July 15, 1959

THE EXECUTE OPERATIONS-A FOURTH MODE OF INSTRUCTION SEQUENCING

F. P. Brooks, Jr.

IBM

RESEARCH CENTER

INTERNATIONAL BUSINESS MACHINES CORPORATION
YORKTOWN HEIGHTS, NEW YORK

RC-213

THE EXECUTE OPERATIONS - A FOURTH MODE OF INSTRUCTION SEQUENCING

by

F. P. Brooks, Jr.

International Business Machines Corporation
Research Center
Yorktown Heights, New York

ABSTRACT: Instruction sequencing modes include normal sequencing, branching, interruption, and use of Execute operations whereby an addressed instruction is executed out of its normal sequence. Execute operations essentially serve as calling sequences for one-instruction subroutines, and this property suggests a variety of applications. The two Execute operations provided in the Stretch computer are described.

Research Report
RC-213
July 15, 1959

Printed in U. S. A.

INSTRUCTION SEQUENCING MODES

Classically, digital computers have had two basic modes of sequencing instructions. In the first, normal sequencing, each instruction has a unique successor which may be defined by an instruction counter or by a next instruction address within the instruction itself. The second mode is the selection of an alternate sequence by a branching, skipping, or suppression operation.

A third mode of sequencing, program interruption, has been recognized more recently,[1] although computers as early as UNIVAC I contained rudimentary interruption provisions. In this mode, execution of a sequence of instructions may be interrupted at an arbitrary point, and specification of the next instruction in this case is completely independent of the last instruction executed. Provision may or may not be made for saving the current location in the interrupted sequence.

In branching, the selection of an alternate instruction implies the selection of a new sequence -- the alternate instruction specifies or implies (through an instruction counter) its own successor. The same may be true of interruption; or, an interruption system may be defined so that the interrupting instruction does not change the instruction counter. In this case, the interrupting instruction may, if a branch,

1. Jules Mersel, "Program Interruption on the Univac Scientific Computer," Proceedings of the WJCC, p. 52 (1956).

specify its own successor, but if it does not, the successor is implied by the last instruction of the interrupted sequence.[2]

It is often desirable for an instruction sequence to execute a single, non-interruption instruction which does not specify or imply its own successor. For this purpose the Execute operations have been independently developed by several groups. These may be considered a fourth mode of instruction sequencing.

The four modes can be summarized briefly by stating the four possible relationships between an original sequence A and a second sequence B:

Normal sequencing	A <u>keeps</u> control
Branching	A <u>gives</u> control to B
Interruption	B <u>takes</u> control from A
Executing	A <u>lends</u> control to B

In an Execute operation, the address specifies, directly or indirectly, an instruction to be executed. When the object instruction has been completed, the instruction next in order after the Execute instruction itself is performed. Thus, the location of the object instruction <u>does not imply the location</u> of its successor. This is equivalent to

2. F.P. Brooks, "A Program-Controlled Program Interruption System," <u>Proceedings of the EJCC</u>, p. 128 (1957).

indirect addressing except that the whole instruction, not just the address, is selected from the specified location.

It may be further desired to prevent the object instruction from specifying its successor explicitly -- i.e., from changing the instruction counter by branching. The Execute operation developed for the IBM Stretch computer has this property. The Execute operation later but independently developed for the IBM 709 computer does not.[3] Neither does the Execute operation provided in the recently described Soviet LEM-1 computer.[4] The branch-inhibiting property is helpful for some uses of the Execute operations and is inconvenient for other uses.

APPLICATIONS OF EXECUTE OPERATIONS

The uses of the Execute operations arise directly from the fact that they do not imply their own successors. In effect, an Execute operation calls a one-instruction subroutine and specifies its return. In the LEM-1, for example, there are 1024 words of erasable storage and 7168 words of read-only storage. The Execute operations permit programs in the read-only storage to use modifiable instructions in the regular storage.

3. Reference Manual, IBM 709 Data Processing System, p. 37 (1958).
4. Yu. A. Makhmudov, Radioteknika, No. 3; pp. 44-57 (March, 1959); in English, Communications of ACM, Vol. 2, No. 10 (Oct, 1959), p. 3.

In machines with explicit instruction addresses for normal sequencing, such as the IBM 650, many efficient programming techniques depend upon instructions which are developed and executed in the accumulator. These techniques have not heretofore been applicable to counter-sequenced computers, but become possible when such computers are provided with Execute operations and addressable arithmetic registers.

The provision of Execute operations in the IBM 709 simplifies modification of non-indexable and non-indirect-address operations such as those for input-output.

The one instruction subroutines provided by the Execute operations are especially useful in linkages between a main program and ordinary subroutines. For example, a subroutine may need several parameters such as character size, field length, index specification, et cetera. The calling sequence may include these parameters in actual machine instructions which the subroutine treats as second-order subroutines. This ability to lend control back and forth between calling sequence and subroutine, should permit many new techniques to be developed for subroutine linkages.

One useful special case of this form of subroutine technique occurs in interpretive routines where machine instructions can be intermixed with pseudo-instructions in the argument program. The interpreter can then execute these directly without transplanting them into itself.

For all of the foregoing purposes it is desirable for the Execute operation to have any machine instruction as its object. Thus, one may desire to execute an arithmetic instruction, a branching instruction, or even another Execute instruction. For program monitoring, however, the object instruction of an Execute operation should be prevented from changing the instruction counter which controls the monitoring routine.

Consider, for example, a supervisory program A, such as a tracing routine, which is to monitor the execution of an object program B, perhaps with testing or printing of the instructions of B as they are executed. With an ordinary set of operations, the programming to effect such monitoring is quite clumsy. Each instruction of B must be moved from its normal place in memory to a place in the sequence of A. Then it must be tested to insure that it is not a branching instruction, or if so, that the branching condition is not met, for the execution of such an operation would transfer control of machine from the supervisory program to some point within the object program. Finally, after the transplanted B instruction has been executed, A must update its pseudo-instruction-counter that keeps track of the progress of B, and repeat the whole process with the next B instruction. If the B instruction is a successful branch, A must appropriately revise the pseudo-instruction-counter. This programmed machinery is common to all monitoring routines, and must be executed in addition to the actual monitoring desired.

EXECUTE OPERATIONS IN THE IBM STRETCH COMPUTER

Two Execute operations with properties suited for this application are provided in the IBM Stretch computer. These are called EXECUTE and EXECUTE INDIRECT AND COUNT. Each causes a single instruction to be fetched from an addressed location and executed, except that execution may not change the instruction counter. If the object instruction specifies a branching operation (which would cause such a change), branching is suppressed, and an indicator is actuated which may interrupt the (monitoring) program.

In the EXECUTE operation, the address specifies the object instruction directly. In the EXECUTE INDIRECT AND COUNT operation, the address specifies a pseudo-instruction-counter, whose contents specify the object instruction. After the object instruction is performed, the pseudo-instruction-counter is incremented according to the length of the object instruction. This last feature is particularly convenient in a computer which has instructions of different length. Any Execute operation may have another Execute operation as its object. A special interruption signals the attempted execution of an overly long chain (several hundred operations) of Execute operations (or indirect addresses).

The Stretch Execute operations, then, provide not only the ability to execute an isolated instruction with an automatic return of control, to the monitoring routine, but also provide for (a) suppression of branching, and (b) signalling the monitoring routine when branching is attempted. These properties considerably simplify monitoring routines. The automatic return obviates the need for transplanting the instructions of the object program into the monitor. The suppression of branching ensures that the monitor can retain control without detailed testing of the object instruction. The notification of attempted branching permits the monitoring program to update the pseudo-instruction-counter for the object program without detailed testing. Since this detailed testing of the object instruction for branching and skips occupies a large part of conventional monitoring programs, the Execute operations make such programs more efficient. The EXECUTE INDIRECT AND COUNT operation gives further efficiency because it automatically increments the pseudo-instruction-counter.

A simple monitoring loop for performing a control trace in the Stretch computer reduces to:

Location	Operation	Address
	EXECUTE INDIRECT AND COUNT	pseudo-instruction-counter
	BRANCH	

When a branch occurs in the object program, this loop is interrupted, and a suitable routine records the tracing data and changes the pseudo-instruction-counter.

The suppression of branching, as mentioned above, slightly restricts the generality of the Execute operations in their non-monitoring uses. In effect, the object instruction which is a branch must be performed indirectly by an interpretive routine. In practice, this restriction is not onerous, and the additional applications of the safeguarded Execute operations appear to justify the provision of the safeguards.

The Execute operations can in theory be put into any stored program computer. Their mechanization is somewhat simpler and more justifiable in computers that use an instruction counter for normal sequencing. Provision of the safeguards that permit the operations to be used for monitoring is greatly simplified in computers which have program interruption systems. In other computers, attempts by the object program to change the sequence must be signalled by setting conditions that stop the machine or are tested by branching instructions.

An obvious extension of the Execute operations would be to have the EXECUTE INDIRECT AND COUNT operation automatically change the pseudo-instruction-counter when the object instruction is a branch. In this case there would still need to be an alarm to the monitoring program, however.

In summary, the Execute operations provide an effective means by which an instruction sequence can temporarily tend control of a computer to an instruction not in that sequence. Safeguards can be provided to guarantee the return of control to the original sequence; and, on balance, these safeguards make the operations more powerful. The Execute mode of instruction sequencing has many uses in subroutine linkages, in special programming devices, and in monitoring routines.