Reed Electronics Group

MICROPROCESSOR REPORT

www.MPRonline.com

THE INSIDER'S GUIDE TO MICROPROCESSOR HARDWARE

A Tale of Two Instructions

Why AMD64 Lost, Then Regained, Two Minor x86 Instructions

By Tom R. Halfhill {7/19/04-01}

Everyone has experienced the woe of cleaning out a closet and discarding something we needed later. Maybe it was something trivial, like a Pet Rock. Maybe it was something important, like the SAHF and LAHF instructions in the x86 microprocessor architecture.

Few closets are as messy as the x86 is. One problem is that several companies share the same crowded space. Another problem is that everyone keeps putting things into the x86 closet, but almost nobody ever takes things out. In a bold move, AMD recently tried to clean out the closet while installing a new 64-bit shelf. Unfortunately, AMD discarded two old instructions from the 16-bit days of the x86 that aren't as obsolete as they seemed. Now AMD is making additional changes to put everything right again.

The most important changes are that future 64-bit processors from AMD—and perhaps from other x86 vendors, if everyone gets on the same page—will restore the two discarded instructions and define a new bit returned by the CPUID instruction. CPUID is the instruction that an operating system or user-level program can use to identify the computer's processor and determine its capabilities. Starting with new AMD64 processors coming later this year, CPUID will set a previously undefined bit in a general-purpose register to indicate whether the processor revives the SAHF and LAHF instructions that AMD mistakenly purged from 64-bit mode in AMD64. The strange story of the death and resurrection of these instructions is a classic example of the reason the x86 architecture has grown so complex over the past 26 years.

SAHF and LAHF: Never Say Die

As we reported last March in our detailed comparison of the 64-bit x86 architectures from AMD and Intel, AMD deleted several old instructions while defining AMD64. (See *MPR 3/29/04-01*, "AMD and Intel Harmonize on 64.") Among the instructions omitted from 64-bit mode—but preserved in 16- and 32-bit modes for backward compatibility—were SAHF and LAHF.

Intel introduced both instructions in 1982 with the 16-bit 286 processor. SAHF and LAHF are simple, parameterless, complementary instructions with single-byte opcodes (9E and 9F). SAHF saves five condition flags (sign, zero, parity, carry, and auxiliary carry) into the AH register, also known as the accumulator. LAHF loads the same five condition flags from the accumulator. The back-and-forth swapping was useful for context switching, because a single instruction saves or restores the state of the flags for a particular process.

When AMD was defining AMD64, it decided to ditch SAHF and LAHF in 64-bit mode, because today's operating systems no longer use them for context switching. Since the 16-bit days of the 286, Intel has added new condition flags that SAHF and LAHF don't know about. Modern operating systems typically use an instruction called PUSHF, to push all the old and new flags onto the x86 stack, and then an instruction called POPF, to pop them off the stack and restore the flag state. SAHF and LAHF seemed to be redundant, obsolete instructions occupying valuable single-byte-opcode real estate, so early copies of AMD64 technical manuals warned that SAHF and LAHF were invalid in 64-bit mode. The Athlon 64 and Opteron chips currently on the market don't support the instructions.

For More Information

AMD's five-volume set of AMD64 programmer's manuals is available in book form or can be freely downloaded as Adobe PDF files here:

- *www.amd.com/us-en/Processors/SellAMDProducts/0,,30_177_4458_3505%5E4699%5E875%5E7044,00.html*

AMD's "Processor Recognition Application Note" for using the CPUID instruction can be freely downloaded as an Adobe PDF file here:

- *www.amd.com/us-en/Processors/TechnicalResources/0,,30_182_739_1102,00.html*

Intel's two-volume set of EM64T programmer's manuals is available in book form or can be freely downloaded as Adobe PDF files here:

- *www.intel.com/technology/64bitextensions/*

Intel's "AP-485 Intel Processor Identification Application Note" for using the CPUID instruction can be freely downloaded as an Adobe PDF file here:

- *http://developer.intel.com/design/xeon/applnots/241618.htm*

Meanwhile, Intel was defining EM64T, its own 64-bit extensions for the x86. Striving for software compatibility with AMD64, Intel studied AMD's early technical manuals and AMD64 processors. Everything indicated that SAHF and LAHF were invalid in AMD's 64-bit mode; Intel therefore purged them from 64-bit mode in EM64T, too.

Then came surprising and disturbing news: some modern programs still use SAHF and LAHF. Without those instructions, programmers had to port their code to 64-bit mode using clumsy patches and work-arounds. Mainly, the affected programs are x86 instruction-set simulators, emulators, and dynamic binary translators. AMD began hearing from third-party vendors of these programs, and even from one of its own internal product groups—because AMD's *SimNow* uses SAHF and LAHF, too. (*SimNow* is an x86 instruction-set simulator that independent software developers can use to write code before actual silicon of a new AMD processor is available.)

Simulators use SAHF and LAHF to avoid some stack madness. An x86 simulator must maintain at least two stacks: the monitor stack, for the simulator itself, and the application stack, for the program running on the simulator. (The application stack is the simulated x86 stack.) These two stacks exist alongside the native stack of the processor on which the simulator runs. Without SAHF and LAHF—which, as noted above, save and restore the flags using the accumulator, not the stack—the simulator would have to do much more work. It would have to execute the PUSHF instruction to push the flags onto the application stack; redirect the stack pointer to the monitor stack for the context switch; redirect the stack pointer back to the application stack before returning to the first context; and then execute the POPF instruction to pop the flags off the application stack and restore the original simulated processor state.

Redirecting the stack pointer from one stack to another during this context-switching process could have troublesome side effects. For instance, another context switch could change the state of the stacks, requiring additional control code so they could remember their original states and the original locations of their stack pointers. Of course, any new control code in the context-switching path would reduce performance, already a concern with simulators. To keep its customers happy, AMD soon realized it would have to restore SAHF and LAHF in 64-bit mode on AMD64 processors. Unfortunately, the first AMD64 processors were already shipping—without the instructions.

Can't We All Just Get Along?

Revising manuals is a lot easier than revising processors, so AMD's first step was to issue new AMD64 technical manuals listing SAHF and LAHF as valid in 64-bit mode. However, AMD didn't call special attention to the revision, so only the most eagle-eyed readers, familiar with both versions of the books, would have divined that a change was coming in future AMD64 processors. Among those who didn't notice were the Intel engineers working on EM64T. It wasn't their fault. AMD didn't communicate the information directly to Intel, because the two companies have approximately the same relationship as Itchy and Scratchy on *The Simpsons*.

Microprocessor Report missed the revision, too. The oversight surfaced when *MPR* sent a draft of our AMD64/EM64T comparison (cited above) to Intel for technical review. After reading that AMD had restored LAHF and SAHF, Intel objected that the information was inaccurate—with good reason: Intel's tests of the latest AMD64 processors showed that invoking either instruction in 64-bit mode caused an illegal opcode exception. *MPR* double-checked with AMD, which acknowledged that current Athlon 64 and Opteron processors (AMD64 revision C) don't support the instructions. AMD said future processors would restore them, but the company didn't say when.

Recently, AMD informed *MPR* that new AMD64 processors, scheduled to ship later this year, will include SAHF and LAHF. To give programmers a familiar way of checking whether a particular processor supports the instructions, AMD has also defined a new bit returned by a CPUID extended function call. The function—identified by the 32-bit hexadecimal number 8000_0001h—returns values in the x86 general-purpose registers. For example, when a program executes CPUID and passes 8000_0001h as a parameter in the EAX register, the function returns the "processor signature" (the processor model, instruction family, stepping number, and so forth) in the same register. In new AMD64 processors, function 8000_0001h will also set bit 0 in the ECX register if the processor supports SAHF and LAHF. If the processor doesn't support the instructions, the function will

clear ECX bit 0. Previously, all bits in ECX were undefined after calling extended function 8000_0001h.

Intel is still deciding whether future EM64T processors will restore SAHF and LAHF in 64-bit mode. The first EM64T processors certainly won't support them, because Intel—like AMD—was too far along toward final silicon when AMD reversed its hasty decision to drop the instructions. We think there's a fair chance the instructions will reappear in EM64T at some point, because Intel says it wants to preserve as much compatibility as possible between the 64-bit architectures.

Whether or not Intel restores SAHF and LAHF, it would be useful for the company to modify extended-function 8000_0001h, as AMD is doing, so programmers can readily identify whether a processor supports the instructions in 64-bit mode. It would also be useful for AMD and Intel to strengthen the diplomatic backchannels between their companies to discuss these kinds of compatibility issues. AMD's first fling in the driver's seat of the x86 resulted in a wrong turn, but it caused no serious harm—this time. Software developers and users have too much at stake for x86 vendors to leave compatibility up to chance.