x86-64 Architecture & Software Porting

Rich Brunner

Senior Member of Technical Staff, AMD

Agenda

AMD DEVELOPER symposium 2002

- Instruction-Set Architecture
- Operating-System Model
- Linux x86-64 Application Model
- Linux x86-64 Development Tools
- Tricks
- Linux x86-64 "Gotchas"

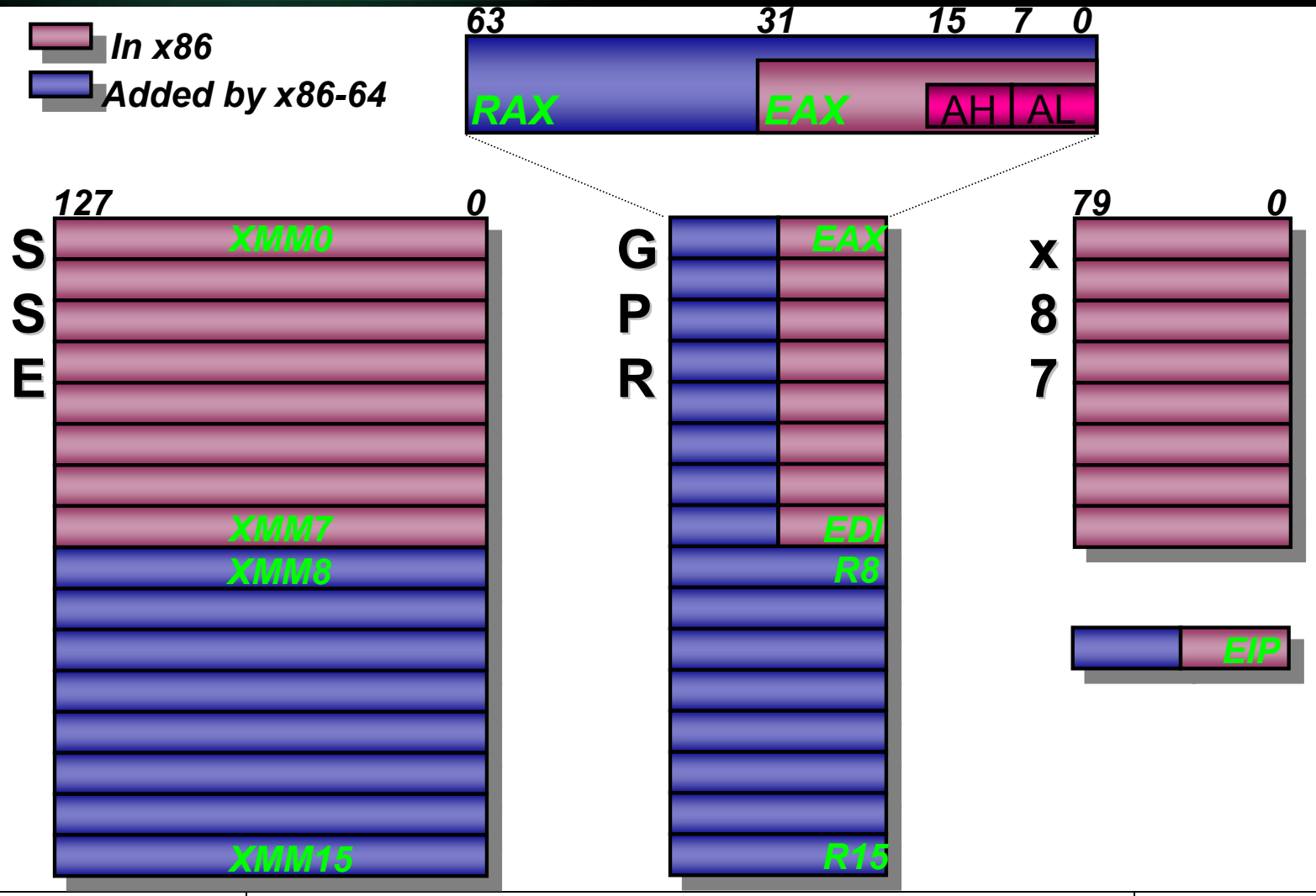
Instruction Set Architecture x86-64

x86-64 Architecture Review

AMD DEVELOPER symposium 2002

- AMD took the x86 architecture and extended it to 64-bits to beget the x86-64 architecture.
  - Processor running in 32-bit x86 Legacy mode executes today's 32-bit operating systems and application software.
  - Processor running in "Long mode" executes a 64-bit OS that can run applications in either 32-bit or 64-bit mode.
  - Only 64-bit mode gives access to 64-bit addressing and 64-bit registers.
- Extensions are simple and compatible, so the processor can support both x86 and x86-64 at full speed & performance.
  - All Customers get 32-bit performance & 32-bit compatibility
  - Customers can move to 64-bit addressing and data types w/o giving up 32-bit compatibility when needed.
  - Leverages key PC infrastructure rather than needing to re-invent it.

x86-64 Programmer's Model



"64-bit Mode" Operation

AMD DEVELOPER symposium 2002

- New instructions – only two
  - MOVSXD: Move sign extended double to quad
  - SWAPGS: Kernel mode instr to reduce overhead in interrupt handlers
- New override allows naming 16 GP and 16 SSE registers
  - Only a single override byte per-instruction is needed to designate extended registers regardless of how many are used by the instruction
- Stack alignment maintained at 64-bits
- Architectural support for 64 bits of virtual address space and 52 bits of physical address space
  - Implementations may support less
- Paging mechanism extended to provide 64-bit addressing
  - Four-level page table
  - Page-Table entries are a simple extension of x86 PAE-formatted entries
- Interrupts and exceptions create 64-bit state
  - SMI affected even in 32-bit mode
- Some redundant instruction encodings reclaimed
- 64-bit Mode Does Not Use Segmentation -- Flat Addressing

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 Mode	Code Segment Attribute		Mode
	CS.L	CS.D	
0	X	0	Legacy 16-bit mode
	X	1	Legacy 32-bit Mode
1	0	0	Compatibility 16-bit mode
	0	1	Compatibility 32-bit mode
1	1	0	64-bit mode
	1	1	Reserved

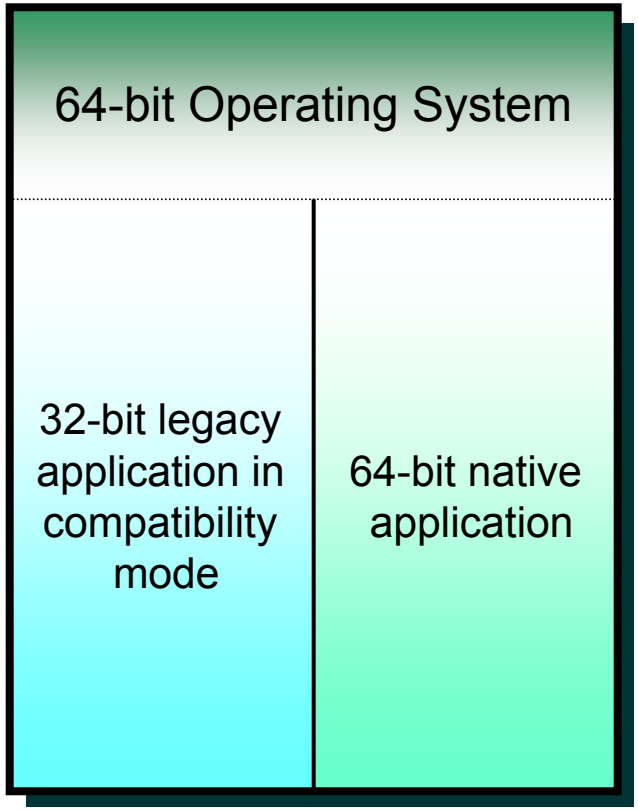
64-bit mode

- 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

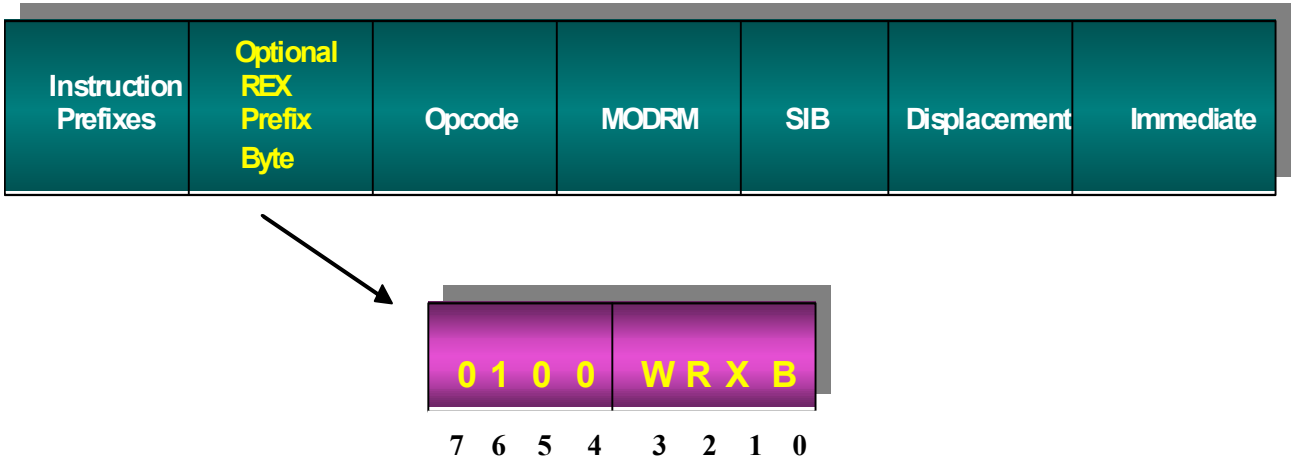
Prefix Type	None	REX	66h
Operand Size	**32**	**64**	**16**

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



REX prefix byte



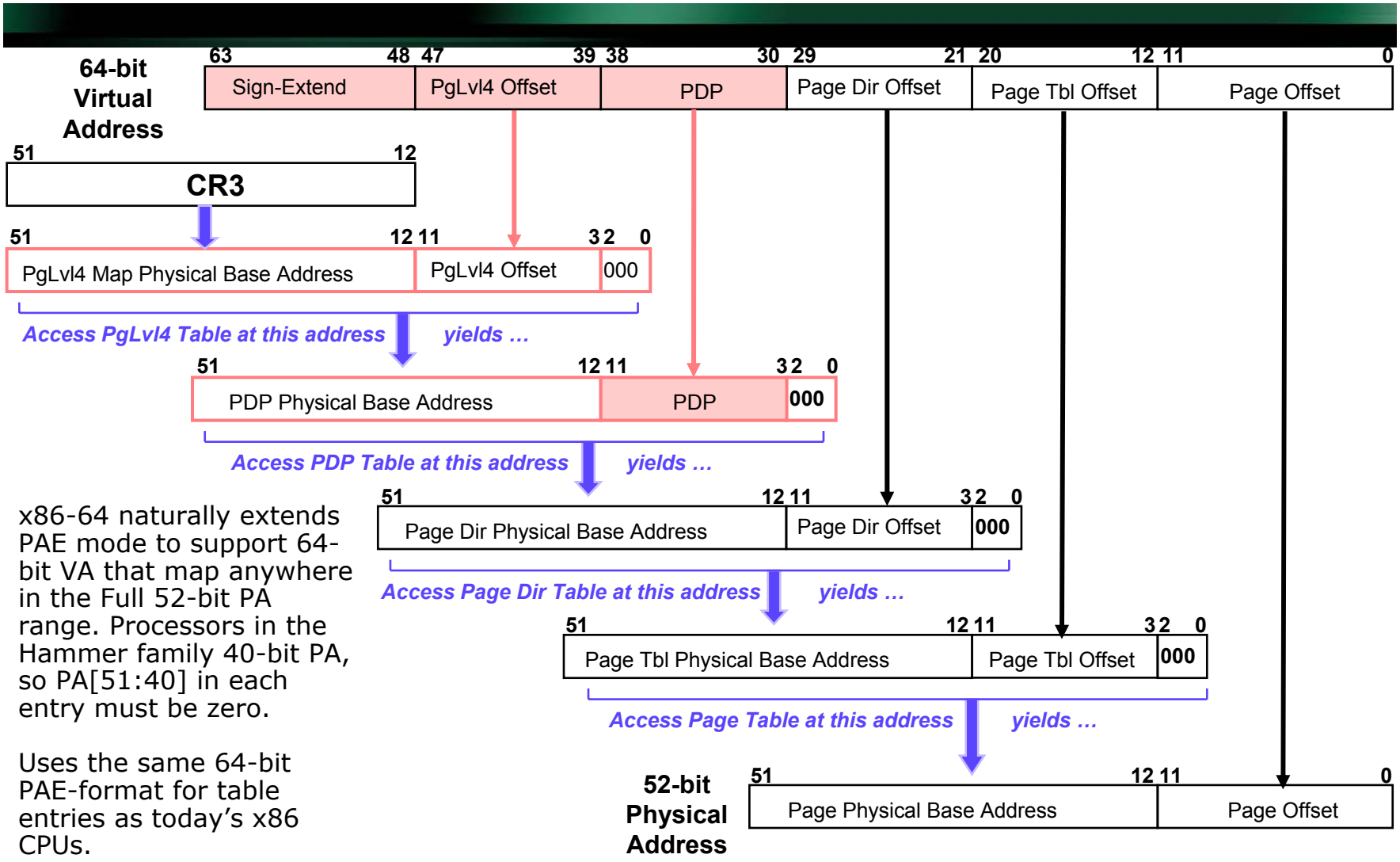
- Additional registers encoded without altering existing instruction format
- Optional REX prefix specifies 64-bit operation size override
  - Plus 3 additional register encoding bits
- REX is actually a family of 16 prefixes (40-4F)
- Average instruction length in 64-bit mode increased by 0.4 bytes

Segments and Segmentation

AMD DEVELOPER symposium 2002

- 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

x86-64 64-bit Virtual Addressing



x86-64 naturally extends PAE mode to support 64-bit VA that map anywhere in the Full 52-bit PA range. Processors in the Hammer family 40-bit PA, so PA[51:40] in each entry must be zero.

Uses the same 64-bit PAE-format for table entries as today's x86 CPUs.

Architectural Feedback

- Instruction set does not affect performance
  - RISC lost the RISC-CISC war
  - Compatibility is the prime motivator for ISA. x86 installed base makes it the most compelling choice for ISA
  - x86 code density is outstanding
- Following information is wrt x86 and derived from compiling SpecINT2000*
  - Code size is up about 5% -> Mostly due to 64-bit literals
  - Instruction count is down about 15%
    - Additional registers really paying off
    - Many spill/fill memory references eliminated
    - Call-Exit sequences vastly improved

* = on a current pre-release x86-64 compiler

Architectural Feedback (cont.)

AMD DEVELOPER symposium 2002

- Reduced instruction count and increased IPC mean substantial performance gains
  - "Hammer" IPC improves about 5%
  - x86-64 instruction count down about 15%
  - Net improvement about 20%
  - Your mileage will vary based on application
- IA-64 feedback is exactly opposite: Instruction count is up; IPC down

Operating System Model

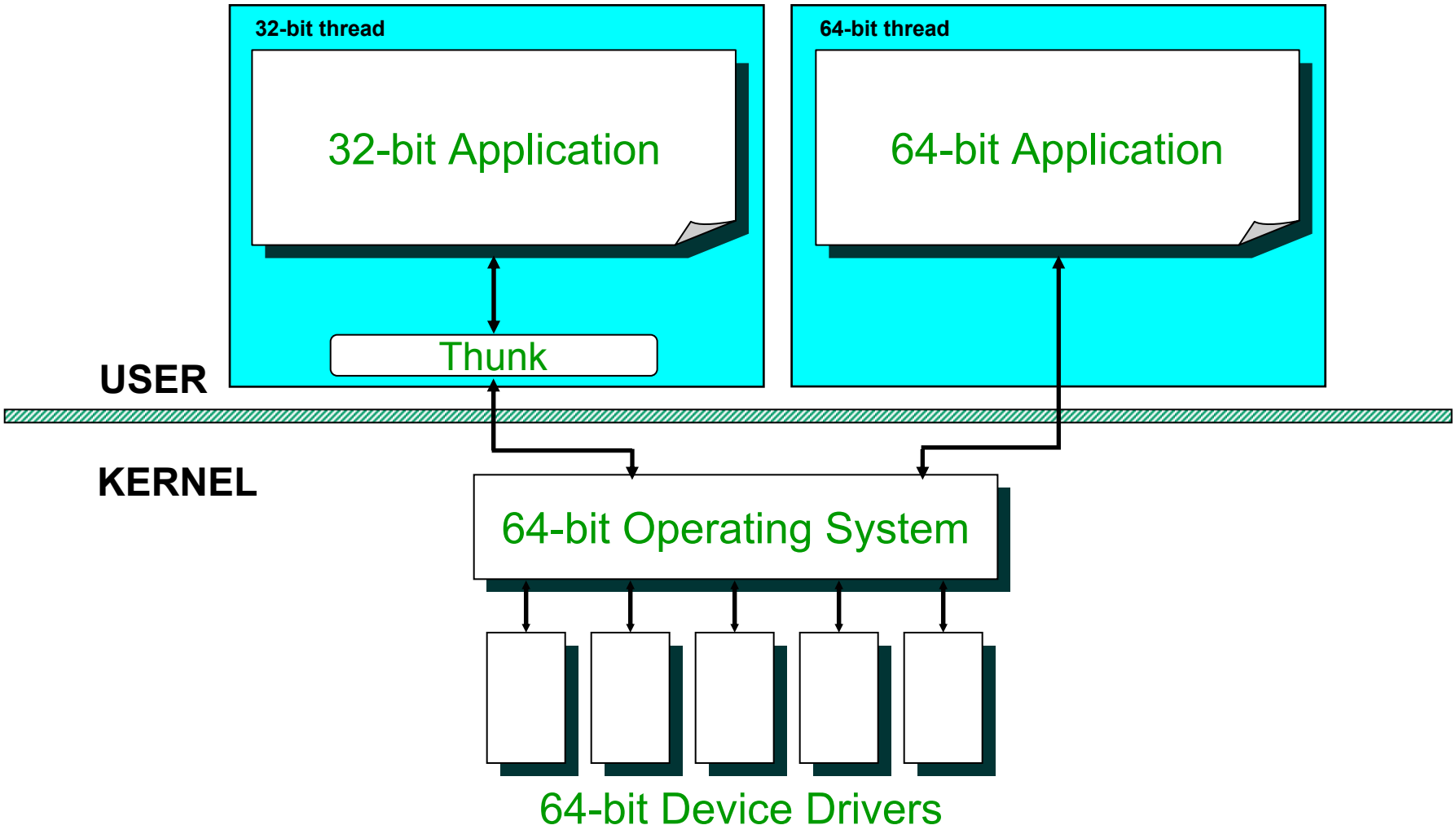
64-bit Considerations

- Hammer BIOS is standard x86 32-bit code
  - Transfer to 64-bit operation occurs under OS load/startup control
  - No additional firmware requirements. Validation cycle only
- 32-bit applications
  - Processor core provides full x86 compatibility
    - At full speed
    - No support for v86
  - OS provides thunking layer at kernel-call boundary
- 64-bit applications require 64-bit OS
  - All brand new; none exist today
- Device drivers
  - 64-bit OS requires 64-bit device drivers
  - Distributions include many device drivers
  - Firmware call-backs no longer supported
- 64-bit tools are new
  - Simulation
  - Compilers, linkers, libraries
- Debuggers, performance analyzers

64-bit Computing Strategy - x86-64

- Legacy Mode
  - Hammer processors are designed to run any 32-bit legacy O/S with leading-edge performance
  - Designed to be fully compatible with prior 32-bit gens: Drivers, OS, BIOS
- Compatibility Mode
  - Under control of a 64-bit O/S, existing 32-bit applications can be run with leading-edge performance
  - No application recompile required, no emulation layer
- 64-bit Mode
  - Desired applications can be written/ported to leverage the full 64-bit capabilities of x86-64
  - Migrate only where warranted and as users demand
  - Perhaps as few as 1% of applications need to be in this mode

64-bit OS & Application Interaction



Compatibility Thunking Layer

AMD DEVELOPER symposium 2002

- A library integral to operating system
  - Transparent to end-user
- Resides within a 32-bit process established by the 64-bit OS to run 32-bit application
- 32-bit application is dynamically linked to Thunking Layer
- Thunking layer implements all 32-bit kernel calls
  - Translates parameters as necessary
  - Calls 64-bit kernel
  - Translates results as necessary
- Well understood technology implemented in Linux & Windows®

Linux x86-64 Application Model

x86 32-bit Binaries on x86-64 Linux (1)

- Work just fine and at full speed
- System Call Thunking provides the necessary infrastructure.
  - Using 64-bit libs requires inter-process communication and is not preferred route.
- 32-bit libraries remain in */lib.
- Same x86 32-bit SYS V ABI. Same 32-bit Development tools.
  - Floating-point model is x87.
  - SSE, SSE2, MMX™, and 3DNow!™ technologies are supported
  - Inline asm still works, etc …
- 64-bit OS is more efficient & faster than 32-bit OS on same hardware, so total solution of 64-bit OS & 32-bit app expected to run faster.

Native Application on Linux x86-64

AMD DEVELOPER symposium 2002

- Still uses std libraries (glibc, etc) with logical extensions for 64-bit datatypes
  - Using 32-bit libs requires inter-process communication and is not preferred .
- 64-bit libraries are in */lib64 instead of */lib.
- In-line asm allowed in “C"
- New, well-documented 64-bit ABI

Native Application on x86-64 Linux

AMD DEVELOPER symposium 2002

- Floating-point model uses SSE and SSE2 for highest performance and improved context switch time
  - 16 flat, 128-bit registers instead of 8-entry FP stack
  - Support for SSE/SSE2 compiler intrinsics
  - x87 is supported for 64-bit Applications thru "long double" data type.
  - All math routines in glibc will use just SSE/SSE2; all routines are expected to match or exceed the performance of the legacy x87 routines

x86-64 Linux ABI: Scalar Types

AMD DEVELOPER symposium 2002

Type	C	sizeof	Byte Align
Integer	_Bool/bool	1	1
	char signed char unsigned char	1	1
	short signed short unsigned short	2	2
	int signed int unsigned int enum	4	4
	long signed long long long signed long long	8	8

x86-64 Linux ABI: Scalar Types

AMD DEVELOPER symposium 2002

Type	C	sizeof	Byte Align
Integer	unsigned long unsigned long long	8	8
	__int128 signed __int128 unsigned __int128	16	16
Pointer	any-type * any-type (*)()	8	8
FP	float	4	4
	double	8	8
	long double	16	16
	__float128	16	16
Packed	__m64	8	8
	__m128	16	16

x86-64 Linux ABI: Register Calling Conventions

AMD DEVELOPER symposium 2002

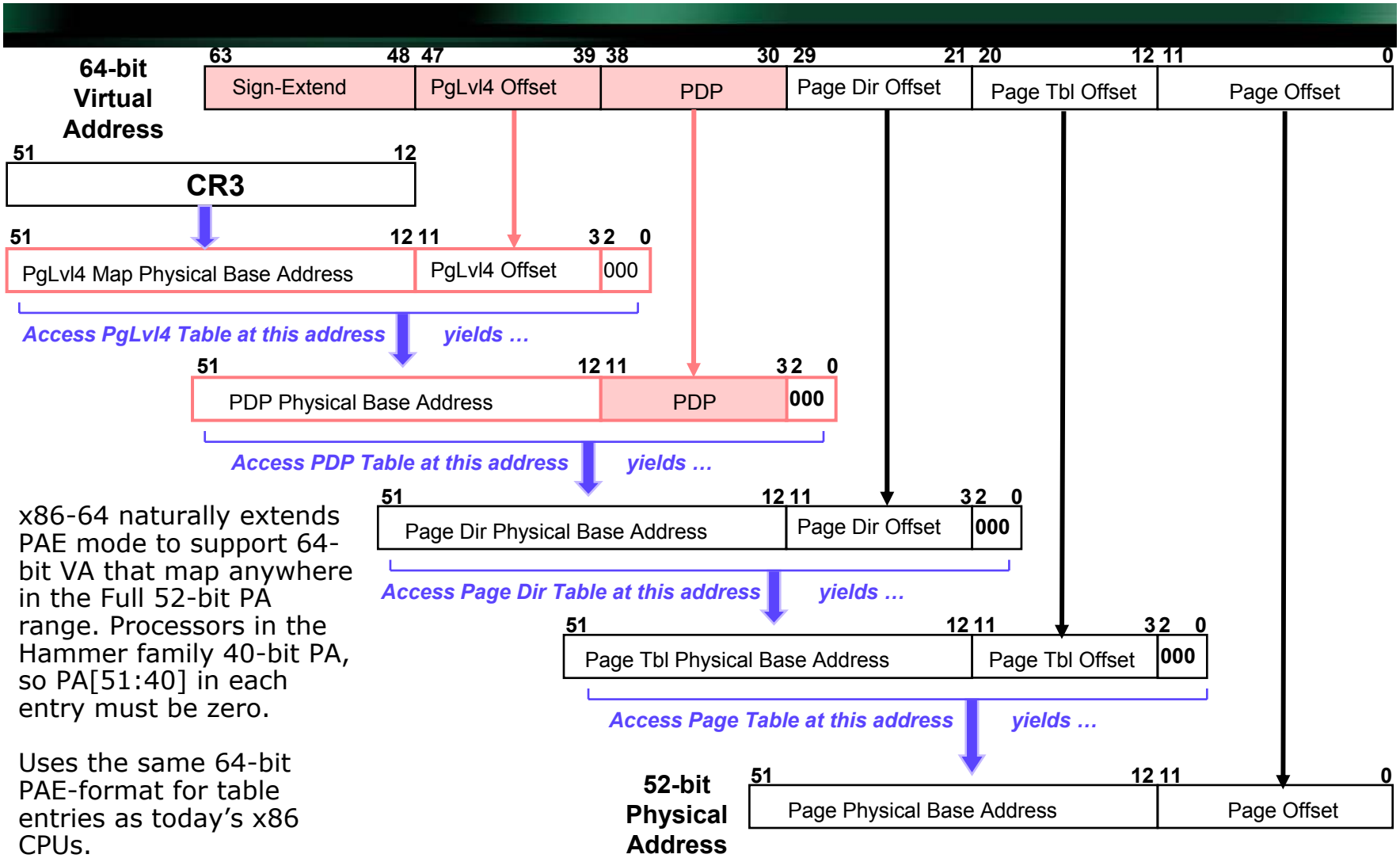
Register	Preserved across Call ?	Usage
%rax	No	Temporary register; with var args indicates number of SSE regs used. Return value reg.
%rbx	Yes	callee-saved register; Optionally base pointer
%rdi, %rsi, %rdx, %rcx, %r8, %r9	No	*1st thru 5th arguments*
%r10, %r11	No	Temp Registers
%r12:%r15	Yes	Callee-Saved
%rbp	Yes	Callee-saved, may be used as FP
%rsp	Yes	Stack Pointer

x86-64 Linux ABI: Register Calling Conventions

AMD DEVELOPER symposium 2002

Register	Preserved across Call ?	Usage
%xmm0:%xmm1	No	used to pass and return FP args
%xmm2:%xmm7	No	used to pass FP args
%xmm8:%xmm15	No	temporary registers
%mmx0:%mmx7	No	temporary registers
%st0	No	temporary register; returns long double arguments
%st1–%st7	No	temporary registers
%fs	No	Thread specific data register

x86-64 64-bit Virtual Addressing



x86-64 naturally extends PAE mode to support 64-bit VA that map anywhere in the Full 52-bit PA range. Processors in the Hammer family 40-bit PA, so PA[51:40] in each entry must be zero.

Uses the same 64-bit PAE-format for table entries as today's x86 CPUs.

Virtual Address Space Mapping

- Each PML4 slot is 512GB of Virtual Address Space.
  - There are 39 address bits [38:0] in the 3-level page table
- The User thread gets 512GB (aka PML4 slot 0).
  - But shared mappings start at 0x0000,002A,9555,6000
  - Usually somewhat less because shared mappings start at (512GB)/3.
- The kernel keeps all the rest for itself.
  - Kernel supports up to 507 PML4 slots for ~ 253TB VA.
- 32-bit Apps constrained to 3.5GB address space

Virtual Address Space Mapping (cont.)

```
0000,0000,0000,0000 - 0000,008F,FFFF,FFFF: user range
0000,0090,0000,0000 - 0000,00FF,0000,0000: reserved to catch bad
                                           pointers
0000,0100,0000,0000 - 0000,78ff,ffff,ffff: ~120TB kernel direct
                                           mapping of all
  Phys Mem
0000,7900,0000,0000 - ffff,fd00,ffff,ffff: reserved
ffff,fe00,0000,0000 - ffff,feff,ffff,ffff: pci mappings
ffff,ff00,0000,0000 - ffff,ff7f,ffff,ffff: vmalloc/ioremap area
ffff,ffff,8000,0000 - ffff,ffff,8fff,ffff: Kernel text virtual mapping
ffff,ffff,a000,0000 - ffff,ffff,ff5f,ffff: loadable kernel modules
ffff,ffff,ff60,0000 - ffff,ffff,ffdf,ffff: vsyscalls

Note some items have to be w/in 32-bit signed range
```

Linux x86-64 Development Tools

Linux x86-64 Tools

AMD DEVELOPER symposium 2002

- Many of the tools in the GNU C Compiler suite have new switches for x86-64 use, so that one tool can compile 32-bit or 64-bit code.
- When tools cannot handle both modes in a single application binary, two different binaries are usually provided.
- "as" - GNU assembler. Handles 32-bit and 64-bit code.
  - "--64": Default, assemble x86-64 assembly into 64-bit code.
  - "--32": assemble i386 assembly into 32-bit code.
- "gcc" - GNU C compiler. Handles 32-bit and 64-bit code.
  - "-m64": Default, compile to 64-bit, x86-64 code.
  - "-m32": compile to 32-bit, x86 code.
- "g++" - GNU C++ compiler. Handles 32-bit and 64-bit code.
  - "-m64": Default, compile to 64-bit, x86-64 code.
  - "-m32": compile to 32-bit, x86 code.

Linux x86-64 Tools (cont.)

AMD DEVELOPER symposium 2002

- g77 - GNU Fortran77 compiler.
  - Handles 32-bit and 64-bit code.
  - "-m64": Default, compile to 64-bit, x86-64 code.
  - "-m32": compile to 32-bit, x86 code.
- ld - GNU linker.
  - Handles 32-bit and 64-bit code.
  - Invoking directly is discouraged; linking should be performed thru gcc/g++/g77.
  - Default: Produce elf64-x86-64 64-bit binaries.
  - "-m elf_i386": Produce 32-bit i386 binaries.
- ar - GNU archive program.
  - Produces 32-bit and 64-bit libraries, depending on what files are passed into it.
  - Libraries containing both 32-bit and 64-bit code must not be created.

Linux x86-64 Tools (cont.)

- nm - GNU program to list symbols in an object file.
  - Handles 32-bit & 64-bit object files w/o needing additional flags.
- gdb - GNU symbolic debugger.
  - For 64-bit x86-64 binaries only.
- gdb32 - GNU symbolic debugger.
  - For 32-bit i386 binaries only.
- strace - Captures list of systems calls
  - made by 64-bit x86-64 app.
- strace32 - Captures list of systems calls
  - made by 32-bit x86 app.
- ltrace - Captures list of library function calls
  - made by 64-bit x86-64 app.
  - No corresponding tool for 32-bit apps.

Linux x86-64 Tools (cont.)

AMD DEVELOPER symposium 2002

- gdb-xfreemod - Experimental version of gdb
  - Includes support for XFree86 modules.
  - Available in package form on companion CD.
- Objdump - Dumps object files.
  - Handles 32-bit & 64-bit object files w/o needing additional flags.
  - Examines ELF header to determine architecture.
- linux32 <app> - Sets personality to "i686".
  - All children of the app will also inherit the personality.
- uname -m
  - Usually returns the normal "linux" personality "x86_64" -- even for 32-bit binaries.
  - However, some shell scripts check the architecture looking for "i686" and will not work when they see "x86_64".
  - This can be forced by doing "linux32 <shell_script>"

Useful GCC switches

AMD DEVELOPER symposium 2002

- -m32
  - "-m32": Compile to 32-bit, x86 code.
- -m64
  - "-m64": Default, compile to 64-bit, x86-64 code.
- -O2
  - GCC performs nearly all supported opts that do not involve a space-speed trade off.
  - Turns on all opts except for loop unrolling, function inlining, and register renaming.
- -O3
  - -O3 turns on all optimizations specified by -O2 and also turns on the -finline-functions and -frename-registers options.
- -O0
  - Do not optimize.

Useful GCC switches (cont.)

- -g
  - Produce debugging information in DWARF. GCC allows you to use -g with -O.
- -ffast-math
  - Sets -fno-math-errno, -funsafe-math-optimizations, and -fno-trapping-math. Defines "__FAST_MATH__".
- -Wa,option
  - Pass option as an option to the assembler. If option contains commas, it is split into multiple options at the commas.
- -Wa,adhl=filename
  - -ad omit debugging directives
  - -ah include high-level source
  - -al include assembly
  - -g -Wa,-adhl=filename -- generates a source/assembler listing

Useful GCC switches (cont.)

- Wproto
  - Finds prototype and 64bit problems.
- -mcmodel=kernel
  - Not documented in man page. Necessary switch for building kernel modules; they will crash upon loading without it.
- -fPIC
  - See section on PIC addressing

Tricks

How to tell a 64-bit App from a 32-bit App

- rpm -qia | grep -i i386
  - Not many 32-bit apps in this distribution …
- Cat /proc/<pid>/maps
  - If the map references "lib64", then it is a 64-bit app
  - If the map doesn't reference "lib64", then it is a 32-bit app.
- uname -m
  - Returns "x86_64" normally.
  - With linux32, can return "i686"

32-bit Thread

```
% cat /proc/`ps --no-headers -o pid -C duh32`/maps

0000000008048000-0000000008049000 r-xp 0000000000000000 03:03 136399
  /home/rbrunner/proj/porting/duh32
0000000008049000-000000000804a000 rwxp 0000000000000000 03:03 136399
  /home/rbrunner/proj/porting/duh32
0000000040000000-0000000040013000 r-xp 0000000000000000 03:03 28949      /lib/ld-
  2.2.5.so
0000000040013000-0000000040014000 rwxp 0000000000012000 03:03 28949      /lib/ld-
  2.2.5.so
0000000040014000-0000000040015000 rwxp 0000000000000000 00:00 0
000000004002b000-0000000040141000 r-xp 0000000000000000 03:03 28955
  /lib/libc.so.6
0000000040141000-0000000040147000 rwxp 0000000000115000 03:03 28955
  /lib/libc.so.6
0000000040147000-000000004014b000 rwxp 0000000000000000 00:00 0
00000000ffffd000-00000000fffff000 rwxp ffffffffffffe000 00:00 0
```

64-bit Thread …

```
% cat /proc/`ps --no-headers -o pid -C duh64`/maps

0000000000400000-0000000000401000 r-xp 0000000000000000 03:03 136400
  /home/rbrunner/proj/porting/duh64
0000000000500000-0000000000501000 rw-p 0000000000000000 03:03 136400
  /home/rbrunner/proj/porting/duh64
0000002a95556000-0000002a95568000 r-xp 0000000000000000 03:03 20316
  /lib64/ld-2.2.5.so
0000002a95568000-0000002a9556a000 rw-p 0000000000000000 00:00 0
0000002a95667000-0000002a9566a000 rw-p 0000000000011000 03:03 20316
  /lib64/ld-2.2.5.so
0000002a9566a000-0000002a95773000 r-xp 0000000000000000 03:03 20321
  /lib64/libc.so.6
0000002a95773000-0000002a9586a000 ---p 0000000000109000 03:03 20321
  /lib64/libc.so.6
0000002a9586a000-0000002a95891000 rw-p 0000000000100000 03:03 20321
  /lib64/libc.so.6
0000002a95891000-0000002a95897000 rw-p 0000000000000000 00:00 0
0000007fbfffe000-0000007fc0000000 rwxp ffffffffffffe000 00:00 0
```

Linux x86-64 Gotchas

Prototypes must be correct when using stdargs

AMD DEVELOPER symposium 2002

- Some programs have inconsistent prototypes over multiple source code files.
- By convention, stdargs and varargs use the %al register to indicate the number of floating-point arguments.
- The entry point of a function using stdargs or varargs, expects the %al register to be initialized properly.
- If the function prototype used by caller doesn't include the "(...)", gcc won't initialize %al as part of the call.
  - And a crash and/or strange behavior will occur after the call.
- glibc defines some functions unexpectedly as stdarg (e.g. ptrace, fcntl, open) and expects %al to be correct.
  - So it is essential that these have correct prototypes.
  - Best done by including the correct glibc header.

Prototypes must be correct when using stdargs (cont.)

- How to check:
  - Make sure your prototypes are correct and consistent. Look for prototypes that are not declared in shared include files and verify that they are correct.
  - Run lclint with only function call checking enabled over the whole program to check for consistent prototypes.
  - The gcc protoize tool may also be used to generate a global prototype file that could be included in every file using gcc's -include option.
  - You should also compile with the gcc "-Wall" or "-Wmissing-prototypes" switches and check for the warnings.

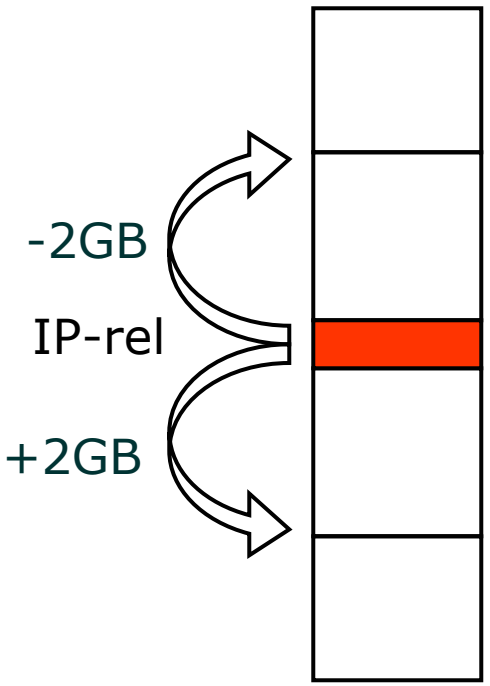
Correct return type declarations for pointers

- Functions returning pointers must be prototyped to return a pointer.
- When the return value is not explicitly declared, the function is assumed to return a 32-bit int.
  - Which leads to a truncation of the pointer and a crash on accessing the pointer.
  - Common victims of this are standard library functions like strerror or malloc when their include file is not included.
- How to check:
  - Enable -Wall in gcc and look for warnings for undeclared functions.

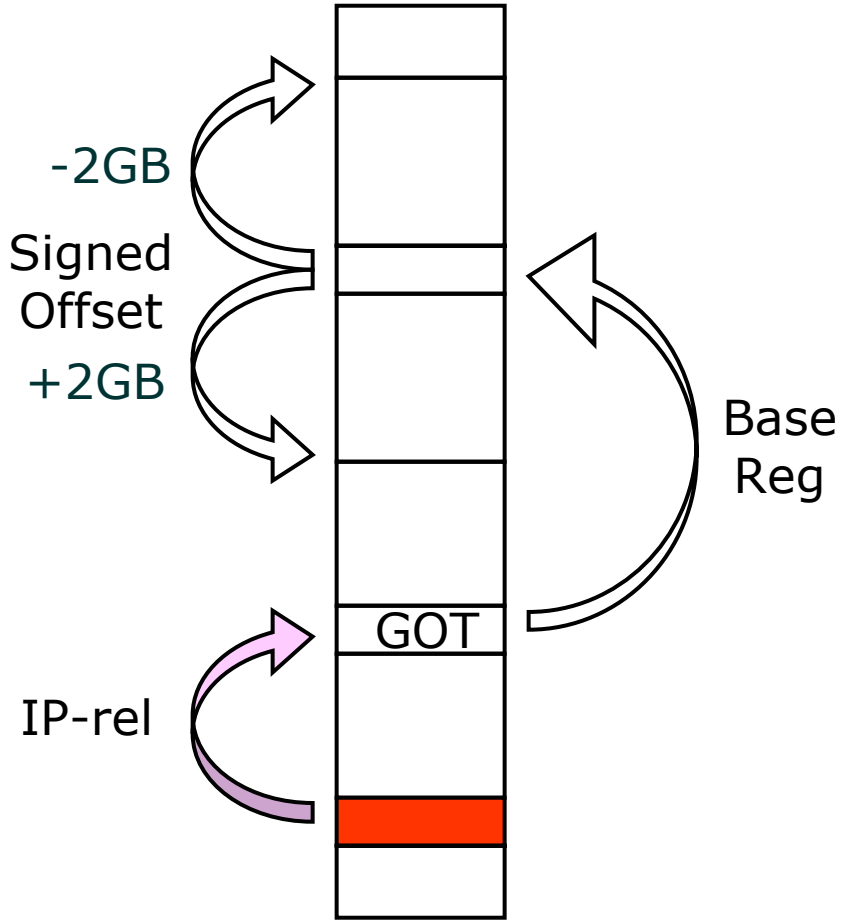
Shared libraries must be compiled with -fPIC

- i386 and a few other ports tolerate shared libraries that are not compiled with -fPIC.
- Compiling 64-bit x86-64 libraries without -fPIC leads to crashes when accessing external symbols to the shared library.
  - e.g. symbols declared in the main program.
  - This is because only 32-bit relocations are used for referencing the symbols
  - But the shared library is loaded more than 32 bits away from the main program.

Shared Lib Access to Global Symbol



Default



-fPIC

Shared libraries must be compiled with -fPIC (cont.)

- Don't rely on 32-bit relocations sometimes working between shared libraries when they are loaded at startup.
  - Dumb Luck: Because the shared libraries are within reach of 32-bit offset
  - But it may fail for shared libraries that are loaded later with dlopen() for example.
- This problem only affects 64-bit libraries. Libraries compiled for 32-bit programs may continue to be compiled w/o -fPIC.
- How to check:
  - The linker (ld) will complain if it encounters a library not compiled with -fPIC. It will also fail to link against the library.
  - If a library fails to link, you can check it by running `readelf -r` and `nm` on it.
  - Relocations shown as "U" in nm must have a corresponding entry in the GOT table under '.rela.got'.

64-bit Library Path

- 64-bit libraries are in */lib64 instead of */lib.
- Makefiles that install libraries must be changed for this.
- Also the common -L/usr/X11R6/lib compile flag or a similar flag for the Qt library must be changed to reference */lib64.

Kernel Only Guidelines

- Interrupt flags used in save_flags() or spin_lock_irqsave() must be "unsigned long", not "int".
- How to check:
  - x86-64 include files force a warning for this with -Wall. Look for the warning and fix it.
  - Also the code may cause assembly failures with wrong types.
- Modules must be compiled with -mcmodel=kernel
  - Modules that are built external to the main source tree must be compiled with -mcmodel=kernel.
  - Otherwise they will crash soon after loading.
  - Modules built in the kernel tree do this automatically.
- Drivers must use the PCI DMA mapping API for bus addresses.
  - See Documentation/DMA-mapping.txt in the kernel source tree.

Questions ???

- See www.x86-64.org

Trademark Attribution