ITEM NUMBER 6404-0051 (T.I.E.)
19 pages

DATE March 4, 1964

AUTHOR K. E. T. Donner

TITLE SPOOF SIMULTANEOUS PROGRAMMING ON THE 1401

SOURCE IBM United Kingdom Ltd.,
Bowmaker House
Kew Bridge, Brentford
Middlesex, England

6404 0051

DISTRIBUTED BY
THE PROGRAM INFORMATION DEPARTMENT (TIE)
IBM CORP.
112 EAST POST ROAD
WHITE PLAINS, NY

Contents

SPOOF. SIMULTANEOUS PROGRAMMING OPERATIONS ON FOURTEEN-O-ONE

A paper based on the fact that 1401 Computer installations which have the Print Storage special feature may well be printer bound, and have considerable processing time available. The purpose of this study is to show how this processing time could be used to run a simultaneous program which does not use the printer. By use of the Tests "Printer Busy" and "Carriage Busy", it should be possible to run a second program without affecting the efficiency of the first to any great extent. The principle should be sufficiently generalised to be of advantage to many commercial 1401 installations.

SIMULTANEOUS PROGRAMMING ON THE 1401

1. GENERAL OUTLINE

1. What is meant by 'simultaneous programming'?

A normal computer program involves two distinct types of operations - internal processing and input/output. The central processing unit can only process one instruction at a time, but several input and output operations can be carried out together. When a computer's input/output facilities and its processing facilities are all being used as fully as possible, then that computer is being used efficiently, and there are various ways in which it is possible to carry out more than one function at once on a computer:

a) More than one kind of input or output at the same time. This again can be carried out in a variety of ways:

   i) "Buffered'operation. Buffered input/output units can operate quite independently of the rest of the computer, and the computer is only held up for the short period of transfer to or from the buffer.

   ii) Multi-channel operation. A non-buffered input/out operation will hold up one input/output channel. By having more than one channel, more than one input/output operation can be carried out at once.

b) Input/Output operations at the same time as processing

   i) As described above, buffered units make it possible.

   ii) 'Overlapped' operation. In this case processing is only interrupted for a short period as each character is transferred to or from the input/output unit. For the rest of the transfer

time other processing can be carried out.

iii) Mechanical operations can be overlapped with provessing - i.e. while the printer carriage is spacing stationery, processing can continue. This applies even when the unit is unbuffered and when the processor cannot carry out 'overlapped' data transfers.

In the case of the 1401, multi-channel input/output is not possible, but all the other kinds of multi-operation mentioned above are available - i.e. with the Print Storage feature, the printer is buffered; with the Overlap feature, all input/output can be overlapped with processing, and lastly, with or without the above features, some part of the mechanical input/output functions can be overlapped with processing. (Some input/output operations can also be performed concurrently by means of the combination instructions, e.g. Read-Punch etc.)

Because it is possible to carry out such a range of parallel operations, it is very likely that any one 1401 program will not in fact be making full use of the computer's potential. In other words, at some stages in the operation of the program this processing unit will be idle, at other stages one or more of the input/output units will be idle, when in fact other processing or other input/output operations could be being performed.

The phrase "simultaneous programming" is here used to refer to a means of utilizing this otherwise wasted computer time. Since the processing unit can only access and execute one instruction at a time, two programs can never be processed simultaneously in the fullest sense; but since, as mentioned above, during the execution of a normal program, the processing unit may well be idle at times, then this idle time could be used by another program. To the extent that the second program did not slow down the first, the two programs can be said to be operating 'simultaneously.' So long as the input/output operations could be overlapped with processi ng, such 'simultaneous programming' could well be a practical possibility on the 1401.

2. 1401 Configurations that would be best suited to this technique.

There are two features available on the 1401 which make this technique a possibility - the Print Storage feature, and the Overlap feature. As the majority of commercial users of the 1401 will probably not need Overlap but will want Print Storage, it is with the latter feature that this paper is primarily concerned.

Usage of the printer in a 1401 installation is often high, and several programs will be 'printer-bound' - in other words, the program will develop the data for a line of printing before the previous line has finished printing, and the program has nothing to do but wait for the printer. A line on the 1403 models 1 or 2 takes 100 milliseconds to print, and with Print Storage 98 m.s. out of this 100 m.s. is available for processing, and the extra 1403 time taken by spaces of more than one line is also available. If a large proportion of this available processing time is not in fact being used, then there is an obvious case for the introduction of a second program.

II DETAILS OF PROPOSED METHOD.

1. Characteristics of programs involved.

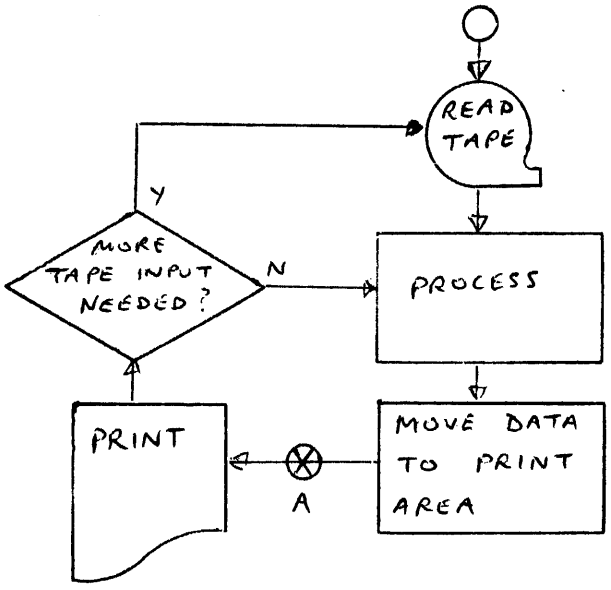
The program which uses the printer, which will from now on be called the 'printer program' should involve as little processing as possible (and as little input/output as possible, though some input is bound to be necessary). A tape-to-printer program would be ideal.

The second program, which will be called the 'process program', should involve as little input/output as possible, and as much processing as possible.

2. The interaction of the two programs.

A 1401 fitted with the Print Storage feature is also provided with two additional branch instructions - Branch if Printer Busy and Branch if Printer Carriage Busy. By making use of these tests, the process program can hand control back to the printer program as soon as the printer is idle. The printer program would then give the next print instruction, arrange the data to be printed on the line after the one now printing, and transfer control to the process program.

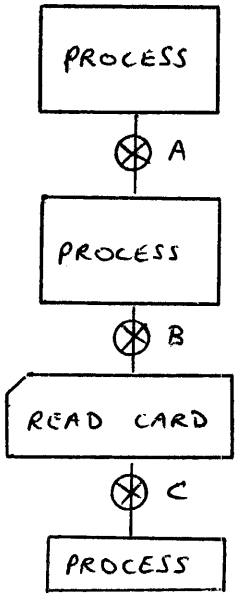
A very simple tape-to-printer program might follow the following lines:-



The exit to the process program would occur at point A - this ensures that when the process program returns control to the printer program (at point A), the next print instruction is given as soon as possible.

The process program itself might be of any type - the important thing is that it should contain as little input/output as possible. At intervals in the program there should be tests for Printer Busy and Printer Carriage Busy; if neither is Busy, then the process program transfers control to the print program. The process program is bound to contain some input/output, and even if it contains a considerable amount there may still be an advantage in running the two programs together. It has been assumed that the printer program is the primary program - in other words, that the objective is to keep the printer as busy as possible. With this in mind, it is worth adopting the same technique for the process program as was adopted for the printer program - i.e. just before any input/output operation in the process program, transfer control to the printer program, This means that when the printer program returns control to the process program, the input/output operation will be executed immediately. As the printer program transfers control at the moment when there is as long an interval as possible before the completion of the last given print operation, this ensures that the input/output operation in the process program is overlapped with printing.

Thus, a section of the process program might be as follows :-



A and C would only branch control to the printer program if the Printer and Carriage were not busy. B would branch unconditionally to the printer program. This might not utilise some of the available process time, but would ensure that the card read operation occurred as soon as control was returned by the printer program, and thus that the printer was not held up by the card read.

3. The control link between the two programs.

As already mentioned, each program will be required to 'transfer control' to the other, at various points during each. A means must be devised of ensuring that the printer program hands control back to the process program at the point where the process program left off, and vice versa. The easiest method of doing this is to include a small control program, called the 'bridge'. This 'bridge' would basically involve the following 4 instructions :-

```
(1)   PRINT     SBR     PRØCES + 7
(2)             B       O
(3)   PRØCES    SBR     PRINT + 7
(4)             B       O
```

During the printer program, whenever it was desired to transfer control to the process program, the program would branch to PRØCES, (i.e. Step (3) of the bridge). At PRØCES, the return address of the printer program would be stored in Step (2) of the bridge. The next step would branch to the location in Step (4), which would be some instruction in the process program. At intervals during the process program, the following three instructions would occur :-

```
BPB     *+10
BPCB    *+5
B       PRINT
```

Thus, whenever the printer and carriage were found to be not busy, the program would go to step (1) of the bridge. The return address would be stored in step (4), and control branched to the address stored in step (2) which, as we have seen, would be the return address stored when the print program was last left.

III SOME PROGRAMMING CONSIDERATIONS.

1. Core storage requirements.

Although the necessary modifications to the two programs would not take up much storage, it is of course necessary to have both programs in core at one time. This is the most serious limitation on the usefulness of this technique. It is possible that the time-saving to be gained would justify extra storage, but unlikely. However, by cutting down on input/output tape areas, and by considering the possibility of program overlays (or especially sub-routine overlays from 1311 diskpacks), it might well be possible to produce a slightly less efficient version of one or other program which could still lead to an overall time-saving when they are run in conjunction.

2. The simplest case.

The simplest case of this kind of simultaneous programming would occur when it was desired to run the same two programs together on every occasion, and when it was desired to load and start both programs together.

Assuming that both programs had already been written, what changes would be needed to the source decks, to ensure that the two programs could be run together?

(i) The control program. - This would have to be inserted, and the best place for the control program would be at the start of both programs (i.e., usually at location 333). When the bridge is initially loaded, the two Branch steps should contain the starting addresses for the two programs, and this ensures that whichever program is started first, the first time that program branches to the bridge, the bridge will transfer control to the initialisation steps of the other program.
If both programs are to use the same index registers, then the bridge should store and restore them, and a 'generalized' bridge would have to cater for this.

(ii) The printer program. - This would need to follow the control program and would need a branch to the control program before each Print instruction. (and before each carriage control instruction if a long

skip was expected). If it was not obvious that printing would still be going on when this stage was reached, then the branch could be a test for Printer Busy or Printer Carriage Busy.

(iii) The process program - This would need to follow the printer program. As explained earlier, tests for Printer and Carriage Busy would have to be inserted at intervals. The frequency of these tests would have to be determined by a balance between extra time taken by the process program in carrying out the tests, and printing time lost because of the interval between them. In general a test per 10 m.s. of processing should be adequate, i.e. on average per 40 instructions in the process program.

Even if both programs were always to be loaded together, there would be the questions of end of job procedures, since one or other program will end first, and the control program must be arranged so that the second program continues by itself. The first requirement is therefore a pair of switches, to signify to the control program which programs are running. During the initialisation of each program these are turned on. When one program reaches end of job, the switch for that program is turned off. The control program then tests to see if the other switch is off. If so, both programs have finished, and the control program goes to end-of-job halt. If the other program is still running, the control program must ensure that the bridge is 'closed', i.e. that the remaining program does not try and transfer control to the program which has now finished.

The simplest way of achieving this can be seen from an example :- The bridge contains the following steps :-

(1)	PRINT	SBR	PRØCES + 7
(2)		B	nnn
(3)	PRØCES	SBR	PRINT + 7
(4)		B	mmm

Assume that the Process program has finished, and the end-of-job routine has discovered that the print program has not yet finished. If the program places the address of step (2) in the bridge in step (4) then any branch to the bridge in the Print program will return immediately to the print program. Having closed the bridge in this way, the end-of-job routine should branch to step (2) of the bridge, which will contain the correct location for continuation of the printer program.

3. Independent assembly and operation.

The next stage is to arrange for it to be possible to assemble the two programs independently, and if necessary run them independently. All this is quite possible, and there are a number of different methods of approach - one will be outlined here; it has been assumed that the control program (i.e. the 4 steps of the bridge, plus the two switches already mentioned, plus, if necessary, provision for exchange of Index Register contents) will be assembled and loaded with the printer program.

From the point of view of assembly this presents no problems - the program makes no reference to locations outside the control program.

If the bridge is assembled in its 'closed' form, i.e.

```
(1)   PRINT     SBR     PRØCES + 7
(2)             B       START
(3)   PRØCES    SBR     PRINT + 7
(4)             B       PRINT + 4
```

then the program can be run as it stands. The branch in step (4) will remain unchanged until the process program is loaded and starts running, when the first transfer from the process program to the bridge at step (1) will 'open' the bridge.

Independent assembly of the process program will involve slightly more difficulty, in that all references to locations in the control program must be by actual machine address. Since the control program will be in locations 333 upwards, however, these locations will be known beforehand.

Independent operation of the process program will again involve some difficulty, since the control program will not be present. It should, however, be very simple to load a dummy control program into locations 333 upwards, which, as outlined above, would simply transfer control back to the process program whenever that program branched to the 'bridge'. The simplest course of all would be to NØP the branches to the bridge, but the aim is to make the simultaneous control as independent as possible of the programs involved.

4. A master printer program and a number of process programs.

It is envisaged that the most usual case of application of these

techniques will involve a long printer program, probably tape-to-printer, and that while this is running it might or might not be convenient to load and execute any one or more of a number of shorter process programs. This again should involve no difficulty. Firstly, the printer program would be loaded by itself and started running as explained above.

Any number of process programs can be available (with the linkages to the control program). When it was desired to load one of these, the printer program could be stopped by use of a sense-switch. This halt would be programmed to place the correct return address for continuation of the printer program in step (2) of the bridge, and then Halt. The process program could be loaded (if the printer program was reading cards this would add complications, but these should not be insuperable) and would go directly to its own initialisation. When the program arrived at the first test for Printer and Carriage Busy it would branch to the bridge, whence it would proceed to the correct return address in the printer program. After that the two programs would run together in the normal way. When one process program had been completed another could be loaded in the same way. (If it was necessary to clear storage, then the clear storage cards must be modified to avoid cleaning the printer program).

IV MISCELLANEOUS POINTS

i) Programs with considerable input/output operation

If it is desired to run two programs together both of which involve a large amount of input/output operations, then it would almost certainly be much more efficient to program the two programs together as one program.

The advantage of using the methods outlined here would only lie in the generalized approach, which means that any two programs assembled with the necessary linkages could be run separately or together, and if run together would certainly show some saving in overall time.

ii) Operator Control

Since it will be necessary for the operator to know that one program has finished running etc., while the other is still running, a 1407 console would be extremely useful for the logging of messages to the operator. If no 1407 is available it might be necessary to insert a programmed halt so that the operator can load another program if desired.

iii) Handling Time

It should be quite possible to arrange that nearly all setting-up for individual programs is carried out while another program is running. Similarly in the case of card wrecks, tape errors, card input preparation errors, etc., it would be simple to arrange, by sense switch control, for the other program to continue independently until the trouble had been corrected.

iv) Common Routines

Several sections of programming may well be required to be common to two or several programs - obvious examples are tape and disk error routines. In this case duplication would be avoided by making the routines part of the control program. Thus they could be assembled with the printer program, and referred to by machine location in the process programs.

If any of the process programs was to be run separately then the common routines would have to be inserted in the extra control package placed at the start of the program (If there was a number of process programs which might be run separately then one standard control package could be kept, which would be placed in front of the programs only if they were to be run independently of the printer program).

v) Available tape input/output time during a print cycle

During 98 ms, it is possible to read a record of some 3,500 characters from a 729 II at 556 characters/inch. This simple calculation is just included to show that if the timing of tape instructions in the process program is made correctly, quite considerable amounts of tape (or disk) input/output can be carried out with no great loss of printer time. In fact, of course, the full 98 ms would never be available, and the available tape time must be reduced accordingly.

V SUMMARY

It is first of al important to realise the limitations of this technique. In terms of hardware, a 1401 with Print Storage and Advanced Programming is essential, and a tape system is a realistic minimum. In terms of the programs themselves, the programs to be run simultaneously must be able to be included in core at the same time, and must use independent input/output units.

So much is essential, but it is also desirable that the printer program should be heavily 'printer-bound', and that the process program should be process bound.

Within the range of these restrictions, however, it is quite possible that some kind of simultaneous programming on these lines would be extremely useful, especially when so many computer manufacturers are stressing the parallel programming facilities of their products. With the growth of interest in SPOOL techniques on the 1410 (where this kind of simultaneous operation is an extremely realistic possibility), it is likely that systems designers will be more prepared to accept this kind of development for the 1401. Further, since the Overlap feature is available on the 1401, but not the Priority Interrupt feature as on 1410, efficient use of Overlap would need this kind of 'test-for-busy' techniques.

A particular case for the use of these techniques might be the 1401 off-line to a larger computer system, which might well have some long tape-to-printer runs ideally suited for this purpose. Even in the ordinary commercial 1401 installation, however, simultaneous programming could well be more than just a gimmick, and is a possibility that all Systems Engineers should at least be aware of.

APPENDIX.

Two Programs to illustrate the technique.

Details follow of two programs written to show use of the 'bridge' etc. The actual programs contain a minimum of instructions, and no input/output is used except the printer.

A. Printer Program.

```
(1)  *   PRINT      SBR     PRØCES + 7
(2)  *               B       MØVE
(3)  *   PRØCES     SBR     PRINT + 7
(4)  *   PR 2       B       PRINT + 4
(5)  *   PRINØN     DCW     #1
(6)  *   PRØCØN     DC      #1
(7)      FIELDA     DCW     @00000000000000000000@
(8)      MØVE       MLC     FIELDA, 250
(9)                 W
(10)                BSS     END, F
(11) *              B       PRØCES
(12) *              BSS     STEPY, D
(13)                B       MØVE
(14) *   END        BW      STEPX, PRØCØN
(15)                H       *-3
(16) *   STEPX      MLC     +PR 2, PRINT + 7
(17) *              CW      PRINØN
(18) *              B       PR 2
(19) *   STEPY      SBR     PRINT + 7
(20) *              H       *-3
(21)                NØP
(22)                END     MØVE
```

Notes on Printer Program.

1. The program simply prints 20 zeros.

2. Steps without an asterisk are the basic program. Steps with an asterisk are those which must be added so that this program can be run with process programs.

3. End-of-job is simulated by turning on Sense Switch F.

4. If Sense Switch D is turned on, the program will store the return address in the bridge, and then halt. This is to allow loading of the process program while the printer program is already loaded.

5. The PRINØN DCW is to show to the end-of-job routine of any process program whether the printer program is running or not.

6. The PRØCØN DC is to show the printer program whether the process program is running.

B. Process Program.

```
(1)  *                ØRG       501
(2)  *    START       SW        350
(3)       STEPB       BSS       STEPC, E
(4)                   A         AFIELD, 370
(5)                   BAV       STEPA
(6)                   A         AFIELD, 370
(7)                   BAV       STEPA
(8)  *                BPB       STEPB
(9)  *                BPCB      STEPB
(10) *                B         333
(11)                  B         STEPB
(12)      STEPA       A         AFIELD, CFIELD
(13)                  BAV       STEPC
(14)                  B         STEPB
(15) *    STEPC       BW        STEPD, 349
(16) *                H         *-3
(17) *    STEPD       MLC       @337@, 348
(18) *                CW        350
(19) *                B         337
(20)      AFIELD      DCW       @00000000000000000001@
(21)      CFIELD      DCW       #20
(22)                  END       START
```

Notes on Process Program.

1. The program goes round a loop, adding two 20 position fields twice on each loop. (In fact, the program adds one to the field being printed by the printer program, so that the interaction of the two programs can be seen).

2. Steps without an asterisk are the basic program. Steps with an asterisk are those which must be added so that the program can be run with the printer program.

3. End of job is simulated by turning on sense switch E.

Interaction of the two programs.

1. The printer program should be loaded in the normal way, and will start printing zeros.

2. When sense-switch D is turned on, the process program can be loaded (without its clear storage cards). Both programs will run together, and the number being printed will start increasing. (In fact, the number is increased by about 136 each time, i.e. 68 times round the loop, i.e. about 97 m.s. of processing.

3. The printer would be used more efficiently if the branch to PRØCES came between steps (8) and (9) of the printer program.

4. If sense switch E is turned on, the process program finishes, and a constant number will print. If the printer program is again stopped with sense switch D, the process program can be loaded again, and the two will run together.

5. If it had been desirable to have the facility of temporarily stopping either program while the other was running, this could be arranged by further use of sense switches.

6. If sense switch F is turned on when both programs are running, the printer program will finish, and the process program will continue processing unitl sense switch E is turned on when end of job halt is reached.