A Self-Healing Control

By BRUCE E. BRILEY

(Manuscript received April 30, 1968)

This article describes an electronic computer self-repair technique which does not require system duplication. It identifies the properties demanded of control hardware to effect this end and considers their practicability. It also decribes a prototype computer constructed to test the feasibility of this form of self-repair, and discusses test results.

I. INTRODUCTION

In electronic computing machines, where a high degree of availability is a requisite, some form of self-repair is provided. The rational in most such machines is that a given piece of hardware must be backed up by an identical part which may be switched in upon failure of the first.

Work in nonreplicative self-repair has chiefly centered about the arithmetic unit and other controlled entities where the solutions, though not trivial, are relatively straightforward. The control portion, however, has consistently been avoided as unmanageable short of replication.[1] This paper describes a technique for what will be called "self-healing" the control of a machine which differs radically from conventional self-repair, and displays potential for considerable economic savings.

II. DEFINITIONS

Terms used in describing the technique and their definitions are:

Order denotes that portion of an instruction conventionally used to identify the instruction type, not including the address and other fields of the instruction, even when used for augmentation rather than addressing.

An order structure is called *closed* (closure under imitation) if to each order there corresponds at least one program of finite length using only a subset of the order structure diminished by the object

order such that the program imitates the salient actions of the object order in every detail except timing.

A control is called *failure autonomous* if the circuitry associated with the sequencing of an order is unique to it and does not propagate the effects of an internal failure beyond the associated circuitry's geographic bounds.

A control unit is called *self-diagnosing* if, under failure, (*i*) the control immediately ceases activity, (*ii*) the identity of the offending order is immediately available, and (*iii*) the control is failure autonomous.

A control unit is called *entropic* if the circuitry associated with each order assumes a state under failure such that any subsequent attempt to execute that order will cause a summary hang-up without any of the order's control signals having become active.

Self-diagnosing means immediate "incidental" diagnosis as a basic property, as opposed to "self-diagnosable," which means lending itself to easy but finite diagnostic processing, and contains the former term.

The fact that all order structures are not closed may not be apparent, but examination of the repertoire of a typical computer will bring to light orders which violate the definition.

Proof of the existence of a nonclosed order structure capable of all the conventional operations requires only observation of the effect of a failure upon Van der Poel's limiting case machine.[2] (Van der Poel shows that a computer with only a specialized subtract instruction can perform all essential operations.) This machine is left with the empty set upon diminishing its repertoire by one. However, existence in the repertoire of any order which performs a unique function such as setting a flip-flop accessible to no other order is sufficient to render the structure, of which it is a member, outside the closure definition.

A seemingly trivial example of a closed order structure is the case of a repertoire with each instruction duplicated; the imitating routine for a given instruction, in this case, is of length one. This extreme appears to be, at best, equivalent to conventional duplication approaches, and certainly worse than any other self-healing case. However, H. Y. Chang has pointed out that no conventional checking circuits would be necessary in the control and imitation would be performed with no reduction in efficiency. This form of duplication, then, may be viewed as the boundary between self-healing and conventional self-repair, with some of the advantages of both.

III. BACKGROUND

The definitions are chosen advisedly because a control which is self-repairing in the sense described must be the realization of an order structure which is closed, and must be entropic and self-diagnosing (which implies failure autonomy).

The practical possibility of building a self-healing control came to light as the result of study of a (picoprogramming) control with self-diagnosing properties.*

Self-diagnosing is a required attribute because (*i*) the identity of the offending order must be immediately available without processing (the control cannot be trusted to perform diagnostic processing), (*ii*) control activity must cease until remedial action is taken or insane processing may be performed, and (*iii*) failure autonomy must be realized or several orders may be rendered imperfect by a single failure.

Entropic behavior is required so that (*i*) the repairing entity may be called automatically into play each time the defunct order is to be executed and (*ii*) the defunct order's circuitry will remain inactive and not interfere with the repairing entity's action.

The order structure realized must be closed to permit the technique to function; this requirement is central to the technique's operation.

IV. PRACTICABILITY

These requirements may seem artificial; indeed, unattainable. However, the self-diagnosing control has the very properties demanded above, so that the only otherwise artificial requirement to meet is that of order-structure closure.

We conjecture that the addition of about 10 percent to the repertoire of a typical machine would effect closure. Recalcitrant instructions, the length of whose imitating routines threaten to increase without bound, can be handled by the compromise of duplicating those instructions only. Their imitating routines then shrink to one instruction in length. (A machine designed specifically to be self-healing would, of course, try to avoid such instructions.)

V. THE TECHNIQUE

The technique is relatively simple. An inductively coupled detection lead monitoring all order circuits observes a continuous sequence of pulses during normal operation. However, because of the second

* See the Appendix and Refs. 3 through 6.

property of the self-diagnosing feature, when the circuitry associated with an order fails, the monitoring lead will notice a cessation of activity. A time-out will take place, and the machine will be placed in the remedial mode.

In the remedial mode, the entire (failed) instruction is stored, then control is transferred to a location in memory whose address is generated from the order field of the instruction, and normal operation is resumed. The transferred-to location in memory contains the first word of a routine which imitates the action of the failed order using only other orders. This is possible because of the closed order structure.

The transition from imitation to a continuation of the program being run is smooth, requiring no intervention. The next time execution of the failed instruction is called for the machine hangs up again, without any control pulses being generated, because of the control's entropic character. The same procedure as before is then followed.

A simple example is logical left shift (shift the contents of the accumulator logically one binary place to the left), because its imitating routine is brief. Figure 1 shows the effective mechanization of self-healing for the shift left instruction. The action of sensing the health

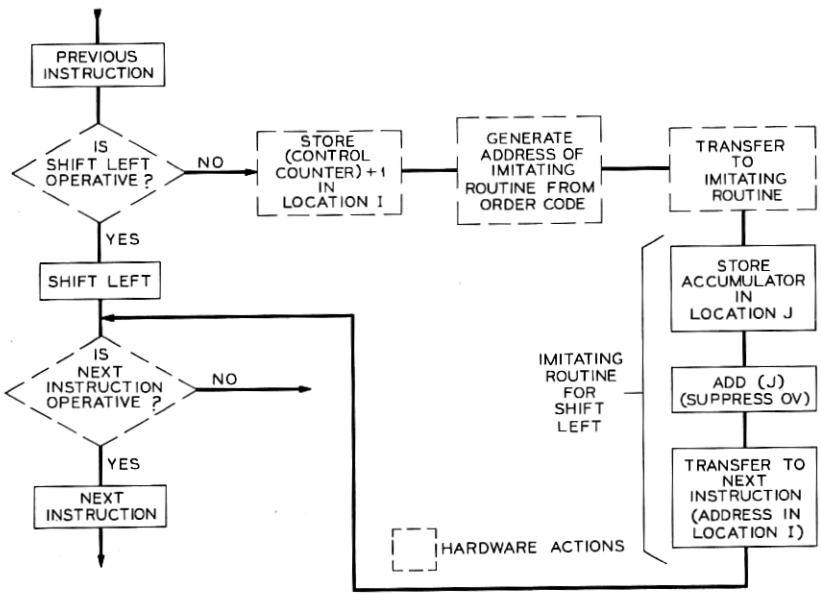

Fig. 1 — Example of self-healing.

of an instruction is indicated by the test preceding each instruction; actually, the test takes place throughout the duration of an instruction's execution, and is much less specific than the illustration implies, asking only the general question, "Is the machine still running?" rather than, "Is shift left operative?"

The net result is that the machine is, in effect, healed; software is used as a patch over the "crack" in the hardware. Execution of the failed instruction will be, in general, very inefficient, but if the time-out is short, the overall speed of the machine generally will be only slightly affected.

If fast memory is considered too expensive or limited in size to contain all of the imitation routines, they may reside in auxiliary storage, and the one selected under failure would be called into a standard block in fast memory. Subsequent uses of the defunct order would execute its imitation routine out of this block. Such "calling" would require more elaborate remedial hardware.

VI. REALIZATION

The picopragrammed control lends itself to self-healing because of certain properties peculiar to its implementation:

(*i*) The circuitry implementing each order is segregated to a single card and is unique to it; magnetic coupling to the control leads prevents propagation of failure effects.

(*ii*) Because the control is autochronous[3] (self-timed), a failure in a sequencer causes an asynchronous hang-up, leaving the order register in one-to-one correspondence with the failed sequencer.

(*iii*) The nature of the ferrite disk[7] sequencer is such that any failure in the active sequencing mode will leave the disk in a partially switched condition (in a state representing one element of a continuum of states) which prevents the success of any subsequent attempts at switching, and results in a hang-up. No control signals are generated by attempts at switching subsequent to failure.

To a "conventional" picoprogrammed control must be added intelligence to perform remedial action. The remedial tasks illustrated in Fig. 1 are:

(*i*) Store in an appropriate location the nonorder fields of the instruction (which usually change each time the order is executed).

(*ii*) Store in an appropriate location the current address (or the current address +1).

(*iii*) Derive from the order field the location of its imitating routine.
(*iv*) Transfer to the imitating routine.

The imitating routine alters its instructions according to the non-order fields of the defunct instruction, performs the imitation, and transfers to the defunct instruction's successor in the interrupted program.

VII. PROTOTYPE IMPLEMENTATION

A small prototype computer was built at Bell Laboratories to test the feasibility of self-healing (see Fig. 2). Picoprogramming was used, providing the necessary properties for self-healing.

7.1 *Duplication*

The special (limiting) case of duplication is the simplest to implement, and was the first tried. In effect, dual repertoires are installed whose order codes differ by one bit (are adjacent). Any number of techniques could be used in a full scale system to guarantee that both halves of the extended repertoire are exercised regularly, such as assembler insertion or internal bit complementing. The equivalent of the second technique was used in the prototype because

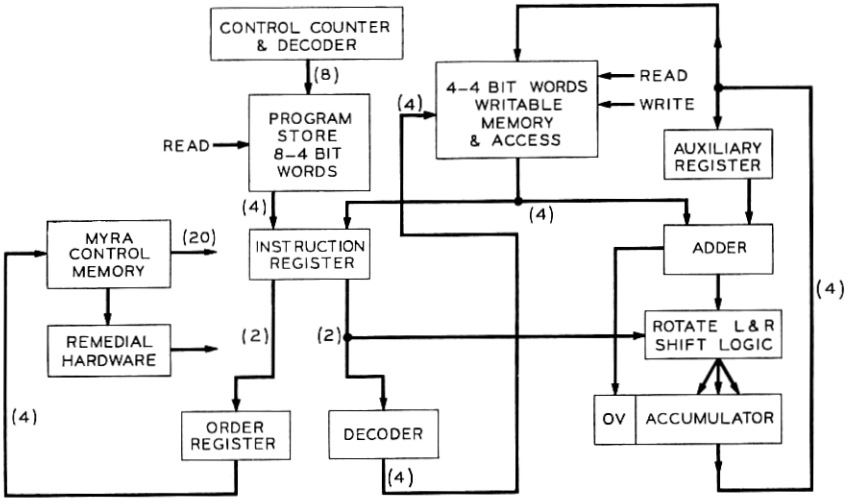

Fig. 2 — Prototype self-healing processor.

it does not require an additional bit per order in main memory; the internal order code was augmented with a bit which served to choose between twin instructions.

The mechanism used to recognize failure (cessation of activity) was a simple integrating circuit, which monitored the completion signal common to all instructions, and a level discriminator.

The remedial action consisted simply of complementing the "twin" bit and injecting a bogus completion signal to start the system again.

Physically simulated failures were experimentally shown to yield to self-healing independent of the instruction class, and the implementation of this grade of self-healing was definitely proved possible.

7.2 *Nonduplication*

A test off the boundary line was necessary to show that a nonduplicating grade of self-healing is possible. The limited size of the prototype's repertoire, however, precluded a complete closed repertoire from being tested. Instead the properties of self-healing were studied "in the small"; repertoires were chosen which could imitate a subset of themselves, but which did not include duplicates.

The procedure followed was to place the machine in a convenient program loop performing calculations easily checked for accuracy, such as a first order Markov chain. In another memory area was stored a routine that could imitate the action of an order used in the calculating routine.

A manual switch was provided to either disable or invoke self-healing capability, and another to disable the object order.

With the machine running in the calculation loop, and with self-healing disabled, it was easily shown that disabling (simulating a failure) the object order would bring the machine to a halt, leaving on display its location, its identity, and its nonorder fields.

Invoking self-healing capability, and running in the calculation routine, disabling the object order had no effect upon the calculations' accuracy in most cases; however, occasionally an error would occur. The errors (which will be explained) occurred only at the instant of simulated failure; calculations after that (while self-healing) were properly performed. When the simulated failure was removed, the machine automatically reverted to its original mode of operation.

An order is always assumed innocent until proven guilty. That is, whether or not an order failed the last time it was to be used, an attempt is made to use it the next time the program calls for it. The

innate properties of a failed order thus provide the necessary memory that the order is inoperative, and prevent these attempts from causing trouble. Should the order spring back to life again, however, either because the "failure" was caused by noise or because the failed card is replaced, the order "takes over" again, and self-healing ceases.

This technique does not solve all problems. One difficulty is that a fragment of an instruction may be executed before failure occurs (this was the cause of the occasionally observed errors at the instant of failure), and it may be impossible to reconstruct the status of the system before execution of this instruction began. Two solutions are:

(*i*) Defensive programming, a technique which is recommended for all real time programming applications, but difficult to implement as a complete solution. It consists of programming in such a fashion that errors (the outward manifestation of a partial instruction execution) do not appreciably disturb operation.

(*ii*) Picoinstruction counting, which forces the picoinstructions to count themselves as they are executed (the counting register is reset upon the successful completion of each instruction). Under failure the contents of this register are digested by the imitating routine, which (the first time) performs only those operations not yet performed by the defunct instruction.

VIII. HARDCORE

8.1 *Control Hardcore*

The hardcore (that portion whose failures cannot be healed) specifically associated with a self-healing control consists of a fixed and a variable component. The fixed component includes the instruction register, an order register which copies the order field of the instruction register, and their associated circuitry; in addition, a relatively simple integrating and level sensing circuit to recognize and react to failure, and a flip-flop and timing circuit to store the control mode and provide interinstruction timing, are required.

The variable component depends upon the degree of duplication. For full duplication a flip-flop is needed to differentiate between twin instructions. Zero duplication requires a means for storing the nonorder field of the failed instruction and its address for use by the imitating routine, and a means for generating the address of the first instruction of the imitating routine and placing it in the instruction counter register (assuming a single address format).

8.2 *Noncontrol Hardcore*

The noncontrol portions of the machine are not directly aided by self-healing, but the indirect help is far reaching. For automatic diagnosis, the hardcore of the machine does not include the entire control, but rather the control's relatively small hardcore. This means that much more of the machine may be automatically diagnosed. For extension beyond diagnosis to self-repair (and possibly self-healing) in the noncontrol portions of the machine, the more effective diagnosis and more trustworthy control are substantial aids.

Among the more interesting possibilities is that of building an all memory machine (including the control) which would use tables in a manner reminiscent of the IBM 1620 to replace the arithmetic unit and reduce the diagnostic problem to one of handling memory (which, because of its relatively homogeneous nature, tends to be tractable).

IX. ANALOGIZING VIEW

Suppose it were feasible to store a set of remedial routines for each order, which could imitate the order's action for all possible combinations of order failures. Then as the machine grew old, and one by one the orders failed, the control would survive but become increasingly slower until a minimum critical subset (possibly one order) remained. Under these conditions the control would be acting in a manner analogous to the compilation of a high-level language, written in itself. It is necessary in the latter case (the compilation) to (software) implement a critical subset of the language which can bootstrap the remainder of the language, just as it is necessary to have a healthy (hardware) implementation of a critical subset of the instructions in the former case (the aged machine). Extending the analogy further, a correspondence may be seen between the computer's microinstruction language (if it exists) for the former case, and the machine language of the computer used in the latter case.

X. CONCLUSIONS

Self-healing is a practicable and potentially useful variation of self-repair.

(Comment: Although picoprogramming was used in this work, it was a vehicle of convenience, not an essential constituent. Other implementations are possible.)

XI. ACKNOWLEDGMENTS

I wish to acknowledge helpful discussions with H. Y. Chang, and the excellent experimental assistance of D. J. Matter.

APPENDIX

Picoprogramming

This is a very brief description of picoprogramming; the serious student should see Ref. 3.

Picoprogramming may be viewed as a logical extension of microprogramming wherein the control memory, which in a practical microprogrammed machine constitutes some fraction of the control, is allowed to expand until it is identical with the control. Figures 3 through 5 depict the evolution in oversimplified form.

Figure 3 illustrates the typical pre-third-generation machine control, with the order portion of the instruction used to identify a sequence implemented, in general, with ill partitioned circuitry denoted by crosshatching.

Figure 4 shows the microprogrammed control where the order field identifies the starting address of a microprogram in the control memory, but a conglomerate of circuitry is still required to implement the microinstructions, count through them, and the like.

Figure 5 shows a picoprogrammed control consisting only of a memory with the order field now choosing a single memory element.

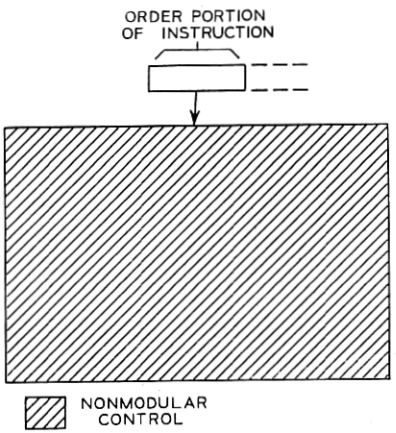

Fig. 3 — Conventional control.

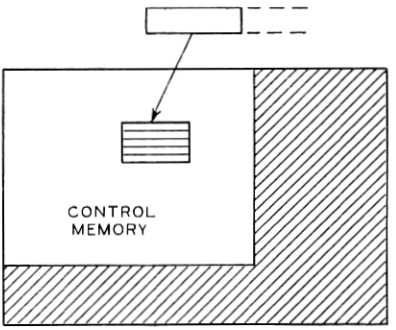

Fig. 4 — Microprogrammed control.

The memory element in Fig. 5 is a square loop ferrite disk called a myra (for myria-apertured) disk. This disk has the property that, when selected, it can spill out many sequential strings of control pulses temporally juxtaposed in almost any desired fashion and at voltage and driving point impedance levels capable of driving most logic circuits directly. Most instructions can thus be implemented with a single disk.

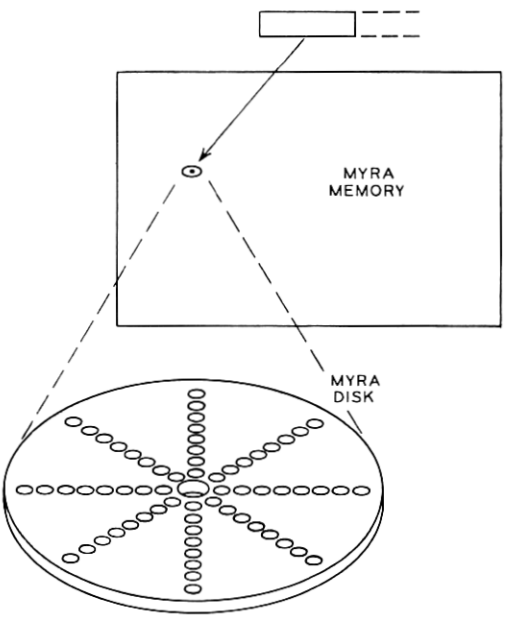

Fig. 5 — Picoprogrammed control.

When these disks are mounted separately with their drivers, they exhibit failure autonomy. Further, since no clock is used, the disk corresponding to an instruction under execution is in complete and independent control, and the machine stops if it fails. In fact, an unequipped card can be used to implement a HALT instruction.

The instruction cards can, in general, be interchanged to provide a variable repertoire.

REFERENCES

1. Avizienis, A., "Design of Fault Tolerant Computers," Proc. Fall Joint Computer Conf., *31* (November 14–16, 1967, Anaheim, Calif.) pp. 733–743.
2. Van der Poel, W. L., "The Essential Types of Operations in an Automatic Computer," Nachrichtentechnische Fachberichte, *4* (1956), pp. 144–145.
3. Briley, B. E., "Picoprogramming: A New Approach to Internal Computer Control," Proc. Fall Joint Computer Conf., *27* (Las Vegas, Nev., November 30–December 2, 1965), pp. 93–98.
4. Valassis, J. G., Macrander, M. C., Pacer, T. A., and Rekiere, R. J., "An Integrated-Circuit MYRA Picoprogrammed Computer," Automatic Electrical Technical Journal, *10*, No. 8 (October 1967) pp. 326–336.
5. Valassis, J. G., Mehta, M. A., and Holden, J. R., "Analysis of the MYRA Picoprogramming Control Technique," Automatic Electric Technical Journal, *10*, No. 8 (October 1967), pp. 327–348.
6. Valassis, J. G., "Modular Computer Design with Picoprogrammed Control," Proc. Fall Joint Computer Conf., *31* (November 14–16, 1967, Anaheim, Calif.) pp. 611–619.
7. Briley, B. E., "MYRA: A New Memory Element and System," IEEE Intermag. Conf. Proc., Washington, D. C. (April 21–23, 1965), pp. 14.8–1—14.8–6.