On Reversible Subroutines and Computers that Run Backwards

E. D. REILLY, JR. AND F. D. FEDERIGHI
General Electric Company, Schenectady, New York*

A computer design is described which permits subroutines to be executed backward as well as forward, either with their instructions unchanged or replaced with conjugate instructions. It is shown that using this concept a number of new subroutine types can be developed with rather unusual properties. Since these properties are analogous to certain matrix operations, a arallel nomenclature is suggested for their classification.

The purpose of this paper is to point out certain unusual properties of a digital computer that possesses both an "Execute" instruction that permits execution of an isolated instruction anywhere in memory and a "Repeat" instruction that allows repetition of one or more instructions a specified number of times with provision for some sort of automatic address modification. Several computers have one of these features, but the authors are not aware of any actual machine that has both, although a complete search has not been carried out.[1]

There is, however, a hypothetical computer called the MOHAC, defined for instructional use in high schools, that does possess both functions as an integral part of its architecture.[2] A MOHAC word consists of a sign and six decimal digits. The instruction format is

C I M

where C is a two-digit command, I is a single-digit increment field, and M is a three-digit address. The Repeat command would then read "Repeat the next I instructions M times," so that up to nine instructions can be repeated up to 999 times. Furthermore, for any instruction under Repeat influence, the I digit will be used to increment (or decrement) the apparent address, *after* each time of execution, according to the following convention:

Specified I Digit	*Resultant Increment*
0-5	0-5
6	−4
7	−3
8	−2
9	−1

For example, using TMA for "Transfer Memory to Accumulator" and TAM for "Transfer Accumulator to Memory," the program:

```
RPT 2 150
TMA 1 200
TAM 8 900
```

will copy the 150 words stored consecutively forward from location 200 and store them backwards in every other location beginning with 900 (900, 898, 896, ···). Analogous to actual computer Repeat instructions, address modification is not made to the home-site memory locations which contain the repeated instructions, but rather to special "repeat registers" to which copies of the instructions are transmitted upon encountering the Repeat command.

An example of the MOHAC "Execute" instruction would be

```
EXE 0 452
```

which is interpreted as follows: "Regardless of where the program is currently sequencing, execute whatever instruction is in memory location 452. After doing so, return to the next instruction in the normal sequence (unless the executed instruction was a Jump of some kind)." In the example given, the I digit (= 0) would be immaterial unless the EXE were under repeat influence. Now consider this example:

```
RPT 1 007
EXE 1 500
```

This program will cause the seven consecutive instructions at 500 to 506 to be executed before the program continues at whatever instruction follows the EXE. Such a program could be useful for patching several instructions into space where only two would normally fit, or, more importantly, could be compiled by an assembler in lieu of a seven instruction (or longer) macro that would ordi-

* Knolls Atomic Power Laboratory.

[1] The GE 625/635 has an Execute and a Repeat, but has no provision for modifying addresses of instructions under Repeat influence. Actually, however, it is the Execute that is the key instruction. Any machine having at least this function could achieve all of the effects described in this paper by using Execute in normally programmed loops; a Repeat mode merely allows an especially succinct representation with minimum effort.

[2] The MOHAC, or MOhawk-Hudson Automatic Computer, was designed as a project of the ACM Hudson Mohawk Chapter; a MOHAC simulator for the IBM 1620 is available from the authors. Results of a recent high school course in programming fundamentals, based on the MOHAC, are now being prepared.

narily be inserted as an in-line subroutine every time it is needed.

Now suppose that, because of some unusual program restriction, it is desired that a data list be interspersed with coding. Although this cannot ordinarily be done, it is readily accomplished in MOHAC by the program:

```
RPT  1  050
EXE  2  600
```

This will execute the fifty-instruction sequence in 600, 602, 604, $\cdots$, 698, leaving 601, 603, $\cdots$, 697 free for data.

There are many variations on this theme, but one is particularly intriguing. Consider:

```
RPT  1  100
EXE  9  700
```

Since an I digit of 9 is equivalent to an increment of -1, such a sequence will actually cause MOHAC to run backward! In the example given, the sequence will be 700, 699, 698, $\cdots$, 601. Once such a possibility has been identified, it is only natural to seek special cases with particular properties. For example, are there "palindromic" or symmetric subroutines which are the same whether run forward or backward? Given that CAC means "Clear ACcumulator," consider the following:

```
CAC
TAM  C
ADD  A
SUB  B
ADD  A
TAM  C
CAC
```

This is, indeed, a palindromic subroutine for performing C = 2A − B, but it is obviously contrived since it contains three more instructions than would normally be required on a one-address machine.

Of considerably more value than a palindromic subroutine would be one which did something different *and useful* in the reverse direction. In particular, it might be asked if there exist "reversible" subroutines which in some sense undo whatever function they perform in the forward direction. If we call a subroutine that undoes the work of another subroutine an "inverse subroutine," what we are asking is whether a transposed (reversed) subroutine is ever its own inverse. Although such *orthogonal* routines may and probably do exist, it is unlikely that one of any length to be of real value can be constructed. Meaningful subroutines tend to have a series of fetches concentrated near their beginning and one or more stores near their end; such structures are inherently irreversible.

TABLE 1. Instructions and Conjugates

TMQ	Transfer Memory to Q	TQM	Transfer Q to memory
SAL	Shift A Left	SAR	Shift A Right
ADD	Add Memory to A	SUB	Subtract Memory from A
MUL	Multiply Memory × Q	DIV	Divide Q by Memory
RDN	Read (Card) Numerically	PTN	Print Numerically
SSP	Set Sign of A +	SSN	Set Sign of A −
CAC	Clear A	CAC	Clear A
AQL	Shift (AQ Together) Left	AQR	Shift AQ Right

A is the accumulator and Q is the multiplier-quotient register.

At this point, we wish to "imagine," since it has not yet been added to MOHAC, an instruction similar to EXEcute which shall be called EXC, or "Execute Conjugate." The conjugate of an instruction is illustrated by listing a few pairs of representative MOHAC instructions which are conjugates of one another (see Table 1).

We are now in a position to define two new types of subroutines based on EXC comparable to the symmetric and orthogonal EXE type subroutines. The first of these would be one whose conjugate executed backwards would be the same as the original subroutine run forward. Such subroutines could well be called hermitian or self-adjoint because of their obvious similarity to matrices of like property. Although these are no more valuable than the palindromes, they are more plentiful. A trivial example is:

```
TMA  JOE  |    TAM  JOE  ↑
SAL    1  |    SAR    1  |
SAR    1  |    SAL    1  |
TAM  JOE  ↓    TMA  JOE  |
```

which, either forward or conjugately backward, will strip off the leading and trailing digits of the word at JOE.

Finally, we might reflect as to the likelihood that, using the conjugation concept, reversible subroutines could be constructed more easily. Again in analogy with matrices, we call a subroutine whose backward (transposed) conjugate is its own inverse a *unitary* subroutine. An example is:

```
TMQ  W    |    TQM  W    ↑
CAC       |    CAC       |
AQL  2    |    AQR  2    |
TAM  S    |    TMA  S    |
CAC       |    CAC       |
AQL  2    |    AQR  2    |
TAM  S+1  |    TMA  S+1  |
CAC       |    CAC       |
AQL  2    |    AQR  2    |
TAM  S+2  ↓    TMA  S+2  |
```

When run forward, this subroutine will separate the six-digit MOHAC word at W into three two-digit words stored at S, S+1 and S+2. When run conjugately backward, the routine will reassemble the three two-digit quantities into one six-digit word and store the result back in W.

Of all of these concepts, the only one at all likely to become more than a curiosity is that of unitary subroutine. Such routines (used in conjunction with EXC) would certainly save memory space, provided both the main routine and its inverse are required in the same program, as is often the case. If it is objected that the small amount of space saved would hardly be worth the effort of constructing unitary subroutines, it can only be stated that

Continued on page 578

there is already considerable research being expended on so-called "common" subroutines which can be entered by several different programs on a time-shared basis without causing chaos. The objective of using either unitary or common subroutines would be the same—to keep memory from becoming clogged with multiple copies of, or the inverses of, routines used frequently in a multiprogramming environment.

Summary. Using the concepts of EXEcute and EXecute Conjugate described in the note, the following subroutine definitions may be developed.

Palindromic Subroutine A subroutine whose instruction pattern is the same whether executed forward or backward.

Transposed Subroutine A subroutine formed from a given subroutine by executing its instructions in reverse order.

Symmetric Subroutine A palindromic subroutine.

Conjugate Subroutine A subroutine formed from a given subroutine by replacing each instruction with its conjugate.

Hermitian Subroutine A subroutine which is identical to its transposed conjugate.

Inverse Subroutine A subroutine which undoes the work of a given subroutine.

Singular Subroutine A subroutine which does not have an inverse.

Orthogonal Subroutine A subroutine whose transpose is its inverse.

Unitary Subroutine A subroutine whose conjugate transpose is its inverse.

Reversible Subroutine An orthogonal subroutine.

Self-Adjoint Subroutine A hermitian subroutine.

RECEIVED MARCH, 1965