INTERRUPT DRIVEN PROGRAMMING

M. Zelkowitz

Technical Report

No. 70-78

October 1970

Department of Computer Science
Upson Hall
Cornell University
Ithaca, New York 14850

INTERRUPT DRIVEN PROGRAMMING

Marvin Zelkowitz

Department of Computer Science

Cornell University

October, 1970

ABSTRACT: In "An Interrupt Based Organization For Management Information Systems" (1), the author of that paper proposes a new form of interrupt, a Boolean expression of variables in the program, as a natural extension from the present interrupt structure. His paper describes a software implementation; this note shows how his system could easily be implemented in hardware, with a corresponding decrease in system overhead.

CR CATEGORIES: 4.32 (Supervisory systems), 3.51 (Management data processing), 4.42 (Program debugging)

KEYWORDS AND PHRASES: Interrupts, Interrupt scheduling, supervisors, monitors, debugging, program checkout, parallel processing, associative memories, microprogramming

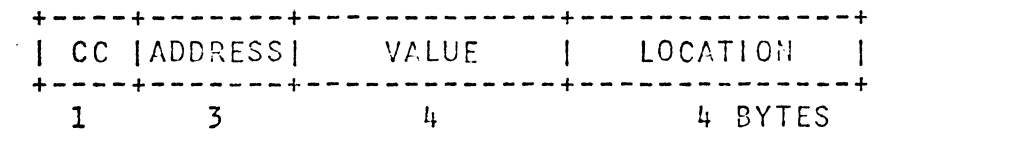
This short note is an extension of the ideas expressed in Morgan's "An Interrupt Based Organization For Management Information Systems" (1). In that paper he proposes a new form of interrupt which he proposes to use to control the execution of a program, in his case a file management system called DPL (2). Simply stated, he has attached to every subroutine a Boolean expression which, if true, causes the subroutine to be executed. Since his implementation is via software, a relatively time consuming check must be made whenever some condition in the program changes (e.g. some variable has a value changed). A rather simple hardware addition to existing machines can easily implement this interrupt. The following is such a proposal.

The specific details of implementation are not crucial to this discussion. What is important is the benefit of hardware implementation. The implementation is described via an IBM 360 environment (3), but is certainly not limited to it. Essentially the implementation involves the addition of a small associative memory to the CPU which is 12N bytes long. Each entry is 12 bytes long, and the number of entries is up to the designer of the computer. (Sixteen will probably be sufficient.) The format for each entry of this CHECK memory is pictured in Figure 1. The CHECK memory operates as follows:

a)For every store or fetch cycle of the main memory, compare the address of the location fetched with the ADDRESS fields in the CHECK memory. This can be done in parallel with the actual fetching or storing of data, and thus should not slow down main memory. If

there is a match, then proceed to step b.

b)Compare the contents of that fetched core location with the VALUE field in the CHECK memory. If the relationship of memory contents to VALUE field is the same as CC, then interrupt and go to LOCATION.



CC = Condition code (for IBM 360)
= 02 (greater than)
= 04 (less than)
= 08 equality
ADDRESS = Address field
VALUE = Test value
LOCATION = Location to begin execution if interrupt

FIGURE 1. FORMAT OF CHECK MEMORY

Since step a is done in parallel with main memory accessing, it should not slow the machine. Step b is only executed for a few special addresses, so if several additional machine cycles must be added, it should not be a significant factor in total system performance.

For example, 8 is the condition code for equality and 4 is the condition code for less than; thus C (in hexadecimal) = 8+4 is the condition code for less than or equal. Therefore, if we wish to go to location 12340 (in hexadecimal) if the location 10000 becomes non positive, then the CHECK memory entry for this is pictured in Figure 2.

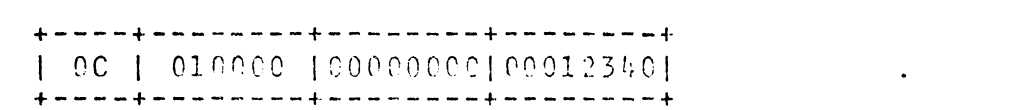


FIGURE 2. INTERRUPT IF LESS THAN OR EQUAL TO 0

Thus one now has the power to do much more checking than is now possible by hardware without slowing the program by software checking. Some possible uses of this memory follows:

1) The concept of event sequenced programming (1,4) is achieved almost without execution time penalty. While complicated Boolean expressions (e.g. A<B or C>D) may not be able to be automatically checked, this CHECK memory does enable the programmer to test against many useful conditions.

2) Statements of the form ON CONDITION (x<0) CALL ERROR_ROUTINE could be added to PL/I (or other higher level languages.) These would have a simple implementation at almost no cost in speed of execution. It would greatly enhance the diagnostic capabilities of present day higher level languages.

3) One could easily envision an operating system where each parallel process is kept in synchronization with other processes by this check memory. For example, a process i may be waiting for a process j to access a buffer x. So if x is placed in the CHECK memory, with i as the LOCATION field, i will be activated when x is accessed by process j. While this description has been naiively simplified, it could be expanded into the design of a real system.

Thus it is seen that many varied uses are possible with this new interrupt. While only a few have been outlined

here, a few moments thought will reveal other practical uses. At present the hardware implementation of such an associative memory is prohibitively expensive for such an untested scheme; therefore, there is the need for interpretive interrupts as in Morgan's DPL to show the feasibility of such a hardware addition.

Even though there appears to be a need for this interrupt, no present day computer has anything like it. The G-20 machine (5) of the early 1960's did have a flag attached to each word which, if set and that word was accessed, would cause an interrupt. This address stop would also be an effective means of implementing the above proposals, but would not be as versatile. (e.g. An address stop could be achieved with the CHECK memory by setting the condition code to 0F.)

The probable reason for the address stop dying with that machine is that programmers debugged on line with it, since it was a rather small machine. They could manually stop the machine when they wanted. It is only with the third generation of computer of the mid 1960's that the interrupt has become an important method to control the execution of a program. It has only become recently apparent that an interrupt like the CHECK memory is needed.

If DPL or future systems prove successful, then hardware additions to implement these interrupts will be sought. This proposal is one simple way to carry this out.

REFERENCES

1. Morgan, H.L., "An Interrupt Based Organization For Management Information Systems. Comm ACM 13, (Nov., 1970)

2. Morgan, H. L., DPL: A Language for instruction in contemporary data processing concepts. Technical report no. 53, Department of Operations Research, Cornell University, Ithaca, New York, 1968

3. System 360 Principles of Operation. IBM Manual A22-6826-7

4. Morgan,H. L., "Event Sequenced Programming". Technical report no.119, Department of Operations Research, Cornell University, Ithaca, New York, 1970

5. Bendix G-20 Central Processor Machine Language. BET-10601-3 Bendix Corporation, 1961