



The AMD x86-64 Architecture

Extending the x86 to 64 bits

Kevin McGrath, Fellow
Dave Christie, Fellow
Advanced Micro Devices

Agenda



- Why 64 bits?
- Goals of x86-64 technology
- Features
 - New modes
 - 64-bit integer operations and data path
 - Extra registers
 - 64-bit addressing
- Effect on compiler-generated code quality

Why 64 bits?



- Driven by apps needing large amounts of memory
 - CAD tools, large databases, simulations
- 64-bit integer arithmetic
 - Security and encryption applications
- Why extend x86 to 64 bits?
 - X86 is the most widely installed instruction set in the world
 - Delivers 64-bit advantages while providing full x86 compatibility
 - Doesn't require a completely new tool chain

Design Goals for x86-64 Technology



- Straightforward extensions for 64 bits
 - Minimize architectural divergences
 - Maintain consistency with existing architecture
 - Minimize instruction set encoding changes
- Add Integer and SSE registers
- Compatibility with existing 32-bit applications
- Eliminate unused/underutilized arcane x86 features within the context of 64-bit mode
- Straightforward implementation & verification

Key Features



- Processor is fully compatible with existing x86 modes
- Architectural support for 64 bits of virtual address space and 52 bits of physical address space
 - Implementations may support less
- 64-bit mode does not use segmentation
 - Flat addressing
- 64-bit integer operations
- Double the number of Integer and SSE registers
 - 16 General Purpose Registers, 16 SSE Registers
 - Integer GPRs extended to 64 bits

Long Mode Overview



- Long Mode consists of 2 sub-modes
 - 64-bit mode
 - Compatibility mode
- Long Mode is enabled by a global control bit (LME)
- When LME=0, CPU is a standard 32-bit processor

LME	code segment attribute		Mode
	L bit	D bit	
0	x	0	Legacy 16-bit mode
0	x	1	Legacy 32-bit mode
1	0	0	Compatibility 16-bit mode
1	0	1	Compatibility 32-bit mode
1	1	0	64-bit mode
1	1	1	Reserved

64-bit mode



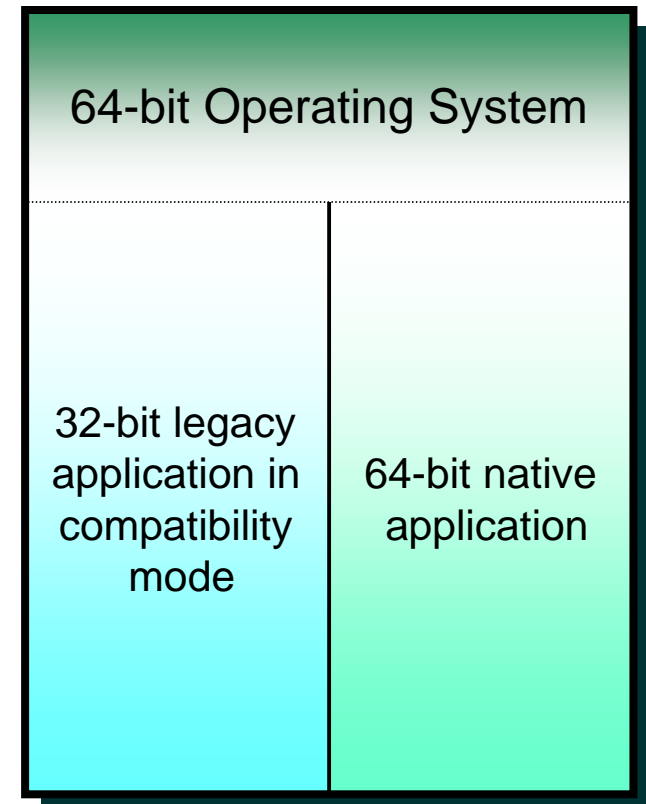
- Default data size is 32 bits
 - Override to 64 bits using new REX prefix
 - Override to 16 bits using legacy operation size prefix (66h)
- Default address size is 64 bits
 - Pointers are 64 bits

Prefix Type	None	REX	66h
Operand Size	32	64	16

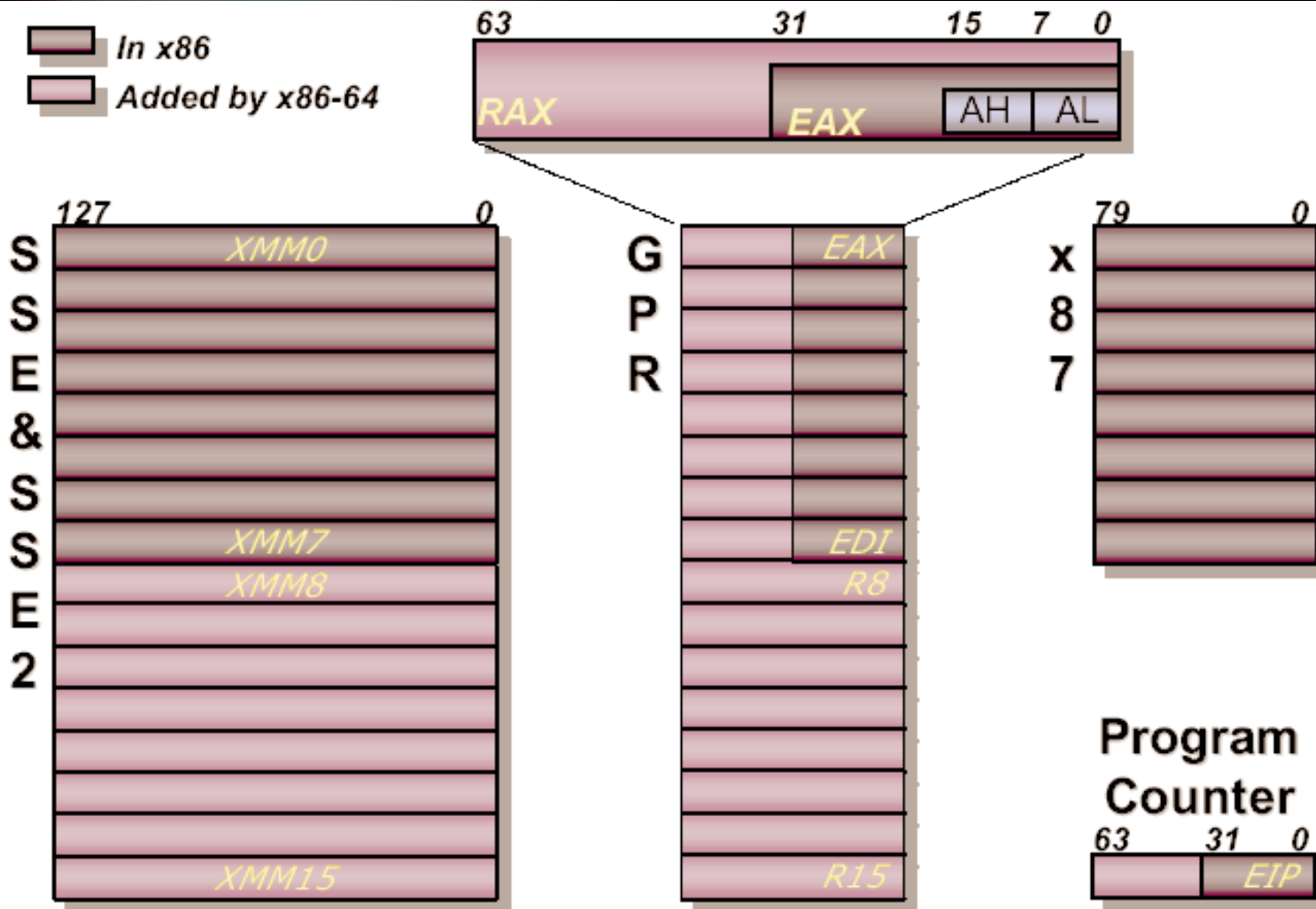
Compatibility Mode



- Provides a mode where existing applications can run unchanged under Long Mode
- Selected on a code-segment basis (CS.L=0)
 - Uses far transfer rather than a full mode switch
 - Faster than mode switch
- Application-level code runs unchanged
 - Legacy segmentation
 - Legacy address and data size defaults
- System aspects use native 64-bit mode semantics
 - Interrupts and exceptions use Long Mode handling
 - Paging aspects use Long Mode semantics



x86-64 Programmer's Model

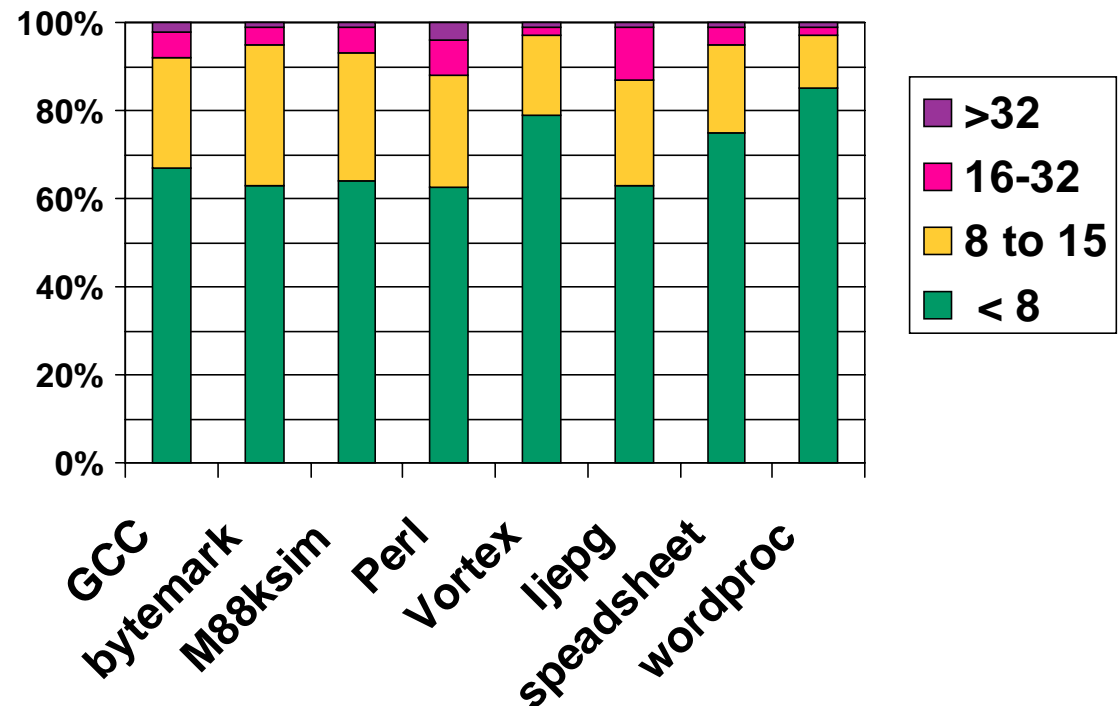


REX (Register Extension)

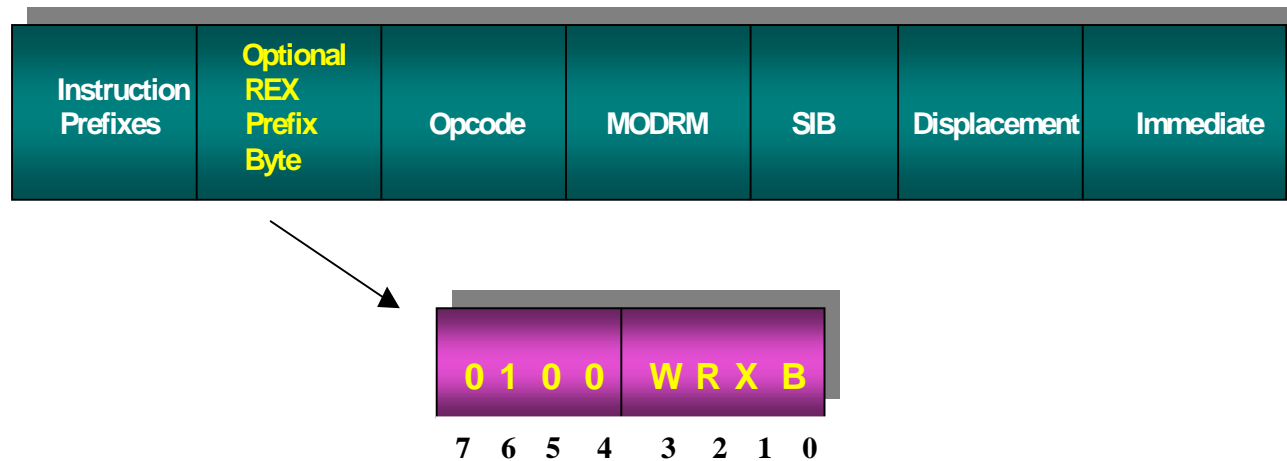


- Doubles the number of GPRs and SSE registers
- 8 new GPRs
 - R8 through R15
- 8 new SSE registers
 - XMM8 through XMM15

**% of Functions in Typical Applications
Requiring N Registers**



REX prefix byte



- Additional registers encoded without altering existing instruction format
- Optional REX prefix specifies 64-bit operation size override
 - Plus 3 additional register encoding bits
- REX is actually a family of 16 prefixes (40-4F)
- Average instruction length in 64-bit mode increased by 0.4 bytes

Segments and Segmentation



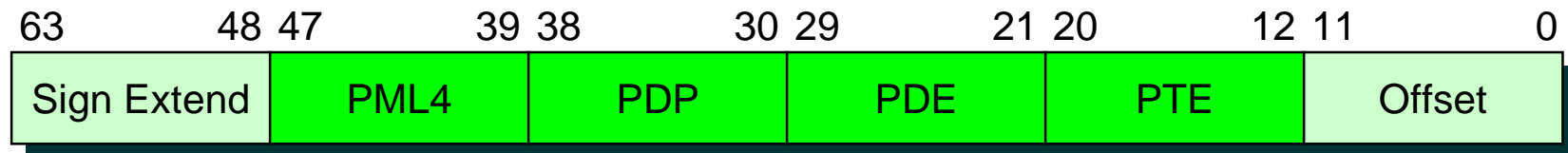
- 64-bit mode presents a flat, unsegmented virtual address space
 - The legacy x86 segmentation scheme is disabled in 64-bit mode
- Code Segments still exist in Long Mode
 - Needed to specify default mode (16-, 32- or 64-bit) and execution privilege level (CPL)
 - Also means that existing privilege level and checking mechanisms are retained
- Switch between 64-bit mode and Compatibility Mode accomplished via normal Far Transfer instructions
 - CALLF, RETF, JMPF, IRET, INT

Addressing and Page Tables



- x86-64 architecture supports 64-bit virtual addressing
 - Implementations may support less
 - Hammer supports 48 bit VA
- New level added to paging structures: PL4
 - leverages existing mechanism to extend page map entries to handle 64-bit addresses

The 48-bit virtual address is broken into 4 fields to index into the 4-level paging structure.



- GCC port alpha quality since Feb '01
 - Compiler generating alpha quality code in 50 man-months
 - Linux kernel ported in 60 man-months
 - Tool chain was straightforward port
- SpecInt2000 code quality, 64bits vs. 32 bits (using GCC 3.1.1)
 - average instruction length increased to 3.8 from 3.4 bytes
 - dynamic instruction count decreased by 10%
 - dynamic load count decreased by 26%
 - number of loads forwarded from recent stores substantially reduced
 - dynamic store count decreased by 36%
 - back to back register dependencies decreased by 10%

Conclusion



- New operand/address sizes, rather than new instructions
- Extend existing mechanisms, rather than creating new ones
 - New operand and address sizes build upon existing code segment and prefixes
 - New paging extensions build upon existing PAE mode
- CPU improvements accelerate both 32-bit and 64-bit performance
- An x86-64 design only adds 2-3% in die size vs. an equivalent x86-32 implementation
 - Larger data path and register files
 - Larger number of bits in cache and TLB addressing
 - Larger number of bits in branch prediction and return address stack
- Innovation with compatibility
 - users transition to 64-bit code on their own timeframe

Trademark Attribution



AMD, the AMD Arrow Logo and combinations thereof are trademarks of Advanced Micro Devices, Inc. Other product names used in this presentation are for identification purposes only and may be trademarks of their respective companies.