Reed Electronics Group

MICROPROCESSOR REPORT

www.MPRonline.com

THE INSIDER'S GUIDE TO MICROPROCESSOR HARDWARE

AMD and Intel Harmonize on 64

Intel's 64-Bit x86 Extensions Are Largely Compatible With AMD64

By Tom R. Halfhill {3/29/04-01}

Intel says its new 64-bit x86 extensions will run the same 64-bit operating systems and almost all the same 64-bit application software as AMD's 64-bit architecture does. AMD says software compatibility should be no surprise, because Intel virtually reverse-engineered its 64-bit extensions from AMD64. An independent analysis by *MPR* indicates both companies are correct. Except for a few minor differences, the two 64-bit architectures are identical.

MPR compared all the new instructions, modified instructions, deleted instructions, and modifications to the register files—including control registers, system registers, and registers visible to application programs. We also compared the memory-addressing schemes and many other architectural features, such as data-addressing modes, context-switching behavior, interrupt handling, and support for existing 16- and 32-bit x86 execution modes. In every case, we found that Intel had patterned its 64-bit x86 architecture after AMD64 in almost every detail.

However, we also found a few differences that could make some software written for one 64-bit architecture incompatible with the other architecture. Some of these differences may be resolved in future 64-bit x86 processors, or even in future steppings of x86 processors already announced or on the market. In other cases, software can easily adapt to the differences by executing slightly different code, after first probing the CPU to learn which 64-bit extensions it supports.

MPR found nothing to contradict Intel's promise that its 64-bit x86 processors will run the 64-bit operating systems developed for AMD64. At the same time, Intel's reluctance to make a blanket guarantee about mutual 64-bit software compatibility is justified by the minor differences we discovered. Of course, that's always the case when Intel or AMD introduces new x86 extensions—such as Intel's SSE3 media extensions, which aren't yet supported by AMD.

Despite the differences, Intel clearly derived its 64-bit architecture by reading AMD's prerelease documentation for AMD64 and by testing AMD64 processors. (See sidebar, "Intel and AMD Manuals Sing Similar Tunes.") Intel's reverse-engineering of AMD64 marks a major turning point in the historical relationship between the companies. Although AMD has in the past introduced some innovations to the x86 architecture—the 3DNow multimedia extensions being a prime example—this is the first time AMD has truly steered the direction of the world's most important microprocessor architecture, which Intel invented in 1978 and has closely guarded for 26 years.

Intel plans to introduce its first 64-bit x86 chip, the Nocona Xeon server processor, in 2Q04. Actually, Nocona is based on the same core as the recently released Prescott Pentium 4, which has the 64-bit extensions but disables them. Nocona activates the extensions for the first time and enables other features to support two-way multiprocessing. (See *MPR 3/15/04-01*, "Intel Addresses the 64-Bit Question.")

AMD, of course, has been shipping 64-bit x86 processors since early 2003, when it introduced the Opteron server processor. (See *MPR 4/28/03-01*, "AMD Serves Up Opteron.") AMD's Athlon 64 desktop processor has been shipping since September 2003 and is now available in several variations. (See *MPR 1/20/04-01*, "Athlon 64 Moving to Mass Market.")

Deciphering the Nomenclature

Before reviewing all the similarities and differences between the 64-bit x86 architectures from AMD and Intel, it's important to understand the terminology, which can be confusing. AMD originally named its 64-bit architecture x86-64, but now calls it AMD64. Intel recently announced that its 64-bit x86 architecture will be called Intel Extended Memory 64 Technology (EM64T). Intel also uses the term IA-32e to refer specifically to the new 64- and 32-bit execution modes of its EM64T processors. ("IA-64" would have been the logical name for Intel's 64-bit x86, but it's already the name of a completely different and incompatible 64-bit architecture that is the basis of Intel's Itanium server processors.)

Now that AMD has dropped "x86-64" in favor of "AMD64," *MPR* will adopt "x86-64" as the generic term for the union of the two 64-bit x86 architectures. We will use the terms EM64T and AMD64 when referring to respective Intel and AMD vendor-specific features.

Got it? Good, because now things become even more confusing. Processors based on either x86-64 architecture have two distinct execution modes. When they are running a 32-bit operating system, they are in "legacy mode," which, to the software, looks exactly like a 32-bit x86 processor. For complete software compatibility, legacy mode supports all the existing execution modes of the x86, including the older 16-bit modes. When a processor is in legacy mode, it's as if the 64-bit architecture doesn't exist.

When an x86-64 processor runs a 64-bit operating system, it enters a different execution mode that AMD calls "long mode" and Intel calls "IA-32e mode." In this mode, the 64-bit extensions become visible to 64-bit software. Long mode and IA-32e mode include a 32-bit submode that both companies call "compatibility mode." In this mode, unmodified 32-bit software can run on the 64-bit operating system and can coexist with 64-bit application software, but it cannot use the 64-bit extensions. Compatibility mode makes it unnecessary to dual-boot a 32-bit operating system with the 64-bit operating system to run 32-bit software.

However, 32-bit compatibility mode is not quite the same as 32-bit legacy mode. Although both can run 32-bit software, compatibility mode *requires* a 64-bit operating system, whereas legacy mode *cannot run* a 64-bit operating system. Furthermore, compatibility mode doesn't support Virtual 8086 mode, which allows a 16- or 32-bit protected-mode operating system to run programs written for 16-bit "real mode," the original unprotected execution mode of the x86. Compatibility mode doesn't support real mode, either, but it does support 16-bit protected mode.

In response to a query from *MPR*, Microsoft has confirmed that its 64-bit versions of Windows XP and Windows Server 2003 won't have a "compatibility box" for running 16-bit software. Therefore, to run old 16-bit programs on an x86-64 processor, it will be necessary to boot a 16- or 32-bit operating system in legacy mode, either instead of or in addition to the 64-bit operating system. Of course, few of today's users run 16-bit software, so this is a minor concern.

In AMD's long mode and Intel's IA-32e mode, an attributes byte in the code-segment descriptor determines when a 64-bit operating system must switch between 64-bit execution mode and 32-bit compatibility mode. Software code in a 64-bit memory segment requires 64-bit mode; code in a 32-bit memory segment requires compatibility mode. The code spaces (and the memory-addressing schemes within those spaces) are completely separate, which is similar to the way a 32-bit operating system switches between 32- and 16-bit modes on the x86, so x86-64 preserves the historical structure. Figure 1 illustrates the relationships among the various execution modes for x86-64 processors.

In general, x86-64 extends or adapts existing x86 structures instead of introducing entirely new ones, even when a new mechanism might be more in vogue with current architectural trends. Both AMD and Intel want to preserve the x86's character, complex though it is, for the sake of familiarity. They believe x86 programmers already face a steep enough learning curve.

Comparing 64-Bit Instruction Sets

Now to the meat of x86-64. Each company has introduced a slightly different instruction-set architecture (ISA); these will probably converge over time. The core x86 instruction set remains the same: it's still very much a CISC instruction set with roots in the 1970s, carrying along such anachronisms as instruction prefixes, complex multicycle instructions that execute in microcode, and variable-length opcodes ranging in size from 8 bits to 120 bits. Nobody will mistake x86-64 for a 64-bit RISC.

Numerous instructions that formerly manipulated 32-bit and smaller operands can now handle 64-bit operands, as well, thanks to new or extended 64-bit registers and a new 64-bit instruction prefix called `REX`. The `REX` prefix modifies existing instructions so they can access the full width of the 64-bit registers. (See *MPR 9/4/00-01*, "AMD Drops 64-Bit Hammer on x86.") Both AMD and Intel use the `REX` prefix in the same way, and both companies have reassigned the same space in

Operating Mode		Operating System Required	Application Recompile Required	Defaults		Register Extensions	Typical GPR Width
				Address Size	Operand Size		
Long Mode or IA-32e Mode	64-bit mode	New 64-bit OS	Yes	64 bits	32 bits	Yes	64 bits
	Compatibility mode		No	32 bits	32 bits	No	32 bits
				16 bits	16 bits		16 bits

Figure 1. This chart, adapted from the 64-bit programming manuals from AMD and Intel, shows that x86-64 processors have two new execution modes distinct from the existing 32-bit "legacy mode."

the opcode map to define various forms of the prefix. That opcode space is used by the single-byte opcode versions of the INC and DEC instructions in the 32-bit ISA, so the opcodes for those instructions are different for 64-bit software but otherwise remain the same.

In all, there are only ten "new" 64-bit instructions in the superset of both 64-bit ISAs. We qualify the term "new" because AMD and Intel quibble over the definition. Intel's 64-bit programming manuals list ten new instructions, but six of them are actually extended versions of existing instructions that use the same opcodes, usually with slightly different mnemonics. These six instructions (CDQE, CMPSQ, LODSQ, MOVSQ, MOVZX, and STOSQ) perform the same kinds of operations on 64-bit operands (quadwords, in x86 parlance) that six instructions with similar or identical mnemonics (CWDE, CMPSD, LODSD, MOVSD, MOVZX, and STOSD) perform on 32-bit doubleword operands. In other words, the instructions are 64-bit extended versions of existing 32-bit instructions, like many other 64-bit instructions, but they may have slightly different mnemonics assigned to the same opcodes. For that reason, AMD doesn't consider these six instructions genuinely new, as Intel does.

Two other "new" instructions—SYSCALL and SYSRET—aren't really new, either. AMD introduced them years ago in its 32-bit x86 processors. Now, in EM64T, Intel supports the instructions for the first time, though in a limited fashion.

Yet another "new" instruction is a simple extension of an existing x86 instruction that already manipulates 64 bits at a time; the new version manipulates 128 bits. Although many other 64-bit instructions merely extend the operands of 32-bit instructions, this particular instruction—CMPXCHG16B—deserves special mention because it's the only new instruction we found that isn't supported by both 64-bit ISAs. We'll explain the genesis of this instruction shortly.

We are thus left with only one truly new 64-bit instruction: SWAPGS. Like CMPXCHG16B, SYSCALL, and SYSRET, it requires some explanation. Refer to Table 1 for a list of all the instructions Intel or AMD considers new.

As noted above, CMPXCHG16B is the only new instruction not supported by both 64-bit ISAs. Intel has it; AMD doesn't. It's an extension of CMPXCHG8B, an instruction found in 32-bit x86 processors. The CMPXCHG8B instruction compares the contents of two 32-bit registers (EDX and EAX) with an eight-byte value in memory. If the values are equal, it replaces the memory value with the contents of two other registers (ECX and EBX) and sets the ZF flag. Otherwise, it loads the memory value into the EDX and EAX registers and clears the ZF flag. The new 64-bit version of this instruction, CMPXCHG16B, works in a similar fashion with pairs of 64-bit values, albeit with different pairs of new 64-bit registers. Note that although CMPXCHG16B operates on 128 bits of data, it's part of Intel's core 64-bit ISA for EM64T, not part of the SSE/SSE2/SSE3 multimedia extensions, which commonly operate on 128-bit operands, using a separate file of 128-bit-wide XMM registers.

In 32-bit software, programmers sometimes use the CMPXCHG8B instruction to compare a 32-bit pointer and a 32-bit version number stored in contiguous memory locations. Because CMPXCHG8B is a single instruction that can compare and (optionally) change both 32-bit values simultaneously, it's useful for updating the pointer and version number atomically, without interruptions. In 64-bit software, pointers can be 64 bits long, so Intel decided it was logical to extend the instruction to create CMPXCHG16B, which operates on two 64-bit values stored in contiguous memory locations. Like CMPXCHG8B, it is an atomic (interlocked) operation.

An AMD64 programmer could try achieving the same result as CMPXCHG16B by using two CMPXCHG8B instructions (or a larger number of ordinary CMP and MOV instructions), but the operation wouldn't be atomic, which could be vital for some functions that must not be interrupted. For that reason, and to strive for 100% compatibility, AMD will almost certainly add CMPXCHG16B to future AMD64 processors.

Faster Context Switching in x86-64

The new SWAPGS instruction was created by Kevin McGrath, an AMD Fellow and the architect of AMD64, and David Cutler, the former VMS guru from Digital who joined Microsoft and masterminded the development of Windows NT. As McGrath tells the story, Cutler sought a faster way for the 64-bit version of Windows XP to switch contexts between applications and the operating system. SWAPGS was the result. It allows the operating system to rapidly load into the GS register a pointer to system data structures. (The GS register is an existing segment register that holds a descriptor for a data segment.)

Typically, SWAPGS will execute after a SYSCALL instruction to the operating system. Before switching back

64-Bit Instruction	Description	Notes
CDQE	Convert doubleword to quadword	New mnemonic for old opcode
CMPSQ	Compare strings by quadword	New mnemonic for old opcode
CMPXCHG16B*	Compare and exchange 16 bytes	Supported by EM64T only
LODSQ	Load string quadword	New mnemonic for old opcode
MOVSQ	Move string quadword	New mnemonic for old opcode
MOVZX	Move with zero-extend	64-bit version of old instruction
STOSQ	Store string quadword	New mnemonic for old opcode
SWAPGS	Swap GS base register	Supported by AMD64 and EM64T
SYSCALL**	Fast system call to ring 0	Supported by AMD64 and EM64T
SYSRET**	Return from fast system call	Supported by AMD64 and EM64T

Table 1. These are the ten "new" instructions in the 64-bit ISAs from AMD and Intel. Half are extended versions of existing instructions with slightly different mnemonics but not genuinely new opcodes. *AMD64 supports only the 32-bit version of this instruction (CMPXCHG8B), not CMPXCHG16B. **Instructions formerly supported by AMD only in its 32-bit ISA, now supported by AMD in AMD64 and Intel in EM64T.

to the context of the application program (with SYSRET), the operating system executes another SWAPGS instruction to restore the application's data pointer in the GS register. SWAPGS is valid for 64-bit software only and is supported by AMD64 and EM64T.

Which brings us to SYSCALL and SYSRET, two almost-new instructions that contribute another confusing chapter to the convoluted history of the x86. Historically, AMD and Intel have offered two different pairs of 32-bit instructions for switching contexts to and from the operating system. Both pairs of instructions perform essentially the same task, but they work somewhat differently. Intel supported SYSENTER and SYSEXIT; AMD introduced SYSCALL and SYSRET in its 32-bit processors several years ago. For the sake of compatibility, AMD's 32-bit processors also support SYSENTER and SYSEXIT, but Intel's 32-bit processors don't support SYSCALL and SYSRET.

When creating AMD64, AMD promoted SYSCALL and SYSRET to 64-bit rank, and AMD continues to support the 32-bit versions of the instructions. In a new spirit of 64-bit harmony, Intel now supports SYSCALL and SYSRET as well, but only for 64-bit software—32-bit software must still use SYSENTER and SYSEXIT or alternative instructions that do the same thing, even when running in IA-32e compatibility mode on a 64-bit operating system. At the same time, Intel provides extended 64-bit versions of SYSENTER and SYSEXIT for loyalists who can't bring themselves to use SYSCALL and SYSRET in IA-32e 64-bit mode.

Meanwhile, AMD decided not to support SYSENTER and SYSEXIT for 64-bit software, claiming that Windows programs almost never use them, anyway. However, both instructions are still valid for 32-bit software running on a 32-bit operating system in legacy mode on an AMD64 processor. Looking at confusing architectural baggage like this, historians will someday scratch their heads and wonder why the x86 so decisively won the RISC vs. CISC war.

Instruction	Description	64-Bit Invalid	64-Bit Reassigned
AAA	ASCII adjust after add	AMD - Intel	—
AAD	ASCII adjust AX before division	AMD - Intel	—
AAM	ASCII adjust AX after multiply	AMD - Intel	—
AAS	ASCII adjust AL after subtract	AMD - Intel	—
ARPL	Adjust RPL field of segment selector	—	AMD - Intel
BOUND	Check array index against bounds	AMD - Intel	—
CALL	Call far absolute addr in operand	AMD - Intel	—
DAA	Decimal adjust AL after add	AMD - Intel	—
DAS	Decimal adjust AL after subtract	AMD - Intel	—
DEC	Decrement by 1 (single-byte opcode)	—	AMD - Intel
INC	Increment by 1 (single-byte opcode)	—	AMD - Intel
INTO	Interrupt to overflow vector	AMD - Intel	—
JMP	Jump to far absolute addr in operand	AMD - Intel	—
LAHF	Load status flags into AH register	Intel	—
LDS	Load DS:r16 with far ptr from memory	AMD - Intel	—
LDS	Load DS:r32 with far ptr from memory	AMD - Intel	—
LES	Load ES:r16 with far ptr from memory	AMD - Intel	—
LES	Load ES:r32 with far ptr from memory	AMD - Intel	—
POP DS	Pop top of stack into DS register	AMD - Intel	—
POP ES	Pop top of stack into ES register	AMD - Intel	—
POP SS	Pop top of stack into SS register	AMD - Intel	—
POPA	Pop all general-purpose regisers	AMD - Intel	—
POPAD	Pop all general-purpose regisers	AMD - Intel	—
PUSH CS	Push CS register onto stack	AMD - Intel	—
PUSH DS	Push DS register onto stack	AMD - Intel	—
PUSH ES	Push ES register onto stack	AMD - Intel	—
PUSH SS	Push SS register onto stack	AMD - Intel	—
PUSHA	Push all GPRs on stack (words)	AMD - Intel	—
PUSHAD	Push all GPRs on stack (doublewords)	AMD - Intel	—
SAHF	Store AH register into flags	Intel	—
SYSENTER	Fast system call	AMD	—
SYSEXIT	Fast system exit	AMD	—

Table 2. Instructions listed in this table as "64-Bit Invalid" remain in the 32-bit x86 ISA but are no longer available to 64-bit programs. Instructions listed as "64-Bit Reassigned" are still available to 64-bit programs but have reassigned opcodes in 64-bit mode. AMD and Intel largely agree on these changes, except that Intel continues to support SYSENTER and SYSEXIT for 64-bit software and does not support LAHF and SAHF for 64-bit software.

Deleted Instructions and Strange Differences

To be fair, AMD did seize the opportunity to do some housecleaning with AMD64. Some old features, rarely or never used by modern operating systems, such as V8086 mode and hardware task switching, are gone. The instruction set is trimmed down, too. In all, 27 instructions from the 32-bit x86 ISA are invalid when running 64-bit software on an AMD64 processor. Intel largely followed suit by likewise invalidating 27 instructions in the 64-bit mode of EM64T processors. However, *MPR* discovered that Intel's 27 invalid instructions aren't the same as AMD's 27 invalid instructions.

The differences are the aforementioned SYSENTER and SYSEXIT plus LAHF (load status flags into AH register) and SAHF (store AH register into status flags). In 64-bit mode, Intel supports SYSENTER and SYSEXIT, but AMD doesn't; AMD supports LAHF and SAHF, but Intel doesn't.

LAHF and SAHF are an odd case that illustrates the lack of communication between these rival companies. Intel doesn't support the instructions because it defined EM64T after studying early versions of AMD64 manuals and testing early AMD64 processors. At first, those manuals and processors made LAHF and SAHF invalid for 64-bit software. Later—it's unclear when—AMD decided to support the instructions. They are useful for fast context switching because they can save and restore all the status flags in a single operation. The latest AMD64 manuals list LAHF and SAHF as valid for 64-bit programs, and AMD says the latest steppings of AMD64 processors will execute them. Unfortunately, because AMD and Intel don't collaborate on issues like this, Intel was unaware of AMD's reversal and omitted the instructions from EM64T.

Even while *MPR* was preparing this article, Intel said any attempt to execute the LAHF and SAHF instructions on the latest AMD64 processors in its possession caused an illegal-opcode exception. *MPR*, too, has been unable to verify that the latest AMD64 processors support the instructions.

As a result, EM64T doesn't support LAHF and SAHF for 64-bit software, at least in the first implementations of Intel's 64-bit processors. (Note that the descriptions of these instructions in Intel's *64-Bit Extension Technology Software-Developer's Guide*, Revision 1.00, are inconsistent: they are listed as both valid and invalid on the same pages.) Intel declines to say whether future EM64T processors will support the instructions. It may not matter, if programmers avoid using LAHF and SAHF—a likely possibility, since the instructions apparently won't execute on most AMD64 processors already in the hands of users. Although it would be possible to use the instructions by probing the CPU ID and then executing a special code path, it's probably not worth the effort.

Except for SYSENTER, SYSEXIT, LAHF, and SAHF, the other instructions deleted from 64-bit mode are the same in both ISAs. In addition, both 64-bit ISAs reassign the opcodes for three instructions: DEC and INC (to release opcode space for the REX prefixes mentioned earlier) and ARPL. Table 2 lists all the deleted and reassigned instructions.

Eagle-eyed programmers may find what appears to be another incompatibility between the 64-bit ISAs from AMD and Intel, but *MPR* has ascertained that it's a documentation error. Volume II of Intel's *64-Bit Extension Technology Software-Developer's Guide* (Revision 1.00) states that the PUSH-immediate instruction will zero-extend a 32-bit operand to 64 bits. Actually, the operand will be sign-extended, not zero-extended, which is consistent with the PUSH-immediate instruction in AMD64.

Register Files Are Fully Compatible

As far as *MPR* has been able to determine, the register files of AMD64 and EM64T are functionally identical. Those register files include the registers visible to application programs as well as the special system and control registers visible only to software running at the highest privilege level of the processor—mainly the operating system. With both 64-bit ISAs, the new registers and extended portions of existing registers are visible to 64-bit programs only.

Application-visible registers are more plentiful in x86-64, although in this respect most RISC processors are still better endowed. The constricted general-purpose register (GPR) file of the old x86 ISA has only eight 32-bit registers, and they're not truly as "general purpose" as GPRs in modern architectures, due to historical limitations that date back to the x86's inception as a 16-bit architecture. In x86-64, all eight of the original GPRs (EAX–ESP) are extended to 64 bits, and there are eight additional 64-bit GPRs (r8–r15). Doubling the GPR file is a big improvement. Even so, a typical RISC processor has 32 GPRs. AMD says it settled for 16 GPRs because they're enough to relieve most of the congestion of the old register file while preserving the shorter register addresses that help make x86 code denser than RISC code.

The multimedia register file, which Intel introduced in 1999 with the Streaming SIMD Extensions (SSE), is still 128 bits wide in x86-64 but twice as large: eight new registers, for a total of sixteen. The floating-point register file, actually addressed as a LIFO stack in the x86/x87, is unchanged in x86-64: it's still eight registers deep and 80 bits wide. MMX multimedia instructions share these registers with floating-point instructions, but they address the registers as a flat file, not as a stack, and use only the 64-bit mantissa portion of each 80-bit register. Figure 2 shows the most important changes to the register files in x86-64.

Several control and status registers have been extended to 64 bits in x86-64: the instruction pointer, flags register, CR0–CR4 control registers, debug registers, and descriptor-table registers. The extended CR3 register allows page tables to be located anywhere in the 64-bit flat-memory space, and the extended descriptor-table registers do the same for segment tables.

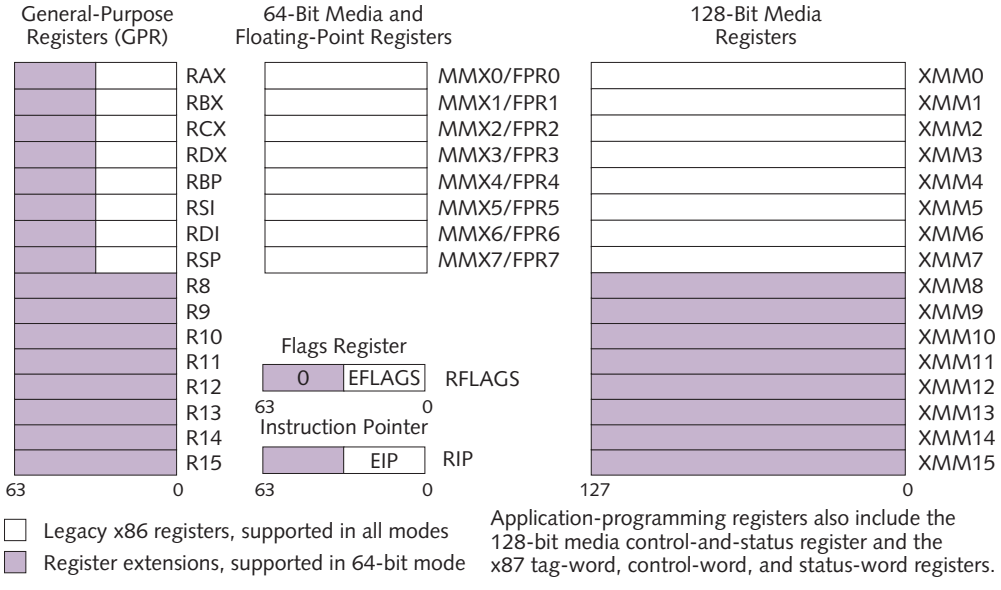

Figure 2. The 64-bit ISAs from AMD and Intel both define the same register files, adding several new 64-bit registers and extending the 32-bit registers to 64 bits. Apart from some minor differences in register names, AMD64 and EM64T are fully compatible in this aspect of the x86-64 architecture.

Intel and AMD Manuals Sing Similar Tunes

Programmers who carefully read the 64-bit programming manuals from AMD and Intel will notice many similarities. Numerous passages in the two-volume EM64T manual echo AMD's five-volume AMD64 manual, sometimes word for word. It's not a coincidence, and neither company considers it plagiarism.

AMD and Intel have a cross-licensing agreement as well as a "gentlemen's agreement" that allows them to borrow portions of each other's documentation, says AMD64 architect Kevin McGrath. The companies have agreed that clarity and consistency are vital for programmers struggling to make sense of the Byzantine x86 architecture.

When one company is trying to reverse-engineer the latest extensions introduced by another company, the documentation is an important resource. Ideally, AMD and Intel would collaborate on their x86 extensions, but their competitive relationship and litigious history make that almost as difficult to arrange as a Middle East peace conference. Consequently, AMD has studied Intel's documentation to support MMX, SSE, and SSE2 in current AMD processors, and Intel has studied AMD's documentation to support AMD64 in EM64T processors. It's a sure bet that AMD's engineers are right now poring over Intel's documentation for SSE3.

Unfortunately, reverse-engineering occasionally results in a misstep. The EM64T architecture doesn't support the LAHF and SAHF instructions in 64-bit execution mode, because the first AMD64 processors didn't implement the instructions, and the AMD64 manuals listed them as invalid. When AMD changed its mind, Intel found out too late to include the instructions in its current definition of EM64T.

Two new special registers are the extended feature-enable register (EFER) and the task-priority register (TPR, also known as CR8). AMD refers to the EFER simply as EFER, whereas Intel calls it IA32_EFER, but they're functionally the same. They control and indicate the status of special features, such as 64-bit mode. For example, bit 8 (LME) in this register enables long mode on an AMD64 processor and IA-32e mode on an EM64T processor, and bit 10 (LMA) indicates whether long mode or IA-32e mode is currently active.

The new TPR is the result of another collaboration between Kevin McGrath at AMD and David Cutler at Microsoft. One of the few registers that Windows uses more often than the accumulator (EAX register) is the TPR in the advanced programmable interrupt controller (APIC), which is usually an external chip. When the operating system accesses this register, it must perform a relatively slow memory-mapped-I/O read or write operation. The new TPR register in x86-64 is an architectural control register, so the operating system can access it with a much faster MOV-CR8 instruction. Multitasking operating systems like Windows frequently use the TPR to block lower-priority tasks from preempting higher-priority tasks. The architectural definition of the TPR appears to be the same in both AMD64 and EM64T.

Both 64-bit ISAs implement uniform byte-size register addressing. In other words, most instructions that use 64-bit registers can manipulate register values 8, 16, 32, or 64 bits long. This helps preserve compatibility with older software and is useful for programs that manipulate smaller data—they don't have to pad the operands with zeroes.

Memory Becomes Bigger and Flatter

Memory addressing is almost identical in AMD64 and EM64T, although the first 64-bit processors from AMD and Intel don't implement the maximum number of memory addresses permitted by their 64-bit ISAs. Nor will they need to for many years, because the amount of memory they can support is mind-boggling. Intel has estimated that the PC industry needs one extra bit of memory addressing per year; at that rate, x86-64 has enough potential physical-address space to last until 2020 and enough virtual-address space to reach 2032. (To put those estimates in perspective, 2020 roughly matches Intel's original end-of-life projection for its other 64-bit architecture, IA-64, once the intended heir to the x86 throne.)

When running 64-bit software, x86-64 supports 64-bit virtual or linear memory addressing and 52-bit physical memory addressing. The actual memory-address ranges can be less than these limits, depending on the implementation. Intel says its first EM64T processors will support 48-bit virtual addressing and 36- or 40-bit physical addressing. AMD's current Opteron processors support 48-bit virtual addressing and 40-bit physical addressing.

The 64-bit virtual space (16 exabytes) is absolutely flat, banishing the clumsy memory segments that have vexed x86 programmers since the 64K-segment days of the 8086. In 64-bit mode, the program code, data, and stack all share the same memory space. However, true to character, x86-64 preserves the concept of segments in a limited way. Operating systems can still define code segments for some of their internal data structures—specifically, the FS and GS segments—and the attributes byte of the code-segment register still controls the processor's privilege level and execution mode (16, 32, or 64 bits). A 64-bit program simply ignores the other attribute bytes that set the code segment's base address and upper limit. This approach to setting privilege levels and execution modes retains a familiar x86 structure instead of introducing a whole new mechanism.

For backward compatibility, 32-bit programs still use segmented memory in their usual address space, even when running on a 64-bit operating system. Therefore, some of the flat 64-bit virtual memory will be divided into 32-bit segments, but the memory mapping is transparent to application programmers. As always, the processor and operating system dynamically map the virtual memory addresses onto physical

memory, keeping the 32- and 64-bit regions separate from each other.

With 52-bit physical-memory addressing, x86-64 supports up to 4PB (petabytes) of memory. Today's 32-bit x86 processors support up to 36 bits (Intel) or 40 bits (AMD) in physical-address extension (PAE) mode, enough for 64GB (Intel) or 1TB (terabyte; AMD) of memory. Again, to preserve familiarity, x86-64 uses the same PAE paging structures found in 32-bit x86 processors, but it defines more address bits. The page tables are the same size—512 × 8 bytes—and each entry

Feature	AMD	Intel	Notes
Architecture and Nomenclature			
64-Bit Architecture Name	AMD64	Extended Memory 64 Technology	Generic substitute: x86-64
64-Bit Execution Mode	Long mode	IA-32e	Requires a 64-bit OS
16/32-Bit Sub-Mode	Compatibility mode	Compatibility mode	64-bit OS runs 16/32-bit software
32-Bit Execution Mode	Legacy mode	Legacy mode	32-bit OS, 16/32-bit software
Mode-Switch Control	By code segment	By code segment	Segment registers define mode
New 64-Bit Instructions			
CDQE	Supported	Supported	New mnemonic for existing opcode
CMPSQ	Supported	Supported	New mnemonic for existing opcode
CMPXCHG16B	Not supported	Supported	CMPXCHG8B in AMD64
LODSQ	Supported	Supported	New mnemonic for existing opcode
MOVSQ	Supported	Supported	New mnemonic for existing opcode
MOVZX	Supported	Supported	64-bit version of existing instruction
STOSQ	Supported	Supported	New mnemonic for existing opcode
SWAPGS	Supported	Supported	Genuinely new; see Table 1
SYSCALL	Supported in all modes	Supported only in 64-bit mode	See text
SYSRET	Supported in all modes	Supported only in 64-bit mode	See text
Deleted Instructions			
SYSENTER	Supported only in legacy mode	Supported in all modes	See text
SYSEXIT	Supported only in legacy mode	Supported in all modes	See text
LAHF	Supported in all modes	Supported only in 32-bit modes	See text
SAHF	Supported in all modes	Supported only in 32-bit modes	See text
Other Instructions	27 instr invalid in 64-bit mode	27 instr invalid in 64-bit mode	Not the same 27 instructions
Register Files			
Extended General-Purpose Registers	8 x 64 bits	8 x 64 bits	Extended from 32 to 64 bits
New General-Purpose Registers	8 x 64 bits	8 x 64 bits	r8–r15
New Media Registers	8 x 128 bits	8 x 128 bits	XMM8–XMM15
Existing Media Registers	8 x 128 bits	8 x 128 bits	XMM0–XMM7 (no change)
Existing Floating-Point Registers	8 x 80 bits	8 x 80 bits	No change
Extended Instruction Pointer (RIP)	1 x 64 bits	1 x 64 bits	Extended from 32 to 64 bits
Extended Flags Register	1 x 64 bits	1 x 64 bits	Extended from 32 to 64 bits
Extended-Register Prefix	REX (4 bits, 16 opcodes)	REX (4 bits, 16 opcodes)	Allows use of new registers
Extended Feature-Enable Register	EFER	IA32_EFER	64-bit feature selection
Extended Control Registers	CR0–CR4	CR0–CR4	Extended from 32 to 64 bits
Extended Descriptor-Table Registers	GDTR, IDTR, LDTR, TR	GDTR, IDTR, LDTR, TR	Flat-memory table location
New Task-Priority Register	TPR (CR8)	TPR (CR8)	Control external interrupts
Debug Registers	8 x 64 bits	8 x 64 bits	Extended from 32 to 64 bits
Uniform Byte-Register Addressing	Yes	Yes	Access 8/16/32/64 bits in registers
Memory Addressing			
64-Bit Mode Virtual Memory	Flat 64-bit addressing	Flat 64-bit addressing	16EB (exabytes) virtual memory
64-Bit Mode Physical Memory*	Flat 52-bit addressing	Flat 52-bit addressing	4PB (petabytes) physical memory
32-Bit Mode Physical Memory*	Segmented 40-bit addressing	Segmented 36-bit addressing	AMD: 1TB (terabyte); Intel: 64GB
64-Bit Mode Page Table	512 x 8 bytes	512 x 8 bytes	Defines 52 bits instead of 36 bits
32/64-Bit Mode Page Sizes	4K, 2MB	4K, 2MB	Introduced with PAE mode
64-Bit Memory Segmentation	Single code, data, stack space	Single code, data, stack space	FS/GS segments optional
Canonical Memory Addressing	Required in 64-bit mode	Required in 64-bit mode	All unused bits must be 0 or 1
Miscellaneous Features			
New Data-Addressing Mode	RIP-relative addressing	RIP-relative addressing	64-bit mode only
Stack-Pointer Size	Extended from 32 to 64 bits	Extended from 32 to 64 bits	No prefix overrides
Virtual 8086 Mode	Supported only in legacy mode	Supported only in legacy mode	Allows 16-bit real-mode execution
Interrupt Handlers and Drivers	Must be 64-bit code	Must be 64-bit code	Exception: system mgmt interrupts
GPRs - Mode Switching	Upper 32 bits not saved	Upper 32 bits not saved	Upper 32 bits invisible in 32-bit mode

Table 3. This is a summary of the similarities and differences between AMD's AMD64 and Intel's EM64T architectures. Similarities greatly outnumber differences, so the table highlights the most important differences in purple. Other differences, such as naming conventions, are insignificant. *Implementation dependent.

For More Information

Both AMD and Intel provide their 64-bit x86 manuals in book form and online. The online versions can be downloaded free. For AMD's manuals, see *www.amd.com/us-en/Processors/DevelopWithAMD/0,,30_2252_9044,00.html#amddocs*. For Intel's manuals, see *www.intel.com/technology/64bitextensions/*.

now defines 52 memory-address bits instead of 36 or 40 bits. Page sizes also remain unchanged: 4K or 2MB.

64-Bit Compatibility Good for Industry

Miscellaneous other features are the same in both 64-bit ISAs. A new data-addressing mode, known as RIP-relative addressing, uses the extended 64-bit relative instruction pointer (RIP) as the base address for the displacement to the next instruction. Previously, only a control-transfer instruction could use a displacement based on the instruction pointer. RIP-relative addressing is useful for position-independent code libraries shared by multiple application programs.

The new x86-64 stack pointer, also extended to 64 bits, fixes the stack size and cannot be overridden by an instruction prefix or a control bit in the stack-size (SS) descriptor, as it can be in 32-bit modes.

Device drivers and interrupt handlers called by a 64-bit operating system must be written in 64-bit code, except for system-management interrupt handlers. When a handler switches contexts between 64-bit execution mode and 32-bit compatibility mode, the upper 32 bits of the 64-bit GPRs aren't preserved in 32-bit mode; they are undefined. That shouldn't be a problem, because the upper 32 bits aren't visible to 32-bit programs. Of course, a switch back into 64-bit execution mode restores the entire contents of the registers for 64-bit programs. Table 3 sums up our comparison of the 64-bit ISAs from AMD and Intel.

Both AMD and Intel deserve commendation for their work on x86-64. AMD is due the most credit for inventing the 64-bit extensions and for preserving the essential character of the x86. The temptation must have been great to purge the architecture of its historical baggage and replace it with a wholly new architecture, reflecting modern design principles. Although some critics may never forgive AMD for not grasping that once-in-a-lifetime opportunity, *MPR* agrees it was more important to retain strong connections with the x86's basic nature, for the sake of familiarity and backward compatibility. In addition, the commonality between the 64- and 32-bit ISAs should simplify microarchitecture design—an x86-64 processor can use the same instruction decoders, registers, and other resources for both instruction sets. A clean break with the past—such as Intel made with the IA-64 architecture, despite its backward compatibility with x86 software—could have split the industry and made AMD64 harder to accept, especially for Intel.

Indeed, AMD64 is such a logical extension of the x86 that Intel, had it acted first, probably would have designed a 64-bit x86 architecture not significantly different from AMD64. That likelihood, plus Microsoft's insistence that it will support only one 64-bit x86 architecture, is probably the reason EM64T and AMD64 are largely compatible. One can only imagine the internal strife that must have preceded Intel's wrenching decision to reverse-engineer EM64T from AMD64. Intel deserves credit for swallowing its pride and not plunging the industry into chaos with two mutually incompatible 64-bit x86 architectures.

Although minor incompatibilities exist, we don't believe Intel will use them as a wedge between EM64T and AMD64. The fact is that Intel remains well positioned to compete with AMD on other grounds. Intel is currently about a year ahead of AMD in fabrication technology, with 90nm processors already shipping in quantity; AMD probably won't deliver 90nm chips in quantity until late this year or early 2005. (Although Intel's first 90nm processors aren't the big leap over 130nm processors usually expected, that situation will soon be rectified.) Intel's latest Prescott Pentium 4 has SSE3 media extensions; AMD is perhaps a year away from supporting SSE3. Intel's recent processors can do simultaneous multithreading (Hyper-Threading technology); AMD's cannot. Intel's Pentium M processors bring strong performance and lower power consumption to the fast-growing mobile-PC market; AMD lacks a comparable CPU microarchitecture especially designed for low power. Intel's IA-64 still enjoys a performance advantage over x86-64, which is important for high-end servers. And, of course, Intel's famous marketing muscle virtually ensures that EM64T processors will eventually outsell AMD64 processors, despite AMD64's earlier sprint from the starting gate.

None of this should detract from AMD's accomplishment, however. Seemingly outflanked by IA-64 and relegated to playing a secondary role for two decades, AMD has pushed the indomitable x86 architecture into a new frontier of 64-bit computing and has done so in a manner so logical and inevitable that even Intel is joining the ride. AMD's coup won't lead to a reversal of fortune—Intel will remain the market leader—but the momentary role-reversal is a strange sight to behold. ◇