1     ; CLEAR # REGISTER
00451'044474         STA     1,BFLAG ; CLEAR BREAKFLAG
00452'044470         STA     1,SFLAG ; SIGN FLAG (NOT -1)
00453'044470         STA     1,TOTAL ; SUBTOTAL

00454'010471 GETC:   ISZ     BFLAG   ; +1 FIRST TIME THROUGH

      000001     .IFN T
00455'010505         ISZ     TFLAG
00456'000412         JMP     .+12

00457'014503         DSZ     TFLAG
00460'000435'        .WAIT
00461'000436'ATTIE:  TTIEC   ; WAIT AGAIN
00462'176460         CLA     3,3
00463'056776         STA     3,@.-2  ; CLEAR THIS TIME
00464'060477         READS   0       ; IGNORE KEYSTROKE IN TASK MODE
00465'101113         IFZP    0,0     ; IF SWITCH 0 = 0
00466'000742         JMP     REENT
00467'000403         JMP     .+3
                 .ENDC

00470'063610         SKPDN   TTI
00471'000777         JMP     .-1
00472'060610         DIAC    0,TTI   ; NO ECHO PRINT HERE
00473'034653         LDA     3,C177  ; 177
00474'163400         AND     3,0
00475'034701         LDA     3,C70
00476'116032         IFLT    0,3
00477'034603         LDA     3,M60
00500'117023         ADDZ    0,3,SNC
00501'004646         JSR     SCAN    ; SCAN FOR BREAK CHAR.
             J:                      ; C = 0
00502'125120         MOVZL   1,1
00503'125120         MOVZL   1,1
00504'125120         MOVZL   1,1
00505'167000         ADD     3,1
00506'006671         JSR     @APUT   ; ECHO
00507'000745         JMP     GETC    ; # MODULO 216
```

```
0013  DB1.5

                 ; OPEN AND MODIFY A WORD

00510'101040 LFS:    MOVO    0,0     ; SET CARRY

00511'034452 CRS:    LDA     3,ADRS  ; CARRY = 0 FOR CRS
00512'024432         LDA     1,AFLAG ; +VE IF CLOSED, -2 IF OPEN
00513'020432         LDA     0,BFLAG ; 1 IF NO #, )1 IF #
00514'107032         ADDZ#   0,1,SZC
00515'051400         STA     2,0,3   ; MODIFY MEMORY
00516'165403         INC     3,1,SNC ; TEST IF LINE FEED ENTRY
00517'000706         JMP     GO      ; NO. ECHO CR AT GO

00520'044443         STA     1,ADRS  ; OPEN NEXT LOCATION
00521'024442 ROS:    LDA     1,ADRS  ; RUB OUT RE-OPENS LAST
                                     ; OPENED WORD. (NO ECHO)
00522'004566         JSR     OCT5    ; 5 DIGIT OCTAL ADDRESS
00523'152440         SUBO    2,2     ; C (- 0, AC2 (- 0
00524'020610         LDA     0,C57   ; / PRINTED AT SLSH2

00525'034436 SLSH1:  LDA     3,ADRS

00526'173002 AS:     ADD     3,2,SZC ; ACCUMULATOR ENTRY
00527'031400         LDA     2,0,3   ; OPEN ON CONTENTS OF LAST WORD
00530'006647 SLSH2:  JSR     @APUT   ; ECHO PRINT
00531'050432         STA     2,ADRS  ; OPEN A WORD
00532'020571         LDA     0,BRADR ; CURRENT BREAK ADDRESS
00533'112415         IFEQ    0,2
00534'000423         JMP     BRM     ; DON'T MODIFY BREAK POINT

00535'025000         LDA     1,0,2   ; CONTENTS
00536'004531         JSR     B.OCT   ; TYPE OCTAL OR BIN
00537'004566         JSR     PSPCE   ; A SPACE
00540'102120         ADCZL   0,0     ; AFLAG (- -2
00541'000706 G5:     JMP     G2

00542'000000 SFLAG:  0
00543'000000 TOTAL:  0
00544'000000 AFLAG:  0
00545'000000 BFLAG:  0
00546'000000 SADR:   0
00547'000000 SCNT:   0
```

```
0014  DB1.5
              ; MISCELLANEOUS ROUTINES
00550'014775 SLSHS:  DSZ     BFLAG    ; / ENTRY
00551'000757         JMP     SLSH2    ; OPEN WORD JUST TYPED
00552'034772         LDA     3,AFLAG
00553'175100         MOVL    3,3      ; C = 1 IF AFLAG IS -VE
00554'000751         JMP     SLSH1    ; AC2 = 0
00555'004551 EQLS:   JSR     PUT      ; ECHO
00556'145001         MOV     2,1,SKP  ; PRINT VALUE
00557'026545 BRM:    LDA     1,@ABRI  ; PRINT BREAK CONTENTS
00560'004507         JSR     B.OCT
00561'000644 G4:     JMP     GO
       000001    .IFN T
00562'000000 TFLAG:  0        ; MAY BE INSPECTED AS 200A
                 .ENDC
00563'000000 ADRS:   0
              ; ARITHMETIC ROUTINES
              ; AC2 IS NEW TOTAL, AC1 = 0
00564'176000 MNS:    ADC     3,3      ; -1
00565'054755 PLS:    STA     3,SFLAG  ; NOT -1
00566'034757         LDA     3,BFLAG  ; TEST FOR PREVIOUS #
00567'175235         MOVZR#  3,3,SNR  ; IF SIGN IS 1ST CHAR, LOAD
00570'030601         LDA     2,OFFST  ; OFFSET REGISTER
00571'004535 DOT1:   JSR     PUT      ; ECHO
00572'050751         STA     2,TOTAL
00573'000661         JMP     GETC     ; AC1 IS CLEAR
00574'034767 DOTS:   LDA     3,ADRS   ; . = ADDRESS OF LAST WORD
00575'175100         MOVL    3,3
00576'175220         MOVZR   3,3
00577'014743         DSZ     SFLAG    ; +VE = NOT -1
00600'010742         ISZ     SFLAG
00601'173001         ADD     3,2,SKP  ; +
00602'172400         SUB     3,2      ; -
00603'000766         JMP     DOT1
```

```
0015  DB1.5

                      ; SET UP A BREAKPOINT

   00604'004522 BS:      JSR     PUT
   00605'151100          MOVL    2,2
   00606'014737          DSZ     BFLAG
   00607'151221          MOVZR   2,2,SKP
   00610'030514          LDA     2,ABRI  ; ADDRESS OF BREAK INSTRUCTION
   00611'022513          LDA     0,@ABRI
   00612'042511          STA     0,@BRADR
   00613'021000          LDA     0,0,2
   00614'042510          STA     0,@ABRI
   00615'050506          STA     2,BRADR
   00616'020533          LDA     0,BREN  ; SET UP BREAK LINKAGE NOW
   00617'040017          STA     0,LINK  ; FOR TASK EXECUTION
   00620'020532          LDA     0,BRKI  ; JMP @LINK
   00621'041000          STA     0,0,2   ; AT NEW BREAK POINT
   00622'000737          JMP     G4
```

```
0016 DB1.5
                ; SEARCH MEMORY
00623'014721 SS:     DSZ     AFLAG    ; TEST IF XXX,
00624'044722         STA     1,SADR   ; NO - START OF SEARCH = 0
00625'126220         ADCZR   1,1      ; 77777
00626'014717         DSZ     BFLAG    ; ANY 2ND ARGUMENT?
00627'147400         AND     2,1      ; YES - MASK BIT 0
                                      ; NO - SEARCH ENDS AT 77777
00630'030716         LDA     2,SADR
00631'132022         ADCZ    1,2,SZC  ; SEARCH LENGTH
00632'000727         JMP     G4       ; -VE - NO SEARCH
00633'050714         STA     2,SCNT
00634'031413         LDA     2,WRD-SAV0,3
00635'004471         JSR     PUT      ; ECHO
00636'026517 SLP:    LDA     1,@AMSK
00637'133400         AND     1,2      ; MASK SEARCH WORD
00640'022706         LDA     0,@SADR
00641'123400         AND     1,0      ; MASK WORD
00642'112414         IFNE    0,2      ; IFEQ FOR OPPOSITE SEARCH
00643'000414         JMP     SNEXT
00644'024702         LDA     1,SADR
00645'004443         JSR     OCT5     ; PRINT CR AND ADDRESS
00646'026700         LDA     1,@SADR
00647'004420         JSR     B.OCT    ; PRINT SPACE AND VALUE
      000001    .IFN T
00650'010712         ISZ     TFLAG
00651'000404         JMP     .+4
00652'014710         DSZ     TFLAG
00653'036606         LDA     3,@ATTIE
00654'175113         IFZP    3,3      ; TEST FOR KEYSTROKE
                .ENDC
00655'063710         SKPDZ   TTI
00656'000703         JMP     G4       ; KEY HAS BEEN STRUCK
00657'010667 SNEXT:  ISZ     SADR
00660'010667         ISZ     SCNT
00661'000755         JMP     SLP
00662'000677         JMP     G4       ; SEARCH COMPLETE
                ; ENTER FROM ",".
00663'050663 CS:     STA     2,SADR   ; LOWER LIMIT
00664'004442         JSR     PUT      ; ECHO
00665'102520         SUBZL   0,0      ; +1 -) AFLAG
00666'000653         JMP     G5
```

0017 DB1.5

```
                ; BINARY AND OCTAL PRINTOUT (AC2 UNCHANGED)

00667'022461 B.OCT:  LDA     0,@ABOFL; BINARY OR OCTAL
00670'101024         MOVZ    0,0,SZR ; 0 = OCT, 1 = BIN
00671'177240         ADDOR   3,3     ; EITHER WAY C = 0
00672'054462         STA     3,SAVOC
00673'004432         JSR     PSPCE   ; PRINT A SPACE

00674'020451         LDA     0,C1.3  ; AC0 (- 100030
00675'101060 O1:     MOVC    0,0     ; C (- C'
00676'125105         MOVL    1,1,SNR
00677'001400         JMP     0,3     ; RETURN
                                     ; CARRY IS SET
00700'101103         MOVL    0,0,SNC
00701'000775         JMP     O1+1

00702'004424         JSR     PUT     ; PRINT CHARACTER
00703'034451         LDA     3,SAVOC

00704'020442 O5:     LDA     0,C2.6
00705'175112         IFM     3,3
00706'000766         JMP     O1-1    ; BINARY
00707'000766         JMP     O1      ; OCTAL

00710'054444 OCT5:   STA     3,SAVOC ; 5 DIGIT OCTAL
00711'004404         JSR     CRLF    ; PRINT CR LF
00712'125143         MOVOL   1,1,SNC ; IGNORE BIT 0
00713'101040 SPACE:  MOVO    0,0     ; C (- 1
00714'000770         JMP     O5

                ; PRINT CR AND LF

00715'020432 CRLF:   LDA     0,C5015
00716'054435         STA     3,SAVCR
00717'004407         JSR     PUT
00720'101300         MOVS    0,0
00721'004405         JSR     PUT
00722'002431         JMP     @SAVCR
```

```
0018  DB1.5

00723'000056'BRADR:  BRINS
00724'000056'ABRI:   BRINS

             ; PRINT A SPACE

00725'020766 PSPCE:  LDA     0,SPACE

             ; PRINT A CHARACTER IN AC0

             PUT:
      000001    .IFN T
00726'010634         ISZ     TFLAG
00727'000411         JMP     P1
00730'054040         STA     3,40    ; L40 CAN ONLY BE USED IN TASK
00731'014631         DSZ     TFLAG
00732'000460'        .WAIT
00733'177777 ATTOE:  TTOE1
00734'176460         CLA     3,3
00735'056776         STA     3,@.-2
00736'061111         DOAS    0,TTO
00737'002040         JMP     @40

00740'054622 P1:     STA     3,TFLAG ; NOT -1
                .ENDC
00741'061111         DOAS    0,TTO
00742'063511         SKPBZ   TTO
00743'000777         JMP     .-1
00744'001400         JMP     0,3

00745'100030 C1.3:   100030
00746'020006 C2.6:   20006
00747'005015 C5015:  5015

00750'000372'ABOFL:  BOFL
00751'000162'BREN:   BRK
00752'002017 BRKI:   JMP     @LINK
00753'000000 SAVCR:  0
00754'000000 SAVOC:  0
00755'000374'AMSK:   MSK

      000001    .IFN T

      000017 TCBW:   .BLK    17
00775'000774'        .-1     ; PREVENT STORING RUBBISH IN TASK
                             ; LIST WHEN FIRST STARTED.

00776'004000         1B4
00777'000430'ARENT:  REENT   ; TASK START

                .ENDC

      000156'.END    DB1.5
```

0019 DB1.5

ABOFL	000750'	17/04	18/35						
ABRA	000057'	3/34	5/18	6/41					
ABRI	000724'	14/14	15/08	15/09	15/12	18/03			
ACRLF	000304'	5/45	9/06	11/28					
ADRS	000563'	13/06	13/14	13/15	13/21	13/26	14/21	14/36	
AFLAG	000544'	2/34	12/02	13/07	13/39	14/07	16/04		
AGO	000133'	2/46	4/20	5/29	6/38				
AMSK	000755'	16/18	18/40						
AOCT5	000400'	10/37	11/26						
APUT	000377'	2/41	4/09	4/25	5/14	5/44	10/36	12/40	13/25
ARENT	000777'	11/20	18/49						
AS	000526'	9/18	13/23						
ASAV	000061'	3/36	7/13						
AT	000305'	9/10	10/16						
ATCBW	000401'	10/40	11/12						
ATFLG	000066'	3/42	5/37	5/55					
ATTIE	000461'	11/18	12/16	16/34					
ATTOE	000733'	11/14	11/16	18/18					
ATTOF	000060'	3/35	5/15	5/46					
BFLAG	000545'	4/05	5/41	12/04	12/08	13/08	13/40	14/04	14/28
		15/06	16/07						
BLL	000012'	2/43	3/21						
BOFL	000372'	10/28	18/35						
BPN	000303'	5/07	6/30	8/07	8/12	8/16	9/05	11/47	
BPN1	000065'	3/40	6/14	6/26	6/33				
BRADR	000723'	3/34	13/27	15/10	15/13	18/02			
BREAK	000402'	6/47	11/04						
BREN	000751'	15/14	18/36						
BRINS	000056'	3/33	5/17	6/42	18/02	18/03			
BRK	000162'	6/16	18/36						
BRKI	000752'	15/16	18/37						
BRM	000557'	13/29	14/14						
BS	000604'	9/19	15/04						
B.OCT	000667'	13/32	14/15	16/28	17/04				
C1400	000055'	3/31	7/07						
C176K	000406'	7/23	11/08						
C177	000346'	9/44	12/28						
C1.3	000745'	17/10	18/31						
C20	000002'	2/32	2/48						
C2.6	000746'	17/21	18/32						
C377	000062'	3/37	7/17						
C4K	000132'	5/28	5/40	7/34					
C5015	000747'	17/34	18/33						
C57	000334'	9/34	13/19						
C60K	000063'	3/38	7/06						
C70	000376'	10/35	12/30						
CDEX	000301'	7/31	8/18						
CRLF	000715'	9/06	17/27	17/34					
CRS	000511'	9/11	13/06						
CS	000663'	9/13	16/47						
CSL	000030'	3/04	3/08						
CT	000326'	9/28	10/05						
DB1.5	000156'	6/11	10/26	18/53					
DOT1	000571'	14/32	14/43						
DOTS	000574'	9/15	14/36						
EMI	000214'	5/22	7/05						
EQLS	000555'	9/17	14/11						
ES	000070'	4/05	9/20						
FLP	000102'	4/16	4/19	4/26					

0020 DB1.5

FS	000107'	4/24	9/21						
GO	000425'	5/09	5/29	11/28	13/12	14/16			
G2	000447'	11/32	12/02	13/35					
G4	000561'	14/16	15/18	16/12	16/38	16/43			
G5	000541'	13/35	16/50						
GETC	000454'	10/07	12/08	12/41	14/34				
GFLAG	000067'	3/43	5/38	5/54					
GS	000136'	5/35	9/22						
INTFL	000064'	3/39	6/12	6/17	6/25	6/32			
J	000502'	10/05	10/16	12/35					
JFLAG	000134'	2/39	2/44	5/30	5/56	6/36	7/46	8/09	
LFS	000510'	9/10	13/04						
LINK	000017	3/33	5/26	15/15	18/37				
LOC	000370'	5/43	10/26						
LOWER	000445'X	11/51							
M60	000302'	4/08	4/24	9/04	12/32				
MNS	000564'	9/14	14/26						
MSK	000374'	10/30	18/40						
NOTJ	000256'	7/37	7/44						
NOTX	000246'	7/11	7/34						
O1	000675'	17/11	17/16	17/23	17/24				
O5	000704'	17/21	17/30						
OCT5	000710'	10/37	13/17	16/26	17/26				
OFFST	000371'	10/27	14/30						
P1	000740'	18/14	18/24						
PLP	000040'	3/13	3/17						
PLS	000565'	9/12	14/27						
PRADR	000135'	5/19	5/21	5/31	6/43	7/05	7/29	7/39	7/44
		8/15	8/18						
PRCNT	000373'	5/13	6/45	10/29	11/04				
PRINS	000275'	7/47	8/10	8/14					
PROC	000112'	2/35	5/07						
PS	000003'	2/34	9/23						
PSPCE	000725'	13/33	17/08	18/07					
PTPS	000000'	2/26							
PUT	000726'	10/36	14/11	14/32	15/04	16/16	16/48	17/18	17/36
		17/38	18/11						
PW	000050'	2/53	2/55	3/09	3/14	3/20	3/26	3/30	4/11
		4/13	4/17						
R1	000242'	5/48	7/29						
RAISE	000433'X	11/39							
REENT	000430'	11/36	12/21	18/49					
RELAD	000131'	2/51	3/10	5/27	5/53	5/57	7/30		
ROS	000521'	9/26	13/15						
RS	000141'	5/41	9/24						
RTI	000001$X	2/27							
SADR	000546'	2/40	13/41	16/05	16/10	16/20	16/25	16/27	16/40
		16/47							
SAVO	000362'	2/34	2/40	2/41	3/36	4/05	4/08	4/09	4/24
		4/25	5/09	5/13	5/14	5/41	5/43	5/44	6/20
		8/02	10/20	16/15					
SAV1	000363'	6/21	8/03	10/21					
SAV2	000364'	6/22	8/04	10/22					
SAV3	000365'	6/23	7/41	8/08	10/23				
SAVCI	000366'	6/29	8/05	10/24					
SAVCR	000753'	17/35	17/39	18/38					
SAVOC	000754'	17/07	17/19	17/26	18/39				
SCAN	000347'	10/05	10/09	12/34					
SCNT	000547'	13/42	16/14	16/41					

0021 DB1.5

SEMDT	000446'X	11/40	11/52						
SFLAG	000542'	10/12	12/05	13/37	14/27	14/39	14/40		
SLP	000636'	16/18	16/42						
SLSH1	000525'	13/21	14/09						
SLSH2	000530'	13/25	14/05						
SLSHS	000550'	9/16	14/04						
SNEXT	000657'	16/23	16/40						
SPACE	000713'	17/29	18/07						
SS	000623'	9/25	16/04						
TCBW	000756'	10/40	11/14	11/16	11/20	18/44			
TFLAG	000562'	3/42	11/22	11/31	11/46	12/11	12/14	14/19	16/31
		16/33	18/13	18/16	18/24				
TOTAL	000543'	10/11	12/06	13/38	14/33				
TTIEC	000461'X	11/43	12/16						
TTOE1	000733'X	18/18							
TTOFL	000367'	3/35	10/25	11/09	11/48				
WRD	000375'	10/31	16/15						
.WAIT	000732'X	11/42	12/15	18/17					