provided sufficient storage and peripheral equipment is available.

This pattern of operation is to be repeated for the larger customer installations which are now being programmed.

The method of operation employed is a development of that used on earlier computers. An operator known as the On-Line Controller is responsible for the operation on the computer floor, and he is assisted by Loaders who load data and tend the magnetic-tape decks and the printers. The On-Line Controller is supported by an Off-Line Controller responsible for scheduling the work and assembling everything required for each run on the computer. This, apart from the data and standard operating instructions, includes a detailed operating sheet for the On-Line Controller, which records briefly and explicitly what is to be done.

This operating procedure applies to both trials and operational work. On LEO Service Bureaux all trials are carried out by the operating staff, the programmers preparing the necessary operating sheets.

Conclusion

To summarize the effects of time sharing:

The introduction of time sharing on large installations for convenience and higher efficiency has speeded the re-organization of computer applications into data vetting, file-processing, and printing operations.

The programmer is not concerned whether his program is to be run on a time sharing basis or singly—all the necessary work is dealt with by the Master Program which, in addition, performs routine file control and logging functions. He must, however, know the store capacity and peripheral equipment available for his use; this will not necessarily be the whole of the computer.

The operator likewise is not concerned with the complex structure of programs but merely acts and deals with them one at a time. All the complications associated with the operation of more than one program are dealt with by the Master Program which thereby relieves the operator of these problems and worries. Whilst an operator reacts, the computer continues with work on other programs.

In conclusion, therefore, it can be seen that it is possible to operate two, three, or more programs on a time sharing basis and thereby achieve more than the nominal 100% efficiency from the computer. It must be emphasized, however, that the selection of these programs must be done judiciously to make the fullest use of the potential power of the computer.

(For discussion at Cardiff see p. 33)

Operational experience of time sharing and parallel processing

By M. R. Mills

The distinction is made between interrupt time sharing and parallel processing. Operational experience on the Honeywell 400 computer shows that considerable savings can be made by time sharing simple programs such as printing along with a main program. Parallel processing on the larger H-800 computer is a much more powerful technique and several complex programs can easily be accommodated simultaneously. An example is quoted where in 68 hours a file of 2,300,000 cards was read, converted and sorted, in parallel with the normal production of other results.

Designers and users have long ago realized that a computer in processing data spends a high proportion of its time on input or output, during which time the central processor is only fractionally employed. The desire to increase the computer efficiency by performing an alternative program during the idle time is now being met by several manufacturers using two separate approaches—Interrupt Time Sharing and Parallel Processing. The former is simple and inexpensive in hardware, while the latter is much more powerful, so that both methods have their merits. A summary of replies to the questions raised by the Chairman is given at the end of the discussion of each of these approaches. (See also p. 33.)

Interrupt time sharing on H-400

Interrupt time sharing utilizes an interrupt signal from a busy peripheral device, such as a card reader, printer or magnetic-tape drive, to switch the central processor from one program to another. In some machines this interrupt signal must be programmed whilst in others, such as the H-400, it is automatically generated when a busy device becomes free. Basically a simple peripheral program performs one program cycle each time the associated device is free—the rest of the computer time is spent on the alternative program.

Since May 1962 the Honeywell 400 computers have been using interrupt time sharing for productive work.
The installation that I operated in June had four magnetic-tape drives, 1,024 words of core storage, a card reader, and a buffered high-speed printer.

The most-used combination of programs is interrupt printing from magnetic tape with simultaneous tape updating or sorting, though many other combinations are possible. The timing is such that a one-hour print run makes available within the hour at least 43 minutes for alternative processing. During the print run a sort on three tapes of about 75,000 items could be performed.

Practical aspects

Now to the practical considerations of interrupt time sharing. First, time saving is not as much as it seems. In the example just quoted the print run of 60 minutes and a tape sort of 40 minutes was time shared into one hour. Run consecutively, the three-tape sort could have been run as a four-tape sort taking three-quarters of the time, and there would have been no necessity to produce an intermediate print tape; thus the actual comparison is 1 hour time shared compared with 1 hour 25 minutes serial processing.

It should be noted that all requirements for interrupt processing are contained within the standard interrupt programs supplied by Honeywell; the main program is unaffected and there is no supervisory program for time sharing. Jobs such as printing or punching are being time shared at present, and a random inquiry to disc store is being programmed. Only simple jobs are envisaged as suitable for this method of processing for the following reasons.

(a) They form a significant proportion, say 20% of the total data-processing load.
(b) They are easy to schedule because they require no elaborate consideration of the computer equipment. Typically they contain about 40 instructions and can thus fit easily into a small core store, together with the main program.

More ambitious time sharing presents serious problems. For instance, two main programs run simultaneously may demand a hardware configuration almost double the normal requirements, and unless full use is made of this larger installation its occasional use for multi-program time sharing may prove to be a financial loss.

(e) The interrupt time to leave the main program and then return to the same point is 380 μsec.
(f) There is no priority queue to plan.
(g) There is no program protection by hardware to prevent programs corrupting each other. This is a point I wish to deal with later.
(h) It is feasible to run programs whilst testing others, but personally I would prefer not to do so.

Finally, a measure of the time which is not time shared can be deduced from my personal experience. The installation at which I worked spent 70% of its time on program assembly and check out (or debugging). Assembly time was invariably time shared with printing, as was one large updating run. Throughout it was never the intention or the practice to save more than about one hour per shift by time sharing, though larger output volumes could increase the efficiency by several hours.

Parallel processing on H-800

The operator requirements are quite different and the scheduling much more complex when a variety of programs are time shared. Further, program switching by program instructions in core storage is expensive in a large machine and so to meet the demand of the larger user the Honeywell 800 was designed to perform parallel processing rather than interrupted time sharing.

The H-800 is a medium to large computer in design, and the installation in the Honeywell Service Bureau in Moorgate, London, has eight tape drives, 8,000 words of three-address core storage, printer, card reader, card punch, paper-tape reader and punch.

In parallel processing the program switching is performed by hardware circuitry, with no loss of time, as opposed to programmed interrupt time sharing. In the H-800 this has been achieved by including a separate control core memory, split into eight independent program channels or program groups, each with its own 32 special registers (including a sequence counter and index registers). This control memory is supervised by a hardware multi-program control which takes one instruction in turn from each program group in demand.

Imagine for the sake of simplicity a program consisting entirely of additions. This program, if running alone, would perform about 40,000 three-address additions per second. Two such programs in the H-800 would each run at 20,000 additions per second, whilst if eight similar programs were switched on, each would accomplish 5,000 additions per second. In this respect the machine subdivides itself into as many computers as there are programs in demand. Whenever a program stops, the others run correspondingly faster. However, one rarely wishes to run a number of computing programs in parallel. The usefulness of multi-program control becomes apparent whenever a program addresses a peripheral device. If this device is busy then the program group is put into a temporary "off" state, and only the remaining programs in demand are processed. As soon

Summary

To summarize:

(a) The H-400 system time sharing first commenced actual production in May 1962.
(b) On an average two programs are run together, and the maximum number appears to be three.
(c) A minimum configuration of four magnetic-tape drives, 1,024 words of core storage, a card reader, and a buffered high-speed printer is required.
(d) There is no time sharing supervisory program.
The continuous logging of computer activity on the typewriter is, I believe, one of the most important features of the system. Attempts by the operator to circumvent unforeseen difficulties during a production run, by moving a few unrecorded switches, almost invariably produces invalid computer results. Such action would not be possible with this kind of Console.

I now wish to cover the two methods of parallel processing which we are using—automatic and manual.

### Automatic parallel processing

I have already mentioned the Executive Scheduler. This produces a program tape for the automatic parallel processing run. The production run is controlled by the Executive Run Supervisor which occupies 512 locations. This Supervisor is concerned with loading new programs, and tape-error correction. It is switched off during parallel processing.

Table 1 shows a page of the log during a short production run. The red (italic) typing is by the operator—the black by the computer. Each new line of computer output is automatically proceeded by the program group number. The Supervisor is loaded and identifies itself in group 0. The Operator starts the run and the first program VLS9 is loaded in group 2, but is not started because the card reader must be prepared. The second program starts automatically.

A S 2 in red is the Operator’s command to start group 2 and the Supervisor records that VLS9 is now running. At this point two programs—VLS9 and INSUR2 are running whilst the Supervisor is “off.” The printouts which follow on the left of the page show the progress of the run as new programs and segments are loaded. They give confidence to the Operator, and aid recovery after a machine or program breakdown.

The information starting on the left and going out to the centre, calls for immediate Operator action, whilst information on the right is for later use, such as reconciliation figures. This complete run took about 20 minutes from start to finish, during which time a detail file of 50,000 items had been sorted and then used to
Operational experience of time sharing

Up-date a master file, produce statistics, and edit out on to a print tape. 7,500 cards had been read on to tape, and another deck of about 1,000 cards had been read, totals computed, and the results printed. All eight tape drives, printer, and card reader were employed.

Whilst this automatic parallel processing is excellent for the stabilized user whose work loads can be scheduled at least several hours or perhaps days in advance, a Service Bureau has different problems. As with a customer during the first year's use of his computer, much time is spent on program compilation and check-outs. During this period the work load fluctuates and a more flexible operating routine is required. This is provided by a manual parallel processing. In automatic processing the operator controls the Executive Run Supervisor which in turn controls the several production programs. In manual processing the Operator directly controls each of the several programs in store.

Table 1

Typeout during automatic parallel processing of four programs

(Lines printed in italics below would be printed in red on the typewriter)

<table>
<thead>
<tr>
<th>R 11 Ex</th>
<th>S 50 0000 Ex</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 EXECUTIVE PRODUCTION RUN</td>
<td>0 EXECUTIVE PRODUCTION RUN</td>
</tr>
<tr>
<td>0 DATE: 62081400000</td>
<td>0 DATE: 62081400000</td>
</tr>
<tr>
<td>0 ARGUS AND FACT P.M. RUN</td>
<td>0 ARGUS AND FACT P.M. RUN</td>
</tr>
<tr>
<td>A S 0 Ex</td>
<td>A S 0 Ex</td>
</tr>
<tr>
<td>0 VLS9 WAITING2</td>
<td>0 VLS9 WAITING2</td>
</tr>
<tr>
<td>0 LOAD CARD DECK (VLS9 INPUT)</td>
<td>THEN START VLS9.</td>
</tr>
<tr>
<td>0 INSUR2 STARTG5</td>
<td>0 INSUR2 STARTG5</td>
</tr>
<tr>
<td>0 JUST START INSUR2</td>
<td>0 JUST START INSUR2</td>
</tr>
<tr>
<td>5 SORTIT</td>
<td>5 SORTIT</td>
</tr>
<tr>
<td>5 MODIFIED</td>
<td>5 MODIFIED</td>
</tr>
<tr>
<td>A S 2 Ex</td>
<td>A S 2 Ex</td>
</tr>
<tr>
<td>2 START OK</td>
<td>2 START OK</td>
</tr>
<tr>
<td>0 REDYCALT</td>
<td>0 REDYCALT</td>
</tr>
<tr>
<td>2 LAUNCH11</td>
<td>2 LAUNCH11</td>
</tr>
<tr>
<td>5 TO MERGE</td>
<td>5 TO MERGE</td>
</tr>
<tr>
<td>5 GENRATED</td>
<td>5 GENRATED</td>
</tr>
<tr>
<td>2 EDIT H</td>
<td>2 EDIT H</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>3 TRAFICIG</td>
<td>THEN START VLS9.</td>
</tr>
<tr>
<td>5 MERGED</td>
<td>VLS9 BAD CARD *GP4:0149</td>
</tr>
<tr>
<td>5 UPDATE</td>
<td>VLS9 BAD CARD *GP1:0062</td>
</tr>
<tr>
<td>4 NEXT INPUT REEL INVENT</td>
<td>INSUR2 END OF MONTH FIGURES ONLY</td>
</tr>
<tr>
<td>0 VLS9 OVER G2</td>
<td></td>
</tr>
<tr>
<td>0 VLS10 STARTG2</td>
<td></td>
</tr>
<tr>
<td>0 JUST START VLS10</td>
<td></td>
</tr>
<tr>
<td>0 CREDIT WAITING6</td>
<td></td>
</tr>
<tr>
<td>0 LOAD CARD DECK(CREDIT INS).</td>
<td></td>
</tr>
<tr>
<td>2 LAUNCH12</td>
<td></td>
</tr>
<tr>
<td>A S 6 Ex</td>
<td></td>
</tr>
<tr>
<td>6 START OK</td>
<td></td>
</tr>
<tr>
<td>0 REDYCALT</td>
<td></td>
</tr>
<tr>
<td>2 COMPUT15</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
</tr>
<tr>
<td>0 INSUR2 OVER G5</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
</tr>
<tr>
<td>0 CREDIT OVER G6</td>
<td></td>
</tr>
<tr>
<td>2 DETAIL38</td>
<td></td>
</tr>
<tr>
<td>0 VLS10 OVER G2</td>
<td></td>
</tr>
<tr>
<td>0 END RUN REDYCALT</td>
<td></td>
</tr>
<tr>
<td>* EXEC PROD RUN OFF AT 14:15 HRS.</td>
<td></td>
</tr>
</tbody>
</table>

31
Manual parallel processing

Table 2 shows the log during manual processing of three programs. The Operator is first loading a card-to-tape transcription program, then a tape-to-printer transcription program, and finally the assembly program. The latter assembled 10 programs, adding new programs to the program file whilst correcting and amending old ones. Meanwhile 6,000 cards were being read and 3,200 lines printed. The run took about 20 minutes and used eight tapes, printer and card reader. Neither automatic nor manual parallel processing require priorities to be arranged; if a job must be done quickly it is run alone, or only with peripherally speed-limited programs.

Summary

(a) In London we commenced parallel processing in April 1962, almost as soon as the machine was installed.

(b) The average number of programs run together is two or three, but thirty or more have been included in a sequential run.

(c) The maximum number of programs performed in parallel has been eight—usually run for demonstration purposes only.

(d) Whilst a minimum H-800 computer, of 4,000-word store, five tapes, printer, and card reader, is probably the smallest required for parallel processing, for extensive parallel processing, a configuration considerably in excess of this figure is needed as, for example:

An insurance company in America has had an H-800 for some 15 months. The following equipment was used in a card conversion program:

8,000 words core storage.
2 card readers.
1 printer.
9 tape drives.

In 68 continuous hours the following was accomplished.

The conversion of 2,300,000 cards to tape.
Edit and validation of these card images.
The printout of the error items and their re-submission.
The sorting of the output file of 20 tapes into order (which took 12 hours).

A normal invoicing run of 4 hours.

Some checkout of new programs.

This is a good example of what can be accomplished by an experienced user who pays attention to his scheduling.

(e) The Executive Run Supervisor occupies 512 locations but does not partake in parallel processing, merely in sequential loading, and automatic error correction.

(f) No time whatsoever is taken to move from one program to another.

(g) There is no priority queue.

(h) Program protection by hardware is not incorporated, because if one considers the most elementary protection, which is the division of storage between programs, then the memory cycle is liable to be lengthened by up to one-third. This greatly increases the production run time of programs on the machine. However, if one has a comprehensive program-test system which detects storage violations, then the number of such program errors liable to occur in production is small. One must compare permanently slowed-down processing using storage protection, as against a rare re-run.

(i) I have never debugged programs while running other tested programs, but several users do this consistently—to my surprise! They appear to be satisfied.
Operational experience of time sharing

At present, August 1962 in London, for a considerable period of the day we are not parallel processing—this is due to our ability to process serially much of our present commitments within our prime shift. About three hours per day is an average figure for parallel operations.

Conclusions

From my experience a great wastage of time occurs between loading programs, and if this time can be cut by automatic sequence, for instance by an Executive Run Supervisor, perhaps even without parallel processing or time sharing, an extra two hours per shift may become available.

Automatic parallel processing is very simple to use, and I feel it one of the cheapest ways of getting most, if not all, of a quart into a pint pot.

For the future, already one customer is considering having a single program in store and controlling it by three program groups in parallel. By this method he hopes to perform three similar sorts or updates on different tapes at the same time. A trace routine has been written by a colleague in which one program group looks over the shoulder, so to speak, of an untried program, and prints out every sequence change.

We are finding many other possibilities come to mind, and I think it will be several years before the full potential of this system is realized.

Acknowledgement

I am grateful to several Honeywell customers for permission to include in this paper their experiences of operational parallel processing.

Summary of discussion at Cardiff Conference

Mr. R. M. Paine (C.E.I.R. (U.K.) Ltd.) (Chairman’s Opening Remarks): This session, entitled “Practical Time Sharing Experience,” is concerned with the actual experience of running time shared programs on existing machines. The speakers have been chosen because of the actual experience of their firms and we can look forward to some interesting information, and comparison of methods and results.

To make sure the speakers give us some definite answers based on actual experience I have given them some points I would like them to follow. These questions, which are written on the blackboard as a reminder, are as follows:

(1) When did the time sharing system first work?
(2) The average number of programs actually run together.
(3) The maximum number of programs actually run together.
(4) The size of computer store and number of input/output units required to run these programs in parallel.
(5) The size of the supervisory program.
(6) The time taken to interrupt and move from one program to another, in actual operation, not just in design theory.
(7) How priorities are arranged and worked out in practice.
(8) Is there program protection by hardware to prevent programs running wild and interfering with other programs?
(9) Have you debugged programs while running other production programs?
(10) Is there any measure of time not used for parallel running due to the difficulty of filling programs to the available configuration?

If the speakers do not cover these points during their talks, I will ask them quickly to answer them at the beginning of the discussion.

Mr. J. W. Lewis: To answer the Chairman’s questions one by one: The time sharing system has been working now for over a year and the demonstration I refer to in my paper was given on 2 January 1962. The average number of programs run together is two, excluding the Master Routine. At the present time we have three completed computers, the two in London are both working 168 hours a week mainly on program trials. In practice, with only a light operational work load, it is difficult to make full use of time sharing. The greatest number of commercial programs run together is three. In fact there are five levels of program priority within the Master Program which make a maximum of eight programs run together.

The size of computer store and number of input/output units required to run these programs in parallel: I will avoid this question by saying that the size of the computer installation and the nature of the work to be processed very much determines the amount of time sharing which can be done. The minimum installation, on which one can do time sharing effectively, is probably two magnetic-tape decks with a paper-tape reader and printer and a store of 4,000 words. In fact this was the size of the configuration used for the demonstration I referred to.

The size of the supervisory program: In my paper I mentioned that the Master Program is in five main parts. The full Master Program occupies about 2,000 words of which the priority-control routine requires only 24 words for each program. I do not know the time it takes to interrupt. I have not got the figure with me. However, what I think is more important is to know what the total effect of interruption is; I mentioned a figure of 1% of the total running time in my paper.

How are the priorities arranged? Generally speaking, printing would be at the top of the priority queue followed by the input program with file processing at the bottom. In point of fact, the order of priorities can be altered within a computer run should the situation require it.

Program protection: Yes, we have this feature. We offer it as an optional facility and feel it is essential when operating on a time sharing basis. The price one pays is small, less than 2% of the cost of an installation. Detection of a corruption can be provided without loss of speed: prevention slows the cycle time of the store by about 5% in which practice may not have any overall effect on the running time of a job.

Have we debugged programs while running other production programs? Yes, I mentioned this point in my paper.

Is there any measure of idle time due to the difficulty of filling programs to the available configuration? This is difficult to answer as I do not think we have really got sufficient experience to give a reasoned answer. In practice, any spare
Operational experience of time sharing

capacity can be absorbed by a program operating at the bottom of the priority queue, possibly at very low efficiency.

Mr. J. A. Fotheringham (Ferranti Ltd., London): Orion, like Leo, has a core store and transfers from peripheral equipment into the core store under the control of what we call "control units." As we can attach several peripherals to each control unit we have the problem of sharing control units as well as sharing store. One particular feature, in which Orion may be slightly different, is that we also have drums to back up our core store. These drums also have control units, but it is usual to have only one drum control unit, even with three or four drums. Some of the most interesting and unexpected things we have discovered about time sharing have been the problems of sharing control units.

Magnetic tape: one usually has two control units, one can have four. Paper tape usually has one or two control units. Printers and card readers usually go through the same set of control units. I am not sure what the typical number is.

The organization of the time sharing is done on the interrupt system by means of a "time sharer." This is a feature of the organization program and this time sharer controls the switching from one program to another in the normal way with a priority system. When a program which is running is unable to continue, because it is waiting for a transfer, the time sharer comes in and looks for the program of next highest priority which can continue. At the end of any transfer a better choice may be possible, so the priority list is scanned again to activate the highest-priority program.

There is independence of programming on Orion. We have a reservation system on the store which means that you can rely on the fact that your store is only going to be interfered with by your own program, and not by anybody else's. We also have a reservation system on the individual peripherals; so that while your program is running, the peripherals you require are allotted to you.

The order structure of the machine allows each program to use 64 accumulators. These are not strictly accumulators, they are the first 64 registers of your own storage area. Your storage area has to begin with an address which is a multiple of 64. The programming scheme provides all the necessary relativization for this and by hardware an origin is added to your 6-bit address so that it turns into an address in the first 64 of your registers.

The time sharer is called in much as in Leo. If you do a peripheral transfer by asking for input to a particular area of store, or output from an area of store, this area is locked out while the transfer is proceeding and any attempt to use it—the violation of this lock-out—produces an interrupt causing program change. There are two sorts of lock-out, which we call "strong" and "weak" lock-outs. In the strong lock-out the part of the store locked-out is inviolable, and may not be referred to in any way. In a weak lock-out you can read from the relevant part of the store but you cannot alter its contents; so for output instructions a weak lock-out is applied and your program is only interrupted if you try to change the contents of your output area before the output operation has been completed. During input this is not sufficient, for obvious reasons, and because input checking tends to go in bulk units.

Another type of interrupt, which may cause program change, is a reference to a basic peripheral. I mentioned also completion of a peripheral transfer, which may allow a higher priority program to proceed. Lastly we have an interrupt every second to stop programs going wild: every second the monitor routine is called in to look at what is happening. You can also put in your program, statements as to how long the next section of the program is expected to take; every second the state of the program currently running is inspected to make sure that the program is not in fact exceeding the time it is expected to take. Clearly you do not refine your expectations too narrowly.

The time sharer, as I said, works on a priority system: I am not sure how widely accepted the rules for priorities with this sort of interrupt time sharing are; I think the necessity for them is appreciated. First of all top priority: programs which are using equipment which cannot be stopped, in other words programs which we cannot afford to let take substantially longer than they would take running on their own. An immediate example is the Xeronic printer. Subject to this first requirement, the arrangement of priorities is again designed to get the most out of the system, and this is done mainly by allowing programs which use the central computer very little to have a high priority, and the programs which use the central computer a lot to have the lower priority. The programs which use the central computer only a little will be interrupted by trying to use peripherals, and allow the other programs to move in. We found by simulating this system that this was not quite straightforward, and this is where the problem came in of using a single control for a set of peripherals which may be divided between different programs, and of using a single control for drum transfers. You want several programs to be able to use the drum, and of course this is where this business of competing for the control comes in.

At first glance it looks as though a program which computes for, say, 10% of the time and the rest of the time is waiting for drum transfers—perhaps a drum transfer every 100 milliseconds between which it computes for 10 milliseconds—should fit in very nicely with a program which does the reverse—computes for 100 milliseconds and then asks for 10 milliseconds' worth of drum transfers. This second program would of course have the lower priority, being more computer limited. However, it turned out that the first program, because it had high priority, was always asking for drum transfers, and the second program did not get a look in, because every time it asked for a drum transfer, the drum was busy executing a transfer for the first program, and as soon as that drum transfer was completed the first program was allowed to continue and produced another drum transfer. As a result the computer-limited program did not get a look in at all.

So, in fact, we had to introduce a slight refinement to this system, and we now have a peripheral queue. In this queue the requests for use of a particular peripheral, or a particular control unit, are entered in chronological order, so that if program A, which is the drum limited one, is held up with, say, two drum transfers in the queue, and program B comes in and computes and then asks for a drum transfer, that request for the drum transfer goes into the queue and no other requests for program A are allowed to jump ahead of it, even though program A has higher priority. Thus program B does eventually get its drum transfer and is able to continue when program A is held up. The gain in the use of the central computer time by introducing this was from a 10% usage in the first case (which, of course, was completely inadmissible) to 78%. This is not quite as good as it might be; the reason again was that by the order of priority A got rather too many requests for drum transfers and every time B wanted a drum transfer there were always two transfers for A waiting.
The final interesting point I wanted to make is that our last refinement was to introduce a bottom priority program, which was always there, and which had the effect of taking the request at the bottom of the peripheral queue and putting it to the top. This meant that at any time when the drum was so clogging up the system that no program could continue, we shuffled round the peripheral queue. This put up the usage of the mill from 78% to 89%, so I find that quite interesting.

In terms of practical experience, we do not have very much to say. We have been doing time sharing on Orion in a fashion. We have been developing peripherals and systems programs at the same time. At the moment we have not got the reservation system in action on Orion, so we have done this fairly crudely. We have divided the 4,000 words of core store into eight blocks each of 512 words. Programs using these have been very carefully checked to see that they do not go outside their own area, and we have been quite happily sharing the engineers’ programs which they need to use for checking up peripherals, and the programmers’ development of their programs. In a system like this it is pretty well impossible to say what sort of percentage you are getting, or what sort of sharing you are getting. Without time sharing they would have to be done one by one, but as to the load on the central computer I am afraid we just could not say.

As this topic is supposed to be entirely on practical experience, perhaps it would not be worth while talking about any more simulation.

What else have I not covered in the Chairman’s questions? When the time sharing first worked? We have been doing this kind of thing for a couple of months. Average number of programs run together: we expect it to be not more than three. These may well be separate parts of one single program. You can in fact split a program into several parts and time share them. Supervisory program: this takes 512 words of core store. Time to change from one program to another is about 2 msec. Priorities I have covered. Hardware protection of programs I have covered. Debugging during production runs: we have not really done any, because we are not yet doing production runs but we have done a whole lot of debugging runs altogether, which is surely even more difficult than debugging during production runs. Measure of usage of central computer: again no practical experience. Simulation suggests that we would use the central computer for productive work about 98% of the time.

Mr. A. H. Armstrong (United Kingdom Atomic Energy Authority): The question which has to be answered is: does time sharing pay?

In time sharing experience it is often true that one of the problems, e.g. printing from magnetic tape, could be carried out on a less powerful computer renting at only a few per cent of the rent of the main computer. If this small program slows the progress of the main program by more than a few per cent, then time sharing is uneconomic in this case, for it would be cheaper to rent the small computer in addition, thus reducing the rental charges on the main computer.

Mr. J. W. Lewis: I would agree that the point Mr. Armstrong makes is valid, provided you can economically balance the cost of renting the small computer against the extra use you would make of the main computer. In my own experience this is not normally possible. One must consider the work load for the installation and evaluate the relative costs for the two different types of computer configurations.

Mr. M. R. Mills: Firstly, the H-800 with no master routine merely slows down the main job by the few instructions required to initiate each print cycle. Secondly, our experience has shown that the greatest waste of time occurs due to operator delays in loading the next program after the previous one has ended. This wastage will be aggravated if two computers are being run instead of one. However, if you can eliminate this delay, the interruption time between running consecutive programs, by automatic sequencing of programs, I think you will gain a considerable advantage.

Mr. P. J. H. King (Associated British Picture Corporation): How does the time sharing facility affect the type of programs that are written? For example, if one has a job which involves three different types of output and one has three output channels, there also being input and file updating, in a normal machine one would write one program to do the whole job: with a time sharing machine one would rather write several small programs and do each printing operation, say, separately?

Mr. J. W. Lewis: Operating a computer on a time sharing basis does mean that in practice a job tends to be broken down into input, file processing and printed output operations. I am not certain that this is a direct consequence of time sharing, but rather of the fact that one is using a faster computer. Data errors can be costly in re-run time, therefore it is worth while editing all data before processing. File processing and printed output are seldom well balanced in the types of commercial file processing jobs being performed on this type of machine and consequently it is natural to split the two parts of the operation. In this way, one can make fuller use of the equipment available.

Mr. Fotheringham: I think it must be realised that if you have a time sharing machine you have an extra facility. To use it you have to think about it. For example, the question of how much core store you use in a particular program: when you write a program so as to use only a part of your core store, you are consciously fitting your program into the time sharing system. Therefore I think it does exert some influence on the program you write, but whenever you get a new facility in anything, you have to pay some thought to use it.

Dr. K. D. Tocher (United Steel Companies Ltd.): Old-fashioned programmers plan the strategy of their programs to minimize the waiting time for peripherals. Would these attempts lead to difficulties in time sharing machines? It might be that too little opportunity is given for low-priority programs to operate. Could the speakers comment?

Mr. J. W. Lewis: The desire is to have all parts of the computer working to full capacity and, as Dr. Tocher remarks, this did affect the strategy of old-fashioned programmers. There is no need to make use of time sharing on LEO III and the computer can continue to work on a single program basis. What time sharing does is to take the burden of balancing a program off the programmers’ shoulders.

Mr. Fotheringham: I don’t think so, particularly if you are so practised at it that you are good at it. The only question is, how much programming time does this attempt to make your peripheral transfers efficient on their own take up? Again it is a question of striking a balance. The more times you have to change programs in the machine the more wasteful will your installation appear to be. In any case, if your installation is peripheral limited there is nothing you can do about it, and however well you program, your computations will not take up all the transfer time. Even so, if you can efficiently and quickly be clever, by all means do so.

Mr. Mills: We have a rather ingenious approach for overcoming the problem of a program with, say, both heavy
Operational experience of time sharing

calculating load and heavy printing load, the central processor normally being held up by the peripheral device.

It is very often convenient to use subroutines to do your printing and you can arrange all the print routines in a different program group.

Since program groups may switch each other on and off, program 1 (your main program) gets ready the print information for several lines of print and switches on program 2. Program 1 continues processing at full speed whilst program 2, though peripherally bound, eventually finishes its printing then switches itself off to await the next cycle of events. The programming effort for this approach is small and though we as a Service Bureau are not using it very much, one or two of our customers rely on this device.

Mr. B. V. Piggott (Eastern Electricity Board): The various programs of a commercial user need different stationery and the volume of work requires more printing time than the time needed for data processing.

Have any of the speakers had any experience of running two or more printers simultaneously at full speed?

Mr. J. W. Lewis: Since we have not a computer with two printer assemblers, we have no practical experience of this, although there is no reason why this cannot be done.

Mr. Mills: Since on the H-800 all the programs groups are independent, one merely writes two separate print programs one for each print run. Then one loads and runs both programs in parallel on an H-800 with two printers just as though one had two large computers both with separate printers. One just has two programs and they run in parallel at full speed.

Mr. S. N. Elders (M.P.N.I., Newcastle upon Tyne): Has the suggestion that a low-priority program should be given over to engineering test routines been used?

Mr. Mills: If insufficient automatic checks are provided and there are doubts of the reliability of the hardware, it is probably an excellent idea to have as a base load an engineering diagnostic routine proving that the machine is behaving properly. We have not found any need for it.

Mr. Paine (Chairman): In contrast to the other speakers, your system does not have hardware protection of programs. Could you tell us the reason for this and whether you suffer from programs running wild?

Mr. Mills: We did consider this and we are still considering it. I think our experience is that it would have been something of a frill. We have very comprehensive programming systems, which enable us to run our programs under a program-test system, during production runs. This means that the results are the same, but while they are being run there is a monitor system working looking all round the store to see that there are no store violations taking place. In fact, this is quite simple. Your program runs as one of the eight, and another program is doing the monitor, looking round the store. Should you violate the store then you know about it. Now, we can carry on doing production work at full speed, the same rate as you would in normal working, for some time, while you are testing. We find, in fact, that the number of violations taking place is very rare. In fact they may take place once every six months. Alternatively, if you have hardware protection you pay for it, you pay very severely, because you take up considerable time in a store cycle while you are doing your tests on the locations you are using.

In fact, some machines extend the store cycle by up to a third when applying store protection, prior to writing into core. One is comparing, therefore, a machine with store-violation protection, which runs somewhat slower all the time, against a similar machine running faster, which may necessitate a re-run once every six months or so. Since restart points are automatically provided by the H-800 Executive system, it is merely necessary for the operator to switch off the faulty program and then perform a 15-minutes re-run. Our experience is that hardware storage-violation protection is not worth while. Several of our customers have reported running programs in parallel under Executive for well over a year without any trouble whatsoever. That is the reason we have not fitted special hardware.

Correspondence

To the Editor,
The Computer Journal.

Sir,

Revised Report on ALGOL 60

The following explanatory footnote is printed, apparently as part of the text of the Report, on page 352 of your January number:

"Whenever the precision of arithmetic is stated as being in general not specified, or the outcome of a certain process is left undefined or said to be undefined, this is to be interpreted in the sense that a program only fully defines a computational process if the accompanying information specifies the precision assumed, the kind of arithmetic assumed, and the course of action to be taken in all such cases as may occur during the execution of the computation."

For all I know this may illuminate the "Professor Friedegg"

Mr. S. N. Elders: (M.P.N.I., Newcastle upon Tyne): Has the suggestion that a low-priority program should be given over to engineering test routines been used?

Mr. Mills: If insufficient automatic checks are provided and there are doubts of the reliability of the hardware, it is probably an excellent idea to have as a base load an engineering diagnostic routine proving that the machine is behaving properly. We have not found any need for it.

Mr. Paine (Chairman): In contrast to the other speakers, your system does not have hardware protection of programs. Could you tell us the reason for this and whether you suffer from programs running wild?

Mr. Mills: We did consider this and we are still considering it. I think our experience is that it would have been something of a frill. We have very comprehensive programming systems, which enable us to run our programs under a program-test system, during production runs. This means that the results are the same, but while they are being run there is a monitor system working looking all round the store to see that there are no store violations taking place. In fact, this is quite simple. Your program runs as one of the eight, and another program is doing the monitor, looking round the store. Should you violate the store then you know about it. Now, we can carry on doing production work at full speed, the same rate as you would in normal working, for some time, while you are testing. We find, in fact, that the number of violations taking place is very rare. In fact they may take place once every six months. Alternatively, if you have hardware protection you pay for it, you pay very severely, because you take up considerable time in a store cycle while you are doing your tests on the locations you are using.

In fact, some machines extend the store cycle by up to a third when applying store protection, prior to writing into core. One is comparing, therefore, a machine with store-violation protection, which runs somewhat slower all the time, against a similar machine running faster, which may necessitate a re-run once every six months or so. Since restart points are automatically provided by the H-800 Executive system, it is merely necessary for the operator to switch off the faulty program and then perform a 15-minutes re-run. Our experience is that hardware storage-violation protection is not worth while. Several of our customers have reported running programs in parallel under Executive for well over a year without any trouble whatsoever. That is the reason we have not fitted special hardware.

Correspondence

To the Editor,
The Computer Journal.

Sir,

Revised Report on ALGOL 60

The following explanatory footnote is printed, apparently as part of the text of the Report, on page 352 of your January number:

"Whenever the precision of arithmetic is stated as being in general not specified, or the outcome of a certain process is left undefined or said to be undefined, this is to be interpreted in the sense that a program only fully defines a computational process if the accompanying information specifies the precision assumed, the kind of arithmetic assumed, and the course of action to be taken in all such cases as may occur during the execution of the computation."

For all I know this may illuminate the "Professor Friedegg"

type of mind. But can anyone actively engaged with computers (which in reality respond to a rather coarse flip-flop two-value logic) read much more into it than that whenever the precision of arithmetic is not defined, it is left undefined?

Would not ALGOL be more widely acceptable if it were written in clear language and/or a consistent symbolic notation?

Yours faithfully,

C. H. R. Morris.

National Coal Board,
Cardiff.
27 February 1963.

[The footnote quoted is an exact reproduction of the official text of the Revised Report. An almost identical footnote was included with the original Report.—Ed.]