Overview of AIX 4.3

By Ahmed Chibib

This article discusses the characteristics of the new 64-bit architecture of the RS/6000 and the new 64-bit capabilities of AIX 4.3. It also presents issues to consider when porting an application from 32-bit to 64-bit.[1]

The need for 64-bit processing is driven by the increasing demands of today's applications. Although there is no such thing as a "typical" 64-bit application, most need to address large amounts of storage. For example, as database engines grow in capacity, they must manage larger and larger amounts of data in memory buffer pools to meet their performance criteria. In addition, they need more processing power to avoid becoming CPU-bound. Data warehousing applications are another example of the need for increasing storage capacity and high performance.

In the technical application arena, greater processing power enables larger problems to be solved, and larger problems typically mean larger arrays of grid points or other mathematical representations of the physical problem being analyzed. For many technical applications, storage requirements grow exponentially with the dimensions of the problem.

Overview of the RS/6000 64-bit Architecture

Applications running on AIX 4.3 can now be 32-bit or 64-bit. When application developers create an application for AIX, they must chose the appropriate architecture, based on several criteria, including the type of application and the performance requirements. A 32-bit application executes in an environment with 4 GB of addressability (virtual address space). A 64-bit application executes in an environment with much larger addressability (over 16 billion gigabytes).

The 32-bit applications can be run on any RISC System/6000 system, but 64-bit applications can be run only on a 64-bit RISC System/6000 with AIX 4.3. Both 32-bit and 64-bit applications (and libraries) can be compiled and linked on both 32-bit and 64-bit systems and AIX 4.3.

The RS/6000's 64-bit architecture has many benefits, but the choice to develop for a 64-bit environment should be based on the type of application and its requirements. The issues to consider include the performance requirements, the type of objects and archive files to be used, how the execution environments differ between 32-bit and 64-bit, and tools that are available for 64-bit architecture.

[1] This article is adapted from the *AIX 64-bit Migration Guide*, which may be viewed in its entirety from the following Web site address: http:/www.developer.ibm.com/sdp/library/aix4.3/.

Performance

The 64-bit address space can dramatically improve the performance of applications that manipulate large amounts of data—multiple gigabytes or larger. This data can be within the application or obtained from files. If a 64-bit application can contain the data in its address space—either created in data structures or mapped into memory—there will be a performance gain compared to a 32-bit application where the data would not fit into the address space.

There are two reasons for this performance improvement. First, the system call overhead of reading and writing files can be avoided by mapping the files into memory. Second, 64-bit systems can support physical memories that are larger than the addressability of 32-bit applications, so 64-bit applications are needed to make full use of the physical memory available.

Although the same source code can be used to create both the 32-bit and 64-bit application, the 64-bit application will be the same size or larger than the 32-bit application. In addition, 64-bit applications often run slower than 32-bit applications, unless they use their larger 64-bit addressability to improve performance. For this reason, developers generally create 32-bit applications, unless 64-bit addressability is required by the application or it can be used to dramatically improve performance. The default mode for AIX development tools is to create 32-bit objects and applications.

64-bit Objects and Archive File Types

The 64-bit libraries and applications can be created only from 64-bit objects. A *64-bit object* is an object type (64-bit XCOFF format) created by a compiler or assembler in 64-bit mode; that is, the compiler or assembler has been requested to create 64-bit objects rather than 32-bit objects. The 32-bit XCOFF format was the only object type in all releases of AIX before AIX 4.3.

Developers cannot create an object or application using both 32-bit and 64-bit object files. While the system-provided archives and libraries contain both 32-bit and 64-bit object files, the linker selects the appropriate objects from the library based on the type of linking that is requested (32-bit or 64-bit) and creates an object or application of that type.

AIX has two archive file types. The first does not recognize 64-bit object files and cannot be larger than 2 GB. Prior to AIX 4.3, this was the only archive file type. The second archive file type recognizes both 32-bit and 64-bit object files and will work with files larger than 2 GB.

32-bit Versus 64-bit Execution Environments

Some differences between the 32-bit and 64-bit execution environments (or modes) include the following:

- All pointer types in 64-bit mode are 64 bits in size
- The C “long” type (and types derived from it) in 64-bit mode is 64 bits in size
- 64-bit applications can use 64-bit PowerPC instructions
- The size of a machine register is 64 bits in 64-bit mode

Theoretically, the maximum limit for 64-bit applications, their heaps, stacks, shared libraries, and loaded objects is millions of gigabytes. However, the practical limits are dependent on the file system limits, paging space sizes, and system resources available.

All C fundamental types, except “long” and pointer types, will be the same size in 32-bit and 64-bit compilation modes.

Tools for 64-bit Development

AIX supports the standard tools for building, examining, and debugging 64-bit applications. The `yacc`, `lex`, and `lint` tools work with source code for both 32-bit and 64-bit compilation.

Both 32-bit and 64-bit objects can be created using the C compiler and the assembler. Both 32-bit and 64-bit objects and applications can be created with the linker.

The following tools work with both 32-bit and 64-bit objects and applications: `make`, `ar`, `strip`, `dump`, `nm`, `prof`, `gprof`, `lorder`, `ranlib`, `size`, `strings`, and `sum`.

The dbx and xldb tools can debug both 32-bit and 64-bit applications.

The 64-bit Architecture of the PowerPC

For a computer system to be considered to have a true 64-bit architecture, it must handle data 64 bits in length; that is, a contiguous block of 64 bits (8 bytes) in memory must be defined as one of the elementary units that the CPU can handle. The instruction set must include instructions for moving a 64-bit datum, and arithmetic instructions for performing arithmetic operations on 64-bit integers.

The 64-bit architectures must also generate 64-bit addresses, both as effective addresses (the addresses generated and used by machine instructions) and as physical addresses (those that address the memory cards plugged into the machine memory slots). Individual processor implementations may generate shorter physical addresses, but the architecture must support up to 64 bit addresses.

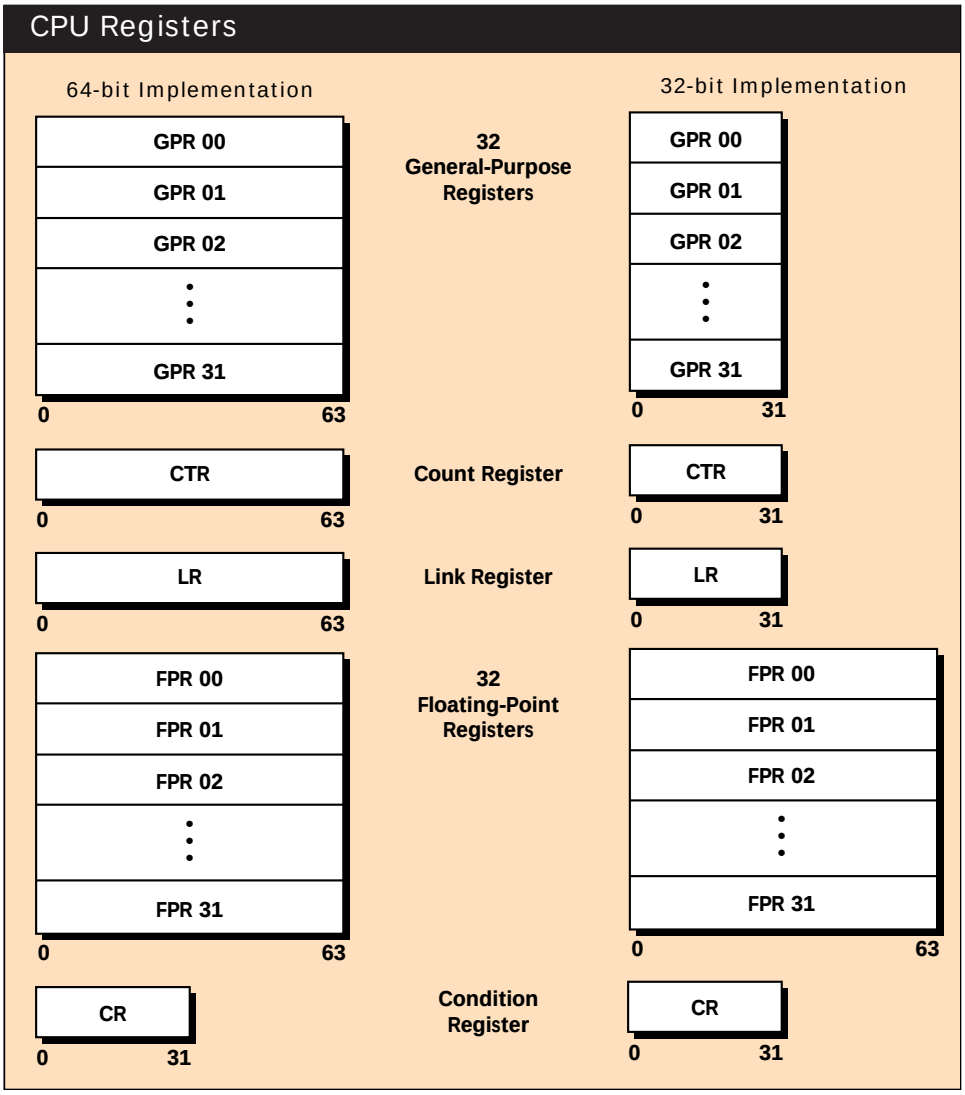

Figure 1. CPU registers for 32-bit and 64-bit architectures

PowerPC and Binary Compatibility

The IBM PowerPC has some important advantages compared to other processor architectures. From the beginning, it was designed as a 64-bit architecture, with 32-bit mode as a functional subset. Its 64-bit capability is *not* an adaptation of an existing 32-bit architecture. This design makes binary compatibility easier to maintain.

Binary compatibility with the current PowerPC processors is an important aspect of the 64-bit version of PowerPC. The 32-bit and 64-bit specifications have several differences. As shown in Figure 1, the number of General-Purpose Registers (GPRs) remains the same, but these registers are 64 bits long, not 32 bits long. A few other control registers move from 32 bits to 64 bits in length. Floating-point registers do not change in size, because they conform to industry standards for floating-point, which require 32-bit or 64-bit length data.

Why Choose 64-bit Architecture?

A 64-bit architecture has significant advantages. Very long integers can be used in computations. It can manage very large file systems. And it has the capability of addressing huge address spaces, both in virtual and physical memory.

Figure 2 shows the size of the address spaces that can be managed, as a function

of the length of the address generated by the CPU.

Complex applications such as large databases, large numeric applications, and multimedia environments are a natural for 64-bit applications because of their need to manage and operate on larger and larger data sets. Some advantages that are important for these applications include the following:

- Supports very large programs. As shown in Figure 3, a 64-bit processor can support a much wider address space than its 32-bit equivalent, which makes it possible to develop much larger and more complex applications. Today's 32-bit applications are limited to 4 GB.
- Can manage large volumes of data. The 64-bit architecture can manage very large amounts of data, possibly in real memory. This simplifies the task of the applications that typically handle large data sets, for example, those in multimedia, statistics, or database.
- Can handle larger files. A 64-bit environment can easily manage huge files, file systems, and databases without forcing the application to explicitly handle the situations in which data sets larger than 2 GB are needed. Today's most sophisticated applications already support this capability. AIX Version 4 itself supports files larger than 2 GB.
- Supports larger physical memory. The physical addresses generated by a 64-bit CPU are up to 64 bits in length. This eliminates the 4 GB limit in real memory of all 32-bit architectures.

64-bit Enabled AIX 4.3

AIX 4.3 is a universal upgrade for all RS/6000 systems currently running AIX Version 4.x, as well as future 32-bit and 64-bit systems.

Address Length	Flat Address Space
8 bit	256 bytes
16 bit	64 kilobytes
32 bit	4 gigabytes
52 bit	4,000 terabytes
64 bit	16,384,000 terabytes

Figure 2. Address space sizes

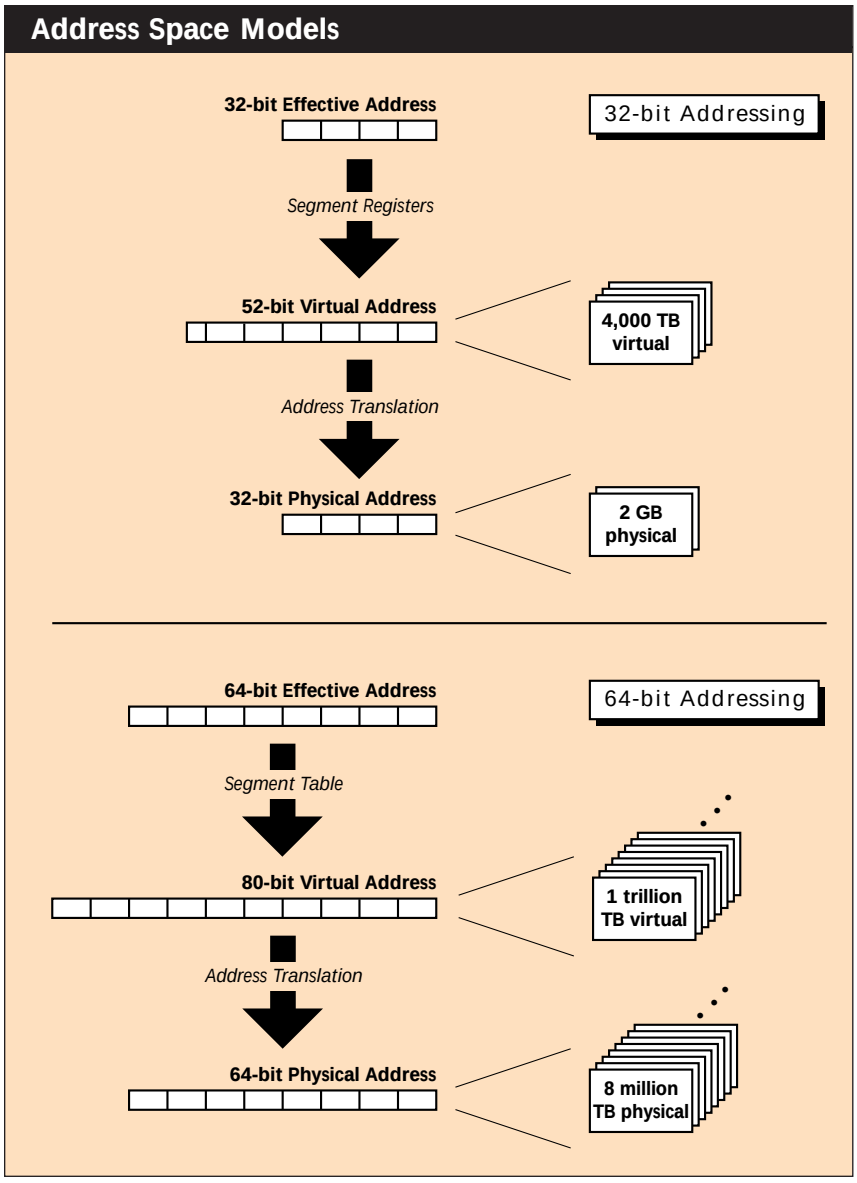

Figure 3. Address space models

End users should find 64-bit applications and 64-bit application support on AIX 4.3 completely transparent. They can run both 32-bit and 64-bit applications simultaneously. When they run 32-bit processes on 64-bit platforms, the execution is transparent and identical to executing on a 32-bit platform.

Platforms and Applications

Figure 4 summarizes the platforms on which 32-bit and 64-bit applications can run.

System Limits

As a result of the new 64-bit capabilities of AIX, system limits have been expanded as shown in Figure 5.

	32-bit Applications	64-bit Applications
32-bit Platforms (existing RS/6000s)	**32-bit applications run as they do on previous releases of AIX without modifications.** **Binary compatibility is maintained.**	**64-bit applications will fail to execute when run on any 32-bit system.**
64-bit RS/6000 Platforms	**Execution of 32-bit applications on 64-bit platforms should be transparent and identical to executing on a 32-bit platform.** **Binary compatibility is maintained.**	**64-bit applications should execute in a manner similar to an analogous 32-bit application.**

NOTE: Applications developed on AIX 4.3 are not backward compatible with previous releases of AIX.

Figure 4. Application platforms for 32-bit and 64-bit architectures

	Prior to AIX 4.3	AIX 4.3
File descriptors per process (both 32-bit and 64-bit processes)	**2,000**	**30,000**
System open file table	**200,000**	**1,000,000**
Number of shared memory IDs	**4,096**	**4,096**
Number of kernel threads per process (both 32-bit and 64-bit processes)	**512**	**32,768**
Number of processes	**131,072**	**131,072**
Number of threads per system	**262,143**	**262,143**

Figure 5. System limits

Figure 6 shows limits for 64-bit executables.

The File System

AIX 4.3 supports file sizes in excess of 2 GB. The 64-bit processes can open files of any size (in excess of 128 GB) without specifically indicating that they "understand" large files.

In the AIX 4.2 large file implementation, an application that is dealing with large files must have its file-size limit set to infinity. In the AIX 4.3 64-bit implementation, there are true 64-bit limits. Processes operating in 64-bit mode and those that are large-file enabled use the 64-bit limits. The 32-bit processes that are not large-file enabled will still see 32-bit limits, with limits exceeding 2 GB treated as infinity.

Shells

The `ksh`, `bsh`, and `csh` shells each have a built-in `ulimit` command. Prior to AIX 4.3, the maximum value of any `ulimit` that could be specified was a 32-bit value. In AIX 4.3, `ulimit` values as large as 64-bits can be specified in each shell. This is the only functional change made to each of the AIX 4.3 shells.

New Features of AIX 4.3

The new functionality in AIX 4.3 is not only available to 64-bit applications, but 32-bit applications can use it as well. The only exception is functionality that results directly from the larger address space.

The following sections describe the key new features available in AIX 4.3.

16 GB of real memory. This feature allows up to 16 GB of physical memory to be supported by AIX. This support is intended for Enterprise Server memory configurations. RS/6000 hardware systems with more than 4 GB of memory will become available in the future.

AIXwindows Version 11 Release 6. This includes all supported core X client programs, libraries, font server, fonts, and all client X internationalization support. This also updates the Motif libraries to Motif 2.1.0. X11R5 X extension load modules are not supported in the X11R6 X server. X11R5 `loadddx` load modules are not supported in the X11R6 X server.

Parameter	Limit
Executables	**$7*2^{56}$ bytes**
Sum of sizes of text data and BSS sections	**$7*2^{56}$ bytes**
Symbol values (generally address of objects)	**2^{64}**

Figure 6. Limits of 64-bit executables

ONC+. The ONC+ technology has been licensed from SunSoft and incorporated into AIX. ONC+ introduces CacheFS functionality:

- AIX headers C++ aware. All shipped header files in the Base System are C++ aware to support the xlC product.
- M:N Pthreads. The current 1:1 threads implementation has been replaced with an M:N version. This conforms with IEEE® POSIX® 1003.1c (threads). It provides increased performance for several types of database applications, as well as greater per-process limits for applications with large numbers of threads, such as databases.

  Existing AIX `Pthread` applications conform to POSIX 1003.1c Draft 7. AIX 4.3 supports both Draft 7 and Draft 10 in 32-bit mode. However, the 64-bit mode supports only Draft 10; therefore, if you are porting existing `Pthread` applications to 64-bit, you must convert to Draft 10.

Thread-Safe Libraries. `Non-thread-safe` and `thread-safe _r` libraries have been combined into one set of libraries, making thread-safety a default. For example:

AIX 4.2 includes: `libc.a (non-thread-safe)` and `libc_r.a (thread-safe)`
AIX 4.3 includes: `libc.a (thread-safe)`

Libraries, such as X11R6, which link with `libc.a` are not thread-safe by default;

they are thread aware. New libraries that are thread-safe include the following: `libbsd.a`, `libm.a`, `libmsaa.a`, `libs.a`, `libdes.a`, `libXext.a`, and `libXi.a`.

These thread-safe libraries provide a convenient programming model to exploit Symmetric Multiprocessors (SMPs) and to simplify the exploitation of threads by applications, middleware, and other Application Programming Interface (API) providers.

WebSM. This is an easy-to-use interface to AIX system management functions that enables administrators to remotely access them over the Internet. WebSM is the new primary interface for system management.

WebSM has the following characteristics:

- Accessible remotely via the Internet or intranets
- Accessible from any client platform that has Java 1.1 support or a Java 1.1-enabled Web browser
- Consistent with the Common Desktop Environment (CDE) desktop, and Microsoft Windows 95, Windows NT™ applications in user interface style
- Less administrative skill required than traditional UNIX system management applications
- Organization of system management administration in the form of "real world" objects in the user interface; for example, users, groups, devices, software packages/applications
- Full coverage of AIX system management functionality; enables administrators to perform all key AIX system management from the WebSM

Enhanced Online Documentation. In AIX 4.3 the online publications are HTML based. InfoExplorer has been replaced by any Web browser that can view, search, and hypertext link an HTML document .

NFS Version 3. NFS Version 3 has been extended to support HANFS hooks and daemons, removing the restriction of HANFS configurations with only two nodes. This allows a node to serve as backup for more than one other node.

IP Version 6. Internet Protocol Version 6 is the next generation of the Internet Protocol, which relieves IP address limitations and adds robust fail-safe routing, automatic configuration of IP addresses, as well as improved security and integrity.

The IP Version 6 implementation in AIX 4.3 protects customer investment in existing applications while introducing the next generation of IP. Customer applications that are written with the current IP (Internet Protocol Version 4) applications interface will work as-is (binary compatible).

Most AIX-provided Internet applications, such as name resolver, configuration, mail, `r-cmds` (`rlogin`, `rsh`), telnet, ftp, and tftp are upgraded to support IP Version 6. AIX 4.3 IP Version 6 provides the required infrastructure for serving all the Internet-ready boxes and also the devices to participate in the growing Internet. AIX IP Version 6 supports host requirements, but not IP forwarding.

IP Security on IP Versions 4 and 6. IPSec, a security protocol in the IP layer, provides security services to ensure packet authentication, integrity, access control, and confidentiality. A secure tunnel is established between the two systems to perform key exchange, message encryption, and message authentication.

AIX 4.3 supports new IP security protocols, Authentication Header (AH) for integrity and authentication, and Encapsulating Security Payload (ESP) for confidentiality. The AIX implementation of IPsec interoperates with other IPsec-enabled products, such as IBM FireWall.

A new key management mechanism is added to AIX 4.3 since keys must be generated and distributed for IPsecurity. A shared master key is manually distributed between systems at the tunnel endpoints where one system generates and exports a master key file and the other system uses this key file to import the master key definition. Session key protocols, both static for tunnel duration, and dynamic are also provided.

Address Space Layout

Segment Number	Use in 32-bit User Mode
0	kernel text
1	user text
2	process private data
3	available for user
4	shmat, mmap
5	shmat, mmap
6	shmat, mmap
7	shmat, mmap
8	shmat, mmap
9	shmat, mmap
0xA	shmat, mmap
0xB	shmat, mmap
0xC	shmat, mmap
0xD	shared lib text
0xE	shmat, mmap
0xF	shared lib data

Segment Number	Use in 64-bit User Mode
0	kernel
1	kernel
2	process private
3–0xC	shmat/mmap
0xD	loader use
0xE	shmat/mmap
0xF	loader use
0x10–0x6FFFFFFF	application text, data, BSS, heap
0x70000000–0x7FFFFFFF	default shmat/mmap
0x80000000–0x8FFFFFFF	private load
0x90000000–0x9FFFFFFF	shared library text and data
0xA0000000–0xEFFFFFFF	reserved for system use
0xF0000000–0xFFFFFFFF	application stack

Figure 7. Address space layout for 32-bit and 64-bit architectures

BEST-X. The BEST-X security product installs on top of AIX 4, and provides the features necessary to meet a European E3/F-B1 certification.

LDAP. LDAP (Lightweight Directory Access Protocol) is a directory service protocol that runs over TCP/IP. It can be used with stand-alone and other kinds of directory servers.

Address Spaces

Figure 7 shows the address space layout for both 32-bit and 64-bit applications. The new 64-bit user address space is substantially larger than the one for 32-bit applications. A 32-bit application will always see the 32-bit address space, regardless of whether it is running on 32-bit or 64-bit hardware, however a 64-bit application will see the entire 64-bit address space.

See *The AIX 64-bit Migration Guide* (http://www.developer.ibm.com/sdp/library/aix4.3/) for a detailed discussion of address space programming interfaces and the kernel's remapping routines.

AIX 4.3 provides a 60-bit effective address space (up to 2^{32} segments). Processes can access address spaces via `shmat()` or `mmap()`. These `shmat` and `mmap` segments are allocated starting at segment 0x70000000 if no address is specified. Segments are allocated in increasing order in the first available segment at or after segment number 0x70000000. The `shmat` and `mmap` segments are placed where requested if the segment number is available and less than segment number 0x80000000.

Explicitly loaded modules—using the `load()` system call—are loaded into separate segments starting at segment number

0x80000000. Segment numbers are allocated in increasing order in the first available segment number at or after segment number 0x80000000. Explicitly loaded modules are limited to segment numbers 0x80000000 to 0x8FFFFFFF.

Application Development Toolkit (ADT)

All development tools that process program source code can target the 64-bit execution environment while running on 32-bit RS/6000s. However, testing 64-bit executables requires 64-bit hardware because there is no simulation environment of the 64-bit ABI on 32-bit hardware. It is easy to port programs between 32-bit and 64-bit environments and have equivalent functionality. It is the data size issues that become the primary porting concern.

Exploiting 64-bit address space requires more substantive programming changes in most C applications. Compiler flags can be used as porting tools that warn about all programming constructs that may cause errors when code is compiled. Examples of such constructs include longs assigned or cast to `int`s, pointers assigned or cast to `int`s, and parameter mismatches.

The C for AIX compiler. The 64-bit implementation of the C front end has not changed the behavior of the compiler. The default compilation and assembly mode is 32-bit. The compiler will only change the behavior of code when compiling for 64-bit mode. The developer can change the default from 32-bit to 64-bit mode.

The 64-bit compiler can be invoked by passing a separate flag, `-q64` (`-qarch=ppc64`). The 64-bit binder is evoked by passing a separate flag (`-b64`).

Compiler Invocation. The new features of the compiler that enable 64-bit mode include the following:

- Predefined `__64BIT__macro` when invoked for 64-bit compilations
- `OBJECT_MODE` environment variable
- `-qarch` support for 64-bit suboption

__64BIT__ macro. When the compiler is invoked to compile for 64-bit mode, it defines the preprocessor `macro __64BIT__`. When it is invoked in 32-bit (default) mode, this macro is not defined. This variable can be tested via `#if defined(__64BIT__)` or `#ifdef __64BIT__` to select lines of code, such as `printf` statements, that are appropriate for 64- or 32-bit mode. This ability to choose execution mode of the final executable at compile time and the existence of the `__64BIT__` macro implies there is no need to test execution mode at runtime.

OBJECT_MODE Environment Variable. The C compiler obtains information from environment variables. These may be the `setlocale` variable which changes the language of messages, or the `_xlC_test_usrincpath` variable, which overrides the default place to look for system header files (`/usr/include`). This latter option makes it easy to test new header files without changing them in the default location. Neither of these environment variables have a competing command-line option that can contradict it.

A new environment variable called `OBJECT_MODE` replaces the default compilation mode of the compiler when it is used. If `OBJECT_MODE` does not exist, the compiler defaults to 32-bit mode, facilitating compatibility with systems such as AIX 4.3 installed on 32-bit machines, which do not support 64-bit.

Command-Line Options. Command-line options override the environment variable `OBJECT_MODE` when setting the compilation mode. When compiling for 64-bit mode, the compiler generates 64-bit instructions when dealing with 8-byte data, and produces files in the 64-bit XCOFF format. The compiler cannot generate 64-bit instructions that operate safely on 32-bit XCOFF, which implies that 64-bit and 32-bit objects cannot be mixed. An architecture (`ARCH`) compile-time flag (option) determines 64-bit instruction and XCOFF. This `ARCH` suboption completely and uniquely determines the instruction set and XCOFF mode.

Mixed Mode Compilation. When 32-bit and 64-bit compilation modes are mixed, XCOFF objects will not bind. This becomes obvious if the compile and link

occurred in one step, however, if the compile and link occurred in different steps, the developer may not be aware of this.

If an application was compiled and produced 64-bit objects, remember to link using the 64-bit mode (when linking using xlc), otherwise the objects will not link. Objects in mixed XCOFF mode will never link and must be recompiled completely, ensuring that all objects are compiled in the same mode.

Compiling and Linking for 64-bit Execution. The default compilation and assembly mode is 32-bit. The developer can change from the default 32-bit to 64-bit mode. In the compiler, an option such as `-q64` and/or `-qarch=ppc64` can be used to change the compilation mode.

Lint. `lint` has been modified to flag some common 32-bit and 64-bit porting problems. Using the `-t` flag enables these changes. For example, `lint -t` will find the following common potential problems:

- Functions not prototyped. Function prototypes allow the compiler and `lint` to flag mismatched parameters.
- Assignment of a long or a pointer to an `int`, which could cause truncation. Even assignments with an explicit cast will be flagged.
- Assignment of an `int` to a pointer. If the pointer is referenced, it may be invalid.
- Shift operations involving a long or pointer. Since the pointer size was increased to 64-bits, the result may no longer be valid.
- Masks using bitwise OR, AND or XOR involving longs or pointers. Again, since the pointer size has changed, the mask may no longer be sufficient.

The following are a few problems `lint -t` cannot find:

- Shared data must be the same size in both 32-bit and 64-bit. For example, the `utmp` structure in `/usr/include/utmp.h` is used to read and write data from `/etc/utmp` and its size must be consistent in both 32-bit and 64-bit modes.
- Unions that use longs or pointers may no longer work. The following two union examples work fine in a 32-bit environment but will not work in a 64-bit environment. Figure 8 shows two examples.

Example 1

```
union {
int *p; /* 32 bits / 64 bits */
int i;  /* 32 bits / 32 bits */
};
```

Example 2

```
union {
double d;  /* 64 bits / 64 bits */
long l[2]; /* 64 bits / 128 bits */
};
```

Figure 8. 32-bit union examples

Two other differences can affect programs. In 32-bit mode registers are only 32-bits wide, so long long's are passed in two general-purpose registers. In 64-bit mode, long long's are passed in only one (64-bit wide) general-purpose register.

In 32-bit mode when a function is passed floating-point arguments and it is not prototyped, each floating-point argument is placed in one floating-point register and in two general-purpose registers. In 64-bit mode, each floating-point argument is still placed in one floating-point register, but only one general-purpose register.

The Assembler. The assembler will continue to execute in 32-bit mode on 64-bit systems, but will also produce object files in the 64-bit XCOFF format for 64-bit execution.

Assembler Execution Mode. The architecture on which an assembly is intended to execute can be specified in two ways:

- Via the `-m` (for "machine") command flag, which sets the default target architecture and mnemonic set for the assembly. Figure 9 shows values for `-m` accepted by the assembler

- Via a `.machine` pseudo-op specifying one of the codes that is valid with `-m`. The `.machine` pseudo-op overrides any `-m` command-line flag. More than one `.machine` pseudo-op can appear in an assembly; that is, different parts of the assembly can be assembled for different target machines with different mnemonic sets

-m	Architecture and Implementation
ppc	**32-bit PowerPC; this is the default if `-m` is not specified**
pwr	**POWER architecture, POWER implementation**
ppc64	**64-bit PowerPC**
com	**POWER and PowerPC**
pwrx, pwr2	**POWER architecture, POWER2 implementation**
any	**POWER, or POWERPC**
601	**32-bit PowerPC, 601 implementation**
603	**32-bit PowerPC, 603 implementation**
604	**32-bit PowerPC, 604 implementation**

Figure 9. Assembler values

The Assembler Lexical Scanner. The assembler's lexical scanner now accepts integer values in a variety of bases that require up to 64 bits to represent. This allows for longer numeric fields; for example, as many as 16 hexadecimal digits can be required to specify a 64-bit mask field.

To avoid the problem of reading, writing, and checking (verifying) such long string of digits, the lexical scanner will now accept and ignore underscore characters ("_") within numeric fields. This allows numbers to be separated into groups, such as grouping by thousands (groups of 3) in decimal numbers.

Thus, instead of coding:

```
.long 4294967295
```

the user can now code:

```
.long 4_294_967_295
```

The Linker. The AIX linker supports the development of 64-bit applications, libraries, and kernel extensions. For 64-bit applications and libraries, the linker reads and writes XCOFF64 files while performing internal processing appropriate for 64-bit mode.

For AIX 4.3, the base kernel and kernel extensions are XCOFF32 files. Thus, a 64-bit kernel extension comprises XCOFF32 object files linked in 32-bit mode, using an export file that marks some symbols as 64-bit kernel services (Device drivers that support 64-bit processes do not need to be compiled or linked in any special way. They indicate their ability to handle 64-bit processes when they are configured.)

Linking in 32-bit mode works the same way on both AIX 4.3 and AIX 4.2.

Two new flags have been added, `-b32` and `-b64`, which will specify linking modes of 32-bit and 64-bit, respectively. These options may come from a stanza file, such as `/etc/xlC.cfg`, or from the command line. The new environment variable, `OBJECT_MODE`, can also be used to specify a mode.

- `-b32`: Specifies 32-bit linking mode. In this mode, all input object files must be XCOFF32 files, or an error is reported.
- `-b64`: Specifies 64-bit linking mode. In this mode, all input object files must be XCOFF64 files, or an error is reported.
- `OBJECT_MODE` environment variable. If neither the `-b32` nor `-b64` option is used, the `OBJECT_MODE` environment variable is examined to determine the linking mode. If the value of `OBJECT_MODE` is "64", 64-bit mode is used. If the value is "32_64", the

linker prints an error message and exits with a non-zero return code. Otherwise, 32-bit mode is used.

The Loader. The loader for 64-bit programs provides equivalent semantics to the 32-bit loader.

Other Application Development Utilities. For the following XCOFF-specific commands, `dump`, `nm`, `lorder`, `ranlib`, `size`, `strip`, a new flag, `X`, has been added. It requires an argument of either 32, 64, or 32_64. This flag indicates whether to recognize only 32-bit objects or only 64-bit objects (in addition to any non-object files, which are always valid), or both.

New Functions. These commands have been enhanced to provide the same functionality available for both 32-bit and 64-bit objects and executables. They will also understand both new and old archive file formats.

The `prof` and `gprof` commands have been expanded to support profiling of 64-bit executable programs. The same functionality is available for both 32-bit and 64-bit executables.

The `lint` command has a new flag to allow the user to specify whether the source code is intended for a 32-bit (default) or a 64-bit compilation, or a port from 32-bit to 64-bit.

C source code generated by the `lex` and `yacc` utilities will be able to compile correctly in either 32-bit or 64-bit compilations. The `string` and `sum` commands perform the same functions for both 32-bit and 64-bit XCOFF files.

All BOS commands recognize and support data values that might exceed 231 in a 64-bit environment.

Archive File Format. The AIX 4.3 `ar` command handles the archiving of 64-bit XCOFF object modules in addition to 32-bit XCOFF object modules. To support the two different formats of object files, the symbols of 64-bit objects are distinguishable from those of 32-bit objects.

There are two global symbol tables: one for 32-bit object symbols and one for 64-bit object symbols. The `ar` command recognizes each type of object file and stores its symbols in the appropriate table.

The `ar` command maintains compatibility with the previous archive file format by adding, deleting, reordering, and listing members without altering the format of an old archive file except in two cases:

- When the user explicitly requests conversion to the AIX 4.3 format by using the `-o` option
- When the user adds a 64-bit object to the archive (This requires conversion because a 64-bit object cannot be handled by the old-format archive.)

The two basic changes of the archive format are separate global symbol tables for 32-bit and 64-bit object symbols and the expanded size limits of a file format. The `ar` command handles both this new archive format (for 32- and 64-bit) while continuing to recognize the old format (32-bit only) for compatibility.

PTX. All PTX commands remain functionally compatible with their previous behavior. The commands on a 32-bit system look and act the same as they would have prior to AIX 4.3. The commands themselves all remain 32-bit executables, enabled to recognize 64-bit values where appropriate.

Ahmed Chibib, IBM Corporation, 11400 Burnet Road, Austin, TX 78758. Mr. Chibib is a senior technical staff member responsible for providing technical assistance to software developers. He worked at IBM Research on several projects including the initial RISC architecture. He has also worked on the RT architecture, and the C PL8 optimizing compilers for the RT and the RS/6000. Prior to his current position, Mr. Chibib was a member of the AIX Architecture team and the software representative on the PowerPC Architecture Working Group.