Instruction continuation vs. restart

To the editor:

The article by Furht and Milutinović on microprocessor memory management architectures (Computer, March 1987), while generally useful and informative, reached some questionable conclusions about Motorola’s decision to use instruction continuation, rather than restart, to cope with page faults in the MC68010 and MC68020 microprocessors.

A page fault may occur in several different places in the execution of a memory-oriented instruction: the fetch of the instruction itself, the fetch of each operand, and the store of the result. In machines with true memory-direct addressing modes, each of these places may actually involve multiple references. All of these may be scattered widely within the address space. The instruction continuation strategy precludes indefinite postponement of an individual instruction; it is conceivable that an instruction in a low-priority task, if restarted after a page fault for one memory reference, could encounter a fault on another. Even architectures which, as the authors say, "... are able to detect an address fault before executing and instruction ..." (which sounds more like a "don’t start!" strategy) might encounter awkward waits for all their parts unless copies of successful fetches were maintained.

I am sure some price in circuit complexity must be paid to allow the detailed state of the Motorola’s microcode machine to be pushed onto a stack in the midst of an instruction, but I would guess that price to be modest. It would seem to be much more costly to have to provide for instruction restart for indivisible instructions, as the authors claim is required. Not so! There are only three places in Test...and...Set (TAS) where a page fault can occur: instruction fetch, operand fetch, and operand store. Instruction fetch is irrelevant. If a page fault prevents operand fetch, what difference does it make? Later, it will be fetched successfully and execution will continue. But what if a page fault occurs when the instruction attempts to replace the operand? Something is dreadfully wrong! The operand is being written into the location accessed scant nanoseconds previously, for starters. More relevant, though, is that from the viewpoint of the operating system, TAS, like all instructions, is indivisible. There can be no intervening event on the local process (executing the TAS) which could result in a page fault: no interrupts, no context switching, no competition for storage. And finally, the instruction does not surrender the bus between operand fetch and store (that’s why they call it "indivisible"). No conceivable design could allow a second processor to affect this memory while the local processor still claimed to be accessing it.

The authors mention the "large address error stack" used in instruction continuation. The stack frame for a page fault is 92 bytes (46 words) long in the MC68020 (MC68020 32-Bit Microprocessor User’s Manual). This hardly seems excessive, especially compared to stack frames of typical argument-passing subroutines...

For these reasons, I question the authors’ conclusion that the restart strategy "seems to be more efficient." An on-chip MMU certainly does not prevent a memory access which will result in a page fault (the MMU will intercept the access in either case ...); restart, or "don’t restart," may well be just as complex and no more efficient, and may reduce the freedom of the architect or microcoders in providing complex address-calculation schemes (especially when memory indirection is available, as with the MC68020). One may consider the indefinite postponement problem as unimportant...but there are situations in which instruction continuation is obviously superior...

J.M. Terry
Monmouth, Oregon

Authors’ reply:

The above discussion represents a useful addition to our article. However, we would like to point to some additional facts taken from MacGregor and Mothersole (“Virtual Memory and the MC68010,” IEEE Micro, June 1983), which discusses the application of the continuation method in the MC68010.

Although the continuation method provides a more natural method of virtual memory support, it introduces several problems.

Silicon resources that must be provided to support the saving and restoring of the internal state are not insignificant, and in the case of the MC68010 the additional resources require a 22 percent increase over the MC68000.

Another penalty associated with the continuation method and larger internal state is the additional time to read and write the state during the save and restore operations. In addition, the save time directly contributes to the interrupt latency. In the MC68010 the interrupt latency is increased by about 50 percent over the interrupt latency of the MC68000.

For instructions that have to be executed without interruption, such as Test...and...Set (TAS) of the MC68000, the continuation method does not provide a suitable solution. For the TAS instruction, the processor must provide an uninterruptable read-modify-write sequence. If a fault occurs during the write operation, according to Mac-

---

**LETTERS TO THE EDITOR**
Gregor and Mothersole, it is necessary to have an additional mechanism to reinitiate the whole sequence read-modify-write.

Therefore, the problem of continuation versus restart method is still open, and we invite other researchers to contribute with their opinions and relevant discussions.

Borivoje Furht
University of Miami
Coral Gables, Florida

Veljko Milutinović
Purdue University
West Lafayette, Indiana

New standard for Basic

To the editor:

In your July issue (Update), you reported regarding a DoD Ada directive on Computer programming languages policy. The list of approved higher level languages included Minimal Basic.

Your readers should be aware that Minimal Basic is an obsolete standard. The new ANSI X3.113-1987 standard for Basic adopted in January, 1987, will also be adopted as the Federal Information Processing Standard. . . .

This change of standard is particularly crucial for Basic. The old standard was very small, and most Basic implementations available today extend it in inconsistent ways. This has meant very limited portability for applications. The new standard will solve this problem over time. It is much larger and incorporates sophisticated features. These include matrix operations, structured programming constructs, and data types. . . .

The new standard will mean a boost for Basic. It is still easy to use, but programs will also be portable.

Faruq Ahmad
NKR Research
San Jose, California

October 1987