A facility for real-time program development

D. G. Bennett* and R. A. Davenport†

A facility is described which, within a multi-access system, allows real-time programs to be developed. The manner of operation is such that to the user it appears that his program has complete control of interrupts. In actual fact, they are monitored by the system's executive to prevent his program interfering with those of other users.

(Received August 1971)

As part of a multi-access system, a facility was developed to assist in the preparation of real-time programs for a specialist application. The multi-access system (GPO, 1970) is located within the Electronic Switching Group GPO Research Station, Dollis Hill. This group is presently engaged on the development of a computer-controlled telephone exchange.

As can be imagined, a considerable software effort is involved even in the initial stage, which consists initially of a four-line exchange controlled by a small computer via a data switch. Plans are in hand to test the system simulating much larger traffic.

This software comprises the exhange system executive, a large number of operating routines, of the order of 100, and a complex test program used in the development of the data switch and the other pieces of hardware which make up the exchange. Powerful debug facilities were available and it was desired to extend their use to real-time programs.

Because of the programming effort required to implement the system for the exchange and also because the system was to be implemented on a small computer, it was necessary to produce initially an operating system that would provide both multi-access capabilities and comprehensive debug facilities. A further advantage was that it would permit hardware and software development to proceed simultaneously.

The first provision is an obvious one since all computing is done in a 'hands-on' situation, i.e. each programmer is allotted time, during which he may assemble, edit or debug his program. Particularly during the last stage, the computer will be idle for appreciable periods during which the programmer cogitates. Multi-access, therefore, is an obvious method of increasing the efficiency of use of the machine, The comprehensive debug facility ensures that the programmer uses his time as efficiently as possible.

The small machine concerned was a Honeywell DDP 516 with 32K words of storage. A high speed punch and reader are connected together with two Teletype ASR 33 and one KSR 35. Optional features which are incorporated in the system are memory lock-out and standard and priority interrupt lines. The high speed reader and punch are connected to the standard interrupt line, while the teletype, real-time clocks, data switch and other hardware links are connected to the priority lines.

In the Honeywell interrupt system, when an interrupt occurs, it causes the contents of the program counter, which contains the address of the current instruction to be executed, to be automatically changed, thereby changing the sequence of instruction execution. Interrupts have unique memory locations dedicated to them, whose contents are interpreted as an indirect address. The action of an interrupt causes the program to branch to the location whose address is stored in the dedicated location. These dedicated locations are held in the base sector. The standard interrupt has one dedicated location for all lines while priority interrupt have one location per line. The obvious advantage of the priority interrupt system is that it eliminates the need for a service routine to determine which one of the available interrupt lines caused the interrupt.

For an interrupt to occur, two requirements must be met. These are the interrupt mask flip-flop must be set and the CPU must be in the permit interrupt condition.

The first requirement is met by means of the SMK instruction, which transfers the contents of the accumulator to the mask flip-flop of various devices. Each bit position of the accumulator controls a unique device. A one sets the mask and a zero resets it. The second requirement is achieved by means of the instruction ENB. This permits interrupts to occur on any line for which the mask is set. This does not take effect until one instruction after the ENB.

The instruction ENB sets the machine status to allow interrupts to occur while the instruction INH prohibits interrupts from occurring.

When an interrupt does occur, no further interrupts are permitted until after the execution of an ENB. That is, an interrupt causes an automatic INH instruction.

The memory lock-out option provides base sector relocation and also equips the CPU with a mode of operation called 'restricted mode'. Base sector relocation allows each user to have assigned to him a sector which to him appears a base sector, i.e. addresses less than octal '1000. Restricted mode has the properties that:

1. Instructions which normally write into memory locations can be locked out of protected sectors.
2. Certain instructions are considered illegal and cannot be performed.

If either of these occurs an interrupt is generated, the memory lock-out violation. Because of this feature, all users' programs can be run in restricted mode. Any instruction which attempts to write into a memory location triggers the memory lock-out violation interrupt. When this occurs, the executive determines whether or not it is legal for the user to alter the contents of that location. The effect is, then, that the user can corrupt only that area of store that has been assigned to him and not the areas occupied either by other users or the executive.

Interrupt facility

The multi-access system allows the user to assemble, load, edit and debug his program via the teletype and the high speed reader. It also allows him to dump the contents of his core via paper tape which can be reloaded into the machine, All operations and running of programs by the user are carried out in restricted mode. Since it was decided that as much real-time

*GPO Research Station, Dollis Hill
†Scientific Control Systems Limited

work as possible should be run on the system, an essential requirement appeared to be the monitoring of interrupts. This would then allow real-time programs to be tested in isolation without affecting the running of other users' programs. In order to implement this requirement the system therefore had to detect and analyse interrupts occurring during the running of the real-time program. The interrupts which were monitored were priority interrupts only. If the user wishes to have his program interrupted, it should be done in a way to approximate as closely as possible to the situation where his program is in sole command of the machine. As has been stated, on the multi-level priority interrupt system, the need for determining which interrupt line has interrupted is eliminated. The lines may be selectively enabled or inhibited by means of masking. The instruction INH suppresses all interrupts whether the mask flip-flop for a particular line is set or not.

The other feature of restricted mode besides memory protection is that certain instructions are privileged. If an attempt is made to execute such an instruction a memory lock-out violation occurs. It would have been possible to allow the executive to find the cause of the interrupt if *both* INH and ENB caused such a violation. However, since INH produces the violation while ENB does not, it was necessary to incorporate a procedure within the executive which searches for these two instructions in the user's program before attempting to run the program. The search is made on the assembled text between limits stated by the user. Each location which has a bit pattern corresponding to either INH or ENB is typed out and is altered to a transfer to the executive if appropriate by the user. The reason for this is that the locations which have these bit patterns may in fact contain data. If the user does not replace all the INH instructions by transfers, the normal interrupts will not be inhibited because of the memory lock-out facility. During the running of the user's program, the INH instruction will be detected and an error message printed out. This, then, provides further protection of the system from misuse.

There are five routines involved. These are:

1. SMK
2. ENBI
3. INTH
4. INHI
5. RTC

1. *SMK*

The instruction SMK sets the priority interrupt lines which the user wishes to be allowed to interrupt, i.e. it is a masking instruction. This instruction is detected in the user's program during running by memory lockout violation. The executive determines what lines the user required set by examining the contents of his accumulator. This is stored, and if the real-time clock (10 millisecond) interrupt is required a toggle is set. Return is then made to the user's program.

2. *ENBI*

Since ENB will not be detected by memory lockout violation, a search is made prior to the running of the program for such instructions. If any are found, they are replaced by transfers to the ENBI routine. When the program is run and such a transfer is made, the next instruction after the ENB is executed interpretively and the interrupt lines requested by the user are enabled and then inhibited one instruction later. The reason for this action is to simulate the actual hardware, since if an interrupt is pending it will take effect one instruction after the ENB. The interrupts are inhibited immediately to prevent further interrupts occurring in executive time. As far as possible, an approximation, therefore, is made to the true time profile of the user's program. After inhibiting interrupts, a check is made for whether or not an interrupt has been logged by the INTH routine. This indicates that the interrupt occurred when the lines had been enabled within this routine, ENBI. Therefore a return is made to the address given in the user's interrupt link, which is located in his relocated base sector. Otherwise, a toggle, called the enable toggle, is set and a normal return is made to the user's program, but now with the user's interrupt lines masked in and enabled. The flow chart of this routine appears in **Fig. 1.**

3. *INTH*

When any priority interrupt, other than the normal 10 millisecond clock or the teletypes, occurs it is forced to this routine. The interrupt which has occurred is first identified and logged. If the enable toggle is not set it indicates that the interrupt must have occurred in the ENBI routine so a transfer to that routine is made. A test is also made for whether or not the interrupt occurred during executive time. If it did a return is made to the executive which, when it has completed its tasks, returns control to the INTH routine. This enable toggle is then cleared and a return is made to the address given in the user's interrupt link, held in his base sector. The flow chart of this routine appears in **Fig. 2.**

4. *INHI*

In the same manner as for ENB instructions a search is made before the running of the program for INH instructions and if any are found, they are replaced by transfers to the INHI routine. When the program is run and such a transfer is made, the enable toggle is cleared and a mask is set for the priority interrupt lines which allows only the real-time clock and the teletypes to interrupt. Thus interrupts are not actually inhibited but are masked out. A record is kept of those lines so treated in order that they be masked in when the interrupts are enabled.

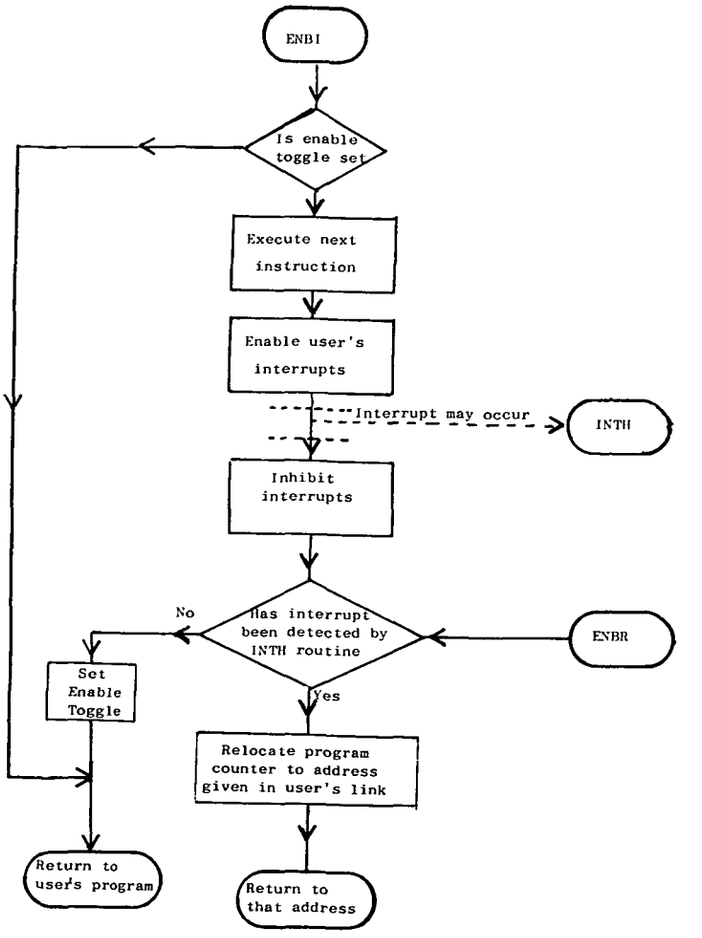

Fig. 1. ENBI routine

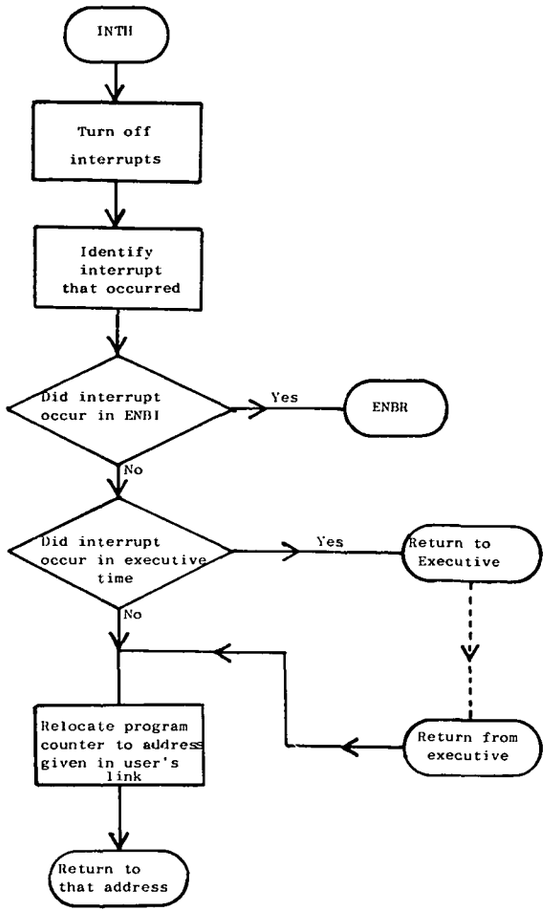

Fig. 2. INTH routine

5. RTC

All interrupts requested by the user, save one, go through the routine INTH. The one exception is the 10 millisecond clock. The reason for this being that it is required by the executive and therefore has to be handled in a different manner. When an interrupt occurs on this line it is forced to the routine RTC. Initially a check is made on whether the user has exeeded his time allocation, a time slice of 40 milliseconds. If he has, a check is made for requests for service from the other users. If requests are present an exit is made to the user next due for service, who has the next time slice allocated to him. However, if service has not been requested by other users, a further time slice is allotted to the current user. A check is made for him requiring the 10 millisecond clock to interrupt.

If he does and the enable toggle is set, the toggle is cleared and a return is made to the address given in the user's 10 millisecond interrupt link. If he does not require this interrupt a normal return is made.

Differences from normal running

The interrupt handling facility has obvious disadvantages for the running of real-time programs. Two obvious ones are:

1. Time profile distortion. This is due to the encroachment of the executive and the sharing of available time between users. This in turn leads to 2.
2. Slow reaction rate. While this is a disadvantage, in the applications handled in this system it proved to be a minor one.

Despite these disadvantages, the system has proved an invaluable tool in increasing program throughput as powerful debug facilities which would normally be available only for linear programs may be applied to real-time ones.

It should be possible to implement this type of system on any small computer with at least 16K words, a word being at least 16 bits long, possessing some form of memory protection hardware and the facility for running user's programs in a different mode to the executive, privileged instructions being able to be run only by the executive.

Acknowledgement

The authors wish to thank Mr. S. Chisman for his help and the Post Office Corporation and Scientific Control Systems for permission to publish this paper.

Reference

Lima System for Honeywell DDP 516 GPO Research Station 1970. (Available from D. G. Bennett, c/o GPO Research Station, Dollis Hill.

Correspondence

To the Editor
The Computer Journal

Sir,

The claim is often made that the argument transmission method used by certain FORTRAN processors (notably those for the IBM 360) in which arguments are passed by value and the possibly modified value copied back on return is non-standard. See for example the comment in J. M. Chambers (1971), page 314.

This is not the case. The USA Standard FORTRAN Report (1964) explicitly allows this mechanism in addition to the more usual call by address (sometimes called call by reference). The relevant quotations from the standard are as follows:

> 'If an entity created by argument substitution becomes defined or undefined (while association exists) during execution of a subprogram, then the corresponding actual entities in all calling program units becomes (sic) defined or undefined accordingly. (ANSI standard section 10.2.2.)
>
> 'If a function reference causes a dummy argument in the referenced function to become associated with another dummy argument in the same function or with an entity in common, a definition of either within the function is prohibited.' (ANSI standard section 8.3.2.)

With respect to the first requirement, the only difference between the two methods rests on whether the corresponding entity definition or undefinition occurs immediately or on subroutine exist. This could only be distinguished by means of a definition of the type explicitly prohibited in the second requirement quoted.

Thus both mechanisms must be regarded as meeting the standard and any program written in accordance with the standard will be insensitive to which of the two mechanisms employed.

Yours faithfully,
R. B. K. Dewar

Department of Computer Science
Illinois Institute of Technology
Chicago
Illinois 60616
USA
30 September 1971

References

Chambers, J. M. (1971). Another round of FORTRAN, *The Computer Journal*, Vol. 14, No. 3, pp. 312-314.

USA Standard FORTRAN (1964). *CACM*, Vol. 7, pp. 591-625.