Griffith

COMPANY CONFIDENTIAL

PROJECT STRETCH
LINK COMPUTER MEMO NO. 15

October 25, 1956

Subject: A Break-In System for Link

By: F. P. Brooks

Objective:

A fundamental concept of the STRETCH System is that of several digital data processing units working simultaneously with data in a single memory. The Exchange, the Link, and the Senior Computer are expected to be in operation at the same time. Aside from the control of memory accesses, which must be treated elsewhere, there are problems of controlling the starting and stopping of each unit and the shifting of a single unit from one program to another. The ability of a single unit, such as the Link Computer, to respond at any time to signals that are pertinent to each of several programs and to perform the proper program for each signal will be known as a break-in system. Such a system not only involves a method of transferring control from one program to another upon receipt of a suitable signal, but also includes supervisory operation of one program upon another, a mode of operation in which the actual instructions of two programs are performed in an interlaced manner by the same computer under the control of one of the programs.

There are two quite distinct purposes for which a break-in system is necessary in the Link. The first of these is to provide a means by which the computer can make very rapid response to extra-program circumstances which occur at arbitrary times, performing useful work while waiting for such circumstances. In the Link, these circumstances will most often be signals from the Link Exchange that some interrogation has begun or that an input operation is complete, or signals from the senior computer that some computation is complete. For efficiency, the Link must respond to these forthwith. This demands a system by which such signals cause a transfer to a special program.

The second purpose is to permit the computer to make rapid and facile selection of alternate instructions when program-activated indicators signal that special circumstances have occurred. It is clearly desirable to have such a system for floating point overflow and underflow conditions, for example, since the alternative is tedious and wasteful programmed testing at frequent intervals. These two purposes are quite distinct, and it would be possible to provide a system for handling each independently of that for handling the other problem. However, it appears that a single system will serve both purposes equally well, so both will be considered together hereafter.

The system of break-in operation adopted must obey several constraints. The most important of these is that programming of the operation must be straightforward, efficient, and as simple as the inherent conceptual complexities allow. Secondly, the special circuitry must be reasonably modest, for maintenance economy as well as low first cost. Thirdly, the computer must not be retarded (at least in its operation upon its baseload problems) by the break-in system, except when break-ins do in fact occur. Fourthly, for programming simplicity and conceptual cleanliness, the system for Link should have as much similarity to the break-in system for the senior computer as possible. Finally, since there is virtually no experience in the use of multiprogrammed systems, the break-in system should be as flexible as possible. It is particularly important to avoid inflexibilities whose suitability depends upon certain postulated methods of use.

In the next section the system proposed will be briefly outlined. Succeeding sections will consider the several facets of the system individually, examining the alternative solutions to the problems of each, and offering some justification for the choices made. Final sections will show examples of the use of the system and will discuss the problems of extending such a system to the senior computer.

The Proposed System

The break-in system proposed is illustrated in Figure 1. It consists of two 20-bit instruction counters whose outputs are connected to the instruction address decoder through or circuits. Selection of one or the other to read-out and increment is governed by the Designator Flip-Flop, labeled F. The input-output indicators, (or other indicators that summarize the condition of the I/O indicators) are grouped together with overflow, etc. indicators and programmable indicators into one Indicator (formerly Selector) word which is contained in an Indicator Word Register in the central logical unit. The parallel output of this register is connected bitwise to and circuits with the output of the Mask Word register, a full sixty-four bit register whose contents are set by the program. When the result word at the end of any instruction execution is other than zero, the bit address of the left-most one is registered in the six-bit Left-One Counter, and a triggering pulse attempts to switch the Designator Flip-Flop. The path to F is controlled, however, by a Gating Flip-Flop, G, which is set whenever the Mask Word is written, and which, therefore, permits only one trigger pulse to pass for each distinct Mask Word setting.

In use, the programmer sets the Mask Word to permit break-in upon the desired conditions, and he sets Instruction Counter 1 with the address of the first instruction of the program that breaks in. He begins his program using Instruction Counter 0. In case of break-in, the supervisory program addressed by I. C. 1 begins by testing the Left-One Counter to determine the condition signalled, resets the mask and Instruction Counter 0 if a higher level break-in is to be permitted, stores the register contents if necessary, and begins the appropriate special program.

At the conclusion of the program, it restores the registers, Instruction Counter 0, the mask itself, and switches control back to Instruction Counter 0, which can begin exactly where it left off.

Special instructions permit a supervisory program, under control of I. C. 1 for example, to execute instructions designated by I. C. 0 and increment it without returning control to I. C. 0. In this fashion, tracing and debugging control routines can be handled in a simple and rapid manner. Examples of this will be given in the section on method of use.

The Left-One Counter will almost always be used to select the proper break-in routine. Since it is six bits long, it can readily be used as an index register for address modification. This use is facilitated if the L. O. C. is considered and wired into the machine as a twelve bit register whose lowest six bits are always zero. Thus, one-bit differences in the contents of the L. O. C. automatically select routines that are displaced by sixty-four words.

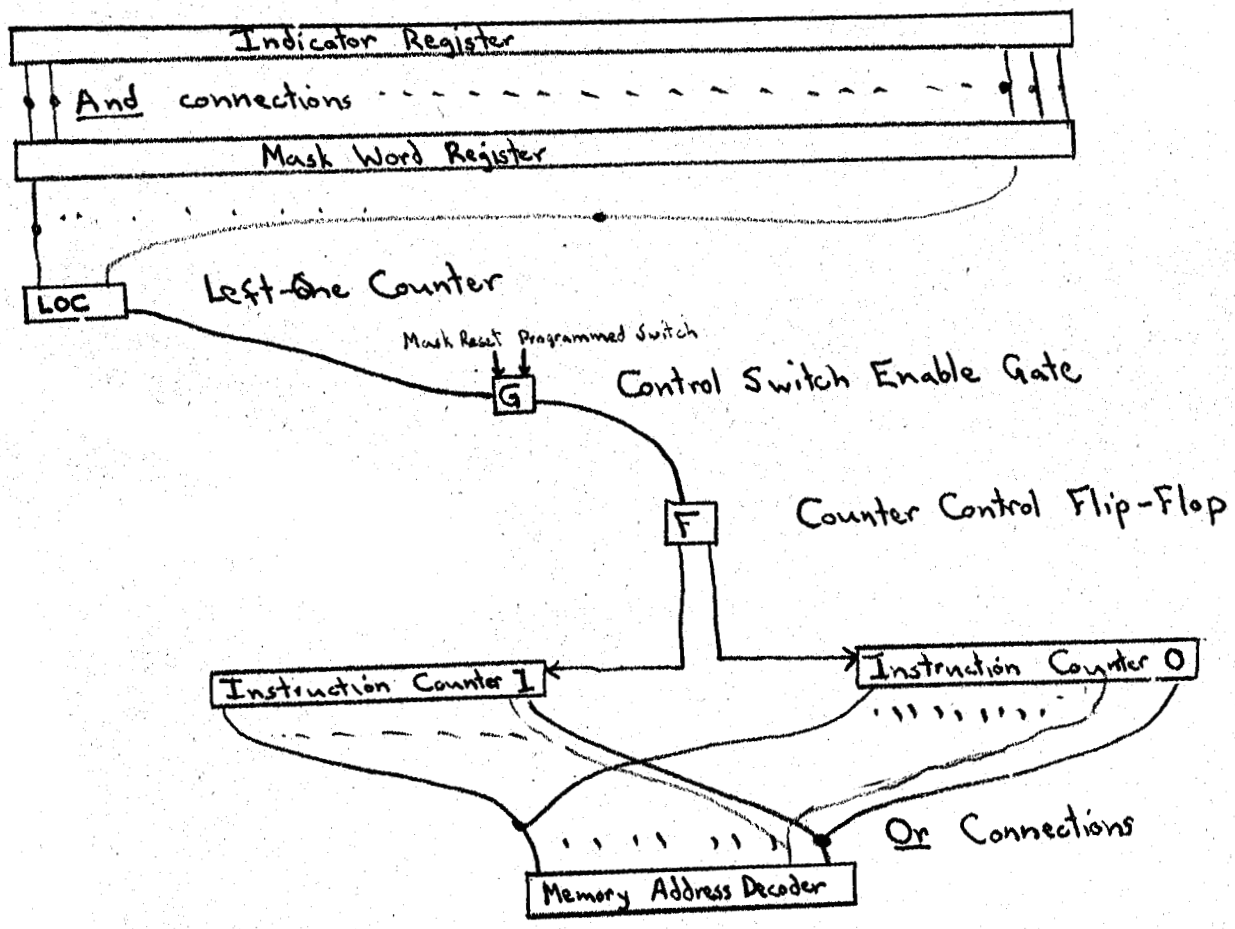

FIGURE I. Data Flow of Break-in System

Instruction Counter Configuration

Ideally, a break-in system would have as many independent instruction counters as there are programs among which the machine is to be time-shared. Since in the most general case, this is a very large number, and in the smallest well-defined case for the Link this is at least four, the ideal system would have prohibitive cost. Not only would the equipment for a four-counter system be excessive, but the complexities of the rules and arrangements for transferring control from each to the others would add programming complication and inflexibility. It is, therefore, worthwhile to examine the possibilities of using fewer true instruction counters in a fashion that serves the purpose of many.

One scheme is to use no central instruction counter at all, but rather to replace it with a register that designates one of the memory locations which serves as an instruction counter. So far as equipment is concerned, this is not significantly different from the single instruction counter, but it does permit greater programming flexibility. Break-in must be accomplished with a reset of the counter address register to zero or some other specified location upon signal, a form of trapping mode. The no-instruction counter system requires an additional memory reference for each instruction executed, increasing the time by something between thirty and fifty per cent. Since this philosophy can hardly be carried over to the senior machine, and since the memory accesses required would seriously handicap the Link, it seems undesirable.

A single internal instruction counter could be used in a manner very similar to the operation of the 704 with the Trap on Signal feature. In this system, the central counter is automatically reset to some specified value whenever the pertinent external signal appears. There can be some choice as to whether the reset shall occur - the programmer may place the machine in the "trapping mode" or remove it therefrom. For convenient multi-programming, the contents of the instruction counter must be automatically stored in some fixed place, such as an index register, so that the interrupted program can be taken up where it was abandoned. It is possible to provide that the counter resets to any one of many prescribed values, depending upon the nature of the external signal that caused the break-in. This is the system proposed for the 704 Model 3. The several fixed locations reduce the flexibility of the system. If one instruction counter is used, it is probably better to switch to a single location for all break-ins from which a supervisory program can direct the computer to the routine suitable for the external condition signalled. Since supervisory program operation appears to be very important in the STRETCH system, if it can be greatly facilitated by the selection of a suitable instruction counter configuration, this is an important consideration. With a single instruction counter, operation of a program under supervisory program control is considerably complicated by the necessity of preventing the subject program from seizing control. For example, when tracing one must analyze operations to insure that no transfers in the traced program are actually executed.

The proposed system uses two central instruction counters, whose outputs are or-d to the instruction address decoder. A flip-flop determines which instruction counter is to be read at any given time. Break-in then involves the setting and changing of the flip-flop under control of the external signals. By the use of a true flip-flop, it is possible to arrange for either to break in on the other, as the programmer might desire. Two counters can thus be used in the same manner as a multitude of true counters, by filling the unused one with the control data for the next higher order break-in at the time of the lower order break-in.

Equipment-wise, the two counter scheme requires one twenty-bit register more than the single counter scheme, and the controls appear to be somewhat simpler and more flexible. Supervisory programming is considerably facilitated with two true counters, since the supervisory program can alter the contents not only of its own counter, but also of that of the subject program, without handing control over to the subject program. This ability to modify the contents of two instruction counters at the same time appears to have significant advantages in supervisory programming. It, therefore, appears wise to base the development of the break-in system for Link on two true instruction counters.

Because two counters appear to suffice, it is unnecessary to explore higher numbers, which introduce complexities of control as well as additional equipment. It is quite possible to arrange two physical counters so that they automatically appear to act as a multitude of counters, but this can be done only at a considerable cost in programming flexibility.

The Control Transfer System

For setting and changing the flip-flop that determines the active instruction counter, a control system similar to that proposed by Mr. D. W. Sweeney in STRETCH Memo No.43 is suggested. Let all the external signal indicators and the programmable indicators be arranged in such a manner that they can be read in parallel as if they were bits of a single word. Although there may be more than 64 indicators, some summary indicators may be used. Then consider this to be connected in parallel to a mask word stored in the logical unit in a suitable register. This is to have an address and instructions so that it can easily be set to any value by the programmer. The parallel logical product of the indicators and the mask is tested, and the presence of any one in the result word causes the counter control flip-flop to reverse. A six bit counter whose contents are available to the programmer indicates the left-most bit in the result word.

This system has problems, of which the greatest is stability. It is important that after a break-in has been signalled, there be no further break-ins for any reason until the program has had time to respond properly. This is provided by a gate such that the triggering pulse cannot pass to the flip-flop but once for each mask setting.

Thus, after a break-in, the mask must be refilled by the program to permit a new break-in. This enabling of the trigger by the program also simplifies programmed arrangement of break-in priority.

Although the presence of any one in the control word is sufficient as a signal for break-in, the actual bits in the control word are necessary for identification of the condition causing break-in. The availability of the control word to the programmer would permit him to use it for address computation in the break-in program. With equipment that identifies, by six bits, the left-most one bit in the control word, programming is considerably simplified. Furthermore, this six-bit register eliminates the need for preserving the result word in a sixty-four bit register at all. Likewise, the result word does not need to be reset after break-in. The resetting of the automatic error indicators presents a much more difficult problem, since it is desirable to reset those that have been tested and not the others, and the test depends upon the programmed mask. One solution is to have the Left-One Counter automatically reset the indicator it identifies whenever the triggering impulse causes the flip-flop to transfer.

There are several alternative systems for control transfer. In a fully automatic system with two or more instruction counters, a distributor could transfer control every n microseconds or every n operations. The difficulties of preserving the contents of the central registers are immense with such a system. A second scheme would cause a transfer whenever the input data for a given program are reported unready upon interrogation of an Exchange status. This is somewhat too limited in utility and inflexible to be considered the whole answer. A third scheme, which presents some advantages in register storage, would have control transfer only after the storage operation that occurs first after the break-in signal. Other methods would depend upon the programmer to transfer control to the supervisory program at frequent intervals or to test the indicators and transfer to the supervisory program when they are on. The last two have very strong arguments of programmer inconvenience and time loss against them, and all three fail to meet the pressing need in the case of some indicators such as floating point overflows - the need to interrupt the action immediately, not several steps later.

Since immediate interruption is necessary for some indicators, it appears reasonable to provide a complete system that permits immediate interruption for any signal, but over which the programmer exercises full control. The flexibility of full programmer control appears to favor the programmed mask system of control transfer.

Data Preservation Upon Break-In

One of the most difficult problems in any break-in system is the provision of an adequate and convenient means of protecting the data of the interrupted program which may be in the central machine registers.

Such preservation is unquestionably necessary, and it needs to be performed with considerable facility if break-in is to be performed often.

There are three fundamentally different approaches to this problem. The first requires the programmer to signal or mark those places in the program where there is no data in the central registers that is to be used again. It is not in general sufficient that the data also exist in memory, for the program may be written in such a manner that it uses the data from the registers. This approach is perhaps the simplest, but it places the whole burden of data protection upon the programmer of the interrupted program. More important, this system does not permit immediate interruption of the program at any point when a signal is given, and this is clearly desirable in many important cases.

The second approach is to provide an automatic means by which the register contents are preserved upon break-in. The simplest of such schemes is the use of the three-address instruction so that no data is ever used from a central register, so that "immediate" break-in after each instruction is possible. For some operations, however, it is desired to break-in before the result is stored. The most important objection is that the scheme is wasteful of instruction storage space, execution time, and memory accesses. A most elaborate automatic storage scheme would require as many sets of central registers as there are break-in levels, with automatic switching between them. This is impractical and unjustified. The compromise automatic storage scheme would have each program assigned a block of memory registers. Upon break-in, automatic circuitry would consult a register that identifies the program interrupted and would cause all the registers to be serially stored in the appropriate memory block. This scheme is quite practical, but the necessity for fixed assignments of the memory blocks for each program gives programming inflexibility, and the equipment needed for the fully automatic storing hardly seems justified.

The third approach is to require the entering program to store the data of the interrupted program. This is favored by the fact that the entering program almost always begins at a fixed point, while the interrupted program has in general, been broken into at an arbitrary point. It is, therefore, feasible to preface the working instructions of the entering program with the instructions necessary to store the registers. This provides full flexibility as to the location in which the data shall be stored, and the entering program can always identify the interrupted program by the contents of its instruction counter.

To facilitate this storage, a few powerful instructions would be valuable, such as Fill X and Dump X, which would fill the central registers from a set of consecutive memory words beginning with X or empty them into the block.

It is not unreasonable to constrain X to be some address divisible by 4, 8, or 16 depending upon the number of registers to be stored, and the equipment might be very simple. Furthermore, in a large memory system with consecutive addresses distributed among memory boxes, the Fill and Dump instructions could be executed very rapidly. If a system of block addressing is used rather than full 20-bit memory addressing, the storage could be effected very simply without any program identification by performing it before the entering program changes the block tag. With the special instructions the time and programming costs of requiring the entering program to protect the interrupted program's data do not appear to be great. The system demands a minimum of equipment. Furthermore, it should show actual time savings over a fully automatic storage system because of the very common case where only an overflow, underflow, etc. adjustment is to be made and the register data need not be stored and retrieved at all.

Priority of Break-In

Another of the problems of an automatic break-in system is establishing the priority of break-in and the conditions under which a condition can interrupt a program initiated by a condition of lower, the same, or higher priority.

There are at least four identifiable levels of priority for the LINK. In increasing order, these are the operating or base load program, correction procedures for program-caused errors (such as exponent overflows), supervisory programs for input-output, and supervisory programs for machine failure conditions.

The problem of inter-level break-ins can be handled by permitting any condition to cause break-in on any program of a lower level. Alternatively, the system could be established so that break-ins were permitted only on the lowest-level program. In this method of operation, the second-level overflow correction routine, for example, would have to finish and return control to the base load problem before the supervisory program for input-output could interrupt. A third alternative is to allow programmed control so that the number of levels and the priorities and rules for inter-level break-in can be established by the programmer. The flexibility of this method urges its use, if it can be provided with reasonable equipment, and if it can be made convenient and easy to use.

Another problem is that of intra-level break-in. Shall an input-output signal (third level), for example, be permitted to cause break-in on another input-output routine? The alternatives are nearly the same as those of inter-level break-in. Fully automatic control might be provided so that such break-ins never occur.

Alternatively, they might be always permitted to occur. This leads to the possibility of instability, with control of the computer oscillating between or among several programs. Another possibility is programmed control, so that some intra-level break-ins may be permitted at some times with complete flexibility. Full programmed control with a supervisory program making all break-in decisions, is one method of gaining the desired flexibility, but such a method of operating appears to be somewhat ungainly to use and wasteful of machine time.

The system proposed is a compromise system, with some special control equipment but with full programmed flexibility. Since the mask must be reset after each break-in before another can occur, the programmer may reset it immediately with a new value to permit any break-ins desired during the operation of the new program, or he may defer its resetting, thus excluding break-in until he is ready to return to the lower level program. If a break-in on a higher level program is to be permitted, the programmer also resets the other instruction counter to the address of the first instruction of the break-in program that might occur. Both of these modes of operation will be illustrated with examples in the next section.

Method of Use of the System

Several new basic instructions will facilitate manipulation of the break-in system. The instruction counter under whose designation the operation is executed, will be called the controlling counter, and the other will be called the other counter.

1. LDC Y Load Other Counter Y

   Resets the other Instruction Counter to the value specified by Y.

2. STC Y Store Other Counter Y

   Stores the contents of the other Instruction Counter in the location specified by Y.

3. ADV Y Advance Other Counter Y

   Causes the other Instruction Counter to advance by quantity specified by Y.

4. CMC Y Compare Other Counter Y.

   Compares the contents of the other Instruction Counter with those designated by Y, and causes control to switch to the other Instruction Counter if its contents are equal to or greater than the Y quantity.

5. LDM Y Load Mask Y

   Resets the Mask Word to the quantity specified by Y and enables break-in gate G.

6. LMD Y Load Mask Y with Delayed Enable.

   Same as above, but G is not enabled until a programmed Instruction Counter switch occurs.

7. STM Y Store Mask Y

   Stores the Mask Word in location specified by Y.

8. TRS Y Transfer and Switch Y.

   Resets the Instruction Counter in control to the value specified by Y and transfers control to the other Instruction Counter.

9. EXC Y, n. Causes the machine to execute the n instructions addressed by the other counter, where n is specified by Y.

EXC must be tagged, and its first tag designates the index register to be used for the n counter. Control is retained in the Instruction Counter originally in control. Any transfer instructions executed alter the contents of the other counter rather than those of the controlling counter, and both counters are incremented. Any of the above instructions are not executed, but instead an indicator is set.

10. DMP Y Dump into Y. Stores to contents of all the registers in a block of n memory words beginning with Y.

    Y must $\equiv$ 0 mod 8.

11. FIL Y Fill from Y. Fills the registers from a block of n memory words beginning with Y.

    Y $\equiv$ 0 mod 8.

12. DMPO Y Dump, including other I. C., into Y. Same as DMP Y, except the contents of the other Instruction Counter are stored in the n + 1st word of the block.

13. FILO Y Fill, including other I. C., from Y. Same as FIL Y, except that the other Instruction Counter is filled with the contents of the n + 1st word of the block.

The instructions for loading and storing the contents of the instruction counters are necessary in any break-in system. Those for advancing and testing an instruction counter are not demanded by any of the examples yet worked, but they appear useful. The two instructions for loading mask words differ as to whether the G flip-flop is immediately enabled or not. The delayed enable is necessary to prevent unwanted flipping when the mask is reset immediately prior to a return to the base load program.

Transfer and Switch facilitates the termination of a break-in routine by permitting a simultaneous resetting of the operating instruction counter and a return of control to the other.

The Execute instruction appears to have great usefulness in supervisory routines, since it permits the execution of parts of a program, including transfers, without permitting the executed program to seize control. It is necessary, of course, to exclude the execution of instructions that cause switching of the instruction counters or change the Mask Word that controls switching. For this reason, the instructions of this sort are not performed by the EXC instruction, but an indicator is set on instead. A much more satisfactory, but much more costly solution, has a separate mask word for each of the two counters, and such instructions alter the one specified.

Execute operates by finding n from Y, which may be an immediate, direct, or indirect address, and which may be indexed. n is entered into the memory word designated by the first tag, which must always be present. At each operation of the other Instruction Counter, the index register containing n is decremented, and its reaching 0 signals completion of the EXC operation.

Example 1. Simple Break-In

When an exponent overflow occurs, it is desired to interrupt immediately and to reset the accumulator to the largest possible number. Exponent overflow is signaled by the nth indicator in the Indicator Word. No break-ins are to be permitted during the fix-up routine. The notation used is that of 704 instructions except for the special instructions.

Instruction Counter 1	Instruction Counter 0		Remarks
	x	LDQ	
	x+1	FMP	Overflow occurs, causing break-in. Control switches to I. C. 1.
y	TRA Z, LOC		Selects proper fix-up routine by automatic indexing from the LOC.
z-	CLA Lpn		
z+1	LMD Mask 0		Resets Mask Word so that break-in will be again permitted after the return to I. C. 0.
z+2	TRS y		Resets I. C. 1 to y and switches to I. C. 0.
	x+2	STO	Original program continues where interrupted.

LOC Left-One Counter
Lpn Address of largest possible number
Mask 0 Mask word defining conditions under which break-in upon the x-program is to be permitted.

This example illustrates the simplest type of break-in, and shows the facility with which such simple break-ins may be performed of a twenty-bit address. Thus, at y in the example, to which control is switched immediately upon the overflow at X 1, the supervisory program provides only a transfer to a location that is indexed by the contents of the Left-One Counter. This takes the program to z, where the overflow routine lies. The largest possible number is inserted into the accumulator, and the break-in program begins to set up the return. First the mask is reset to its former value, with the delayed enable so that when control is returned to Instruction Counter 0, further break-ins may occur, and only then. Finally, the TRS instruction resets Instruction Counter 1 to the beginning of the break-in supervisory, and switches control to Instruction Counter 0.

Example 2. Multi-level Break-In

A more sophisticated break-in routine is to be defined as follows: Upon signal from the Link Exchange, the operating program is to be interrupted and the contents of the registers stored. During the input-output routine, break-in by certain machine error routines is to be permitted. In event of this higher order break-in, the register contents are to be stored until the machine error routine is complete. Then the input-output routine is to continue until it is complete, whereupon the original program is to begin where it was interrupted. This is accomplished with seven special instructions in the input-output routine.

Instruction Counter 1	Instruction Counter 2	Remarks
	x LDJ	Mask 1 is in Mask Word, defining conditions for break-in on x-program.
	x+1 FMP	
	x+2 FAD	External signal from exchange received during this instruction causes break-in to I. C. 1.
y TRA Z, LOC		
z DMPO P1		Dumps register contents in specified place.
z+1 LDC y		I. C. 0 set to address for higher level break-in.
z+2 LDM Mask 2		Mask Word set to define conditions for interruption of the input-output routine. This setting enables G.
z-3 begin input-output routine • • •		Break-in may occur during routine but return is to point of interruption.
w LMD Mask 1		At end of I/O routine, prepares to return. Mask reset to original value, but enable delayed.
w+1 FILO P2		Registers and I. C. 0 reset as they were originally.
w+2 TRS y	x+3 STO • •	Original program continues where it left off.

The second example shows the versatility, power, and economy of the system on complex break-in routines. In general, any break-in routine that may itself be interrupted, requires seven housekeeping instructions, including two for data protection, and any break-in routine that may not be interrupted requires three or five housekeeping instructions (depending on the need to store the registers).

Example 3. Supervisory Program - A General Tracing Routine

The third example illustrates the use of the two-instruction counter system for supervisory programs. In this case, it is desired to trace a program, storing results of each step for printing.

Instruction Counter 1	Instruction Counter 0	-	Remarks
y	LDC x		Set up I. C. 0 for tracing
y+1	EXC 1, k		Execute x and advance I. C. 0
y+2	DMPO P1, j		Store data for print
y+3	TIX y-1, j		Repeat

Subsequent cycles of the loop would cause the execution of x+1, x+2, etc., and any transfers in the x-program would only change the contents of I. C. 0.

Although the routine shown is the most unsophisticated of general tracing routines, its economy is significant, since other than printing, it requires only four instruction executions for each instruction traced, instead of the fifteen to thirty required on present day machines.

Clearly, a supervisory program for testing effective addresses of a new program is just as simple so far as the logical control is concerned. Such a program appears an essential part of any on-machine program debugging system.

Example 4. A Specialized Tracing Routine

A more powerful tracing routine illustrates some of the flexibility accorded to supervisory programs by the two-instruction counter system. The routine illustrates the use of special tracing program designed for the program being traced. There is no conceptual difficulty in imagining the construction of such programs by automatic coding techniques. Other special tracing routines might call for printing data only after operations of certain types or after executions of operations affecting certain effective addresses.

Instruction Counter 1	Instruction Counter 0	Remarks
y LDC x		
y+1 EXC 4, k	x LDQ ,j	
	x+1 MPY ,j	
	x+2 ADD ,,j	
	x+3 STO ,j	
y+2 DMPO P1, i		Store for printing
.		
. any logical tests on conditions		
.		
y+8 EXC 3, k		
	x+4 CLA ,j	
	x+5 ADD ,j	
	x+6 SUB ,j	
y+9 DMPO P2, i		
y+10 EXC 2, k		
	x+7 STO ,j	
	x+8 TIX x,j	Resets I. C. 0
y+11 EXC 32, k		Single instruction completes next four iterations without detailed testing.

This example illustrates a method by which a tracing program might be automatically arranged to trace the first (or any other) cycle of a loop in detail and execute the remaining cycles without tracing. It also illustrates the methods by which tracing programs might be made much more efficient by tailoring so that they print only the data that is apt to be revealing to the programmer.

In general, it is impossible to permit break-in in programs that are executed subject to the immediate control of a supervisory program. The Execute instruction provides, however, that instructions that cause control switching or alter the conditions under which control might be switched set indicators, and programmed reference to these indicators may serve the same purpose. In general, break-in can be reasonably excluded from programs that are undergoing debugging.

Example 5. Distributor Action Time Sharing

It is conceivable that some programmer might desire to time share the machine among two autonomous parts of his program in some arbitrary fashion. Example 5. admittedly not a routine that is likely to be of any great use, illustrates the method by which the break-in system can simulate the action of a distributor in assigning machine time to each of several programs in turn.

Instruction Counter 1	Instruction Counter 0	Remarks
z FILO Place 1		Set up for first program
z+1 EXC 100, k	X	Begin part of first program
	x+1	
	x+2	
	•	
	•	
	x+99	
z+2 DMPO P1		Store data of first program.
z+3 FILO P2		
z+4 EXC 107, k		
	y	
	y+1	
	•	
	•	
	y+106	
z+5 DMPO P2		Store data of second program.
z+6 TIX z		Repeat alternation of programs.