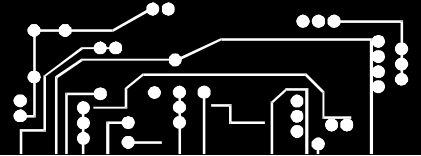


PowerPC Embedded Processors Application Note



Programming Model Differences of the IBM PowerPC 400 Family and 600/700 Family Processors.

Microcontroller Applications
IBM Microelectronics
Research Triangle Park, NC
ppcsupp@us.ibm.com
Version: 1.0

September 30, 1999

Abstract – This application note describes how the programming models differ between the IBM 400 family and the 600/700 family of PowerPC architecture processor implementations. It is useful to system designers and programmers porting code from one family to another.

The 600/700 family is compliant with the original PowerPC processor architecture designed for desktop applications. The 400 family, designed for use in embedded environments, maintains compatibility in the base User Instruction Set architecture. However, it defines separate Virtual Environment and Operating Environment architectures that have been enhanced for embedded applications. These optimizations have prompted changes in features such as memory management, debug support, and cache access and control, and processor extendibility.

1 PowerPC Architecture Overview

The original PowerPC Architecture defines a family of processors that span a range of price and performance. The architecture specification is defined in four books that define four levels of the architecture. IBM currently offers two families of PowerPC implementations, the 600/700 family which is targeted towards general purpose computer applications, and the 400 family which has specific features which optimize its use in embedded control applications.

Book 1 defines the *User Instruction Set Architecture*, to which all Power PC processors conform. Thus, application level programs that only use Book 1 features are portable among implementations without modification. Book 1 also defines floating-point support as well as the option of implementing 64 or 32-bit sized general-purpose registers. Floating-point operations may be implemented with hardware assistance or, to maintain binary compatibility, as software routines where floating-point instructions cause illegal instruction exceptions.

Book 2 defines the *Virtual Environment Architecture*, which defines features that permit application programs to share data among programs in a multiprocessing system; and optimize the performance of storage accesses. This book has been modified by IBM for embedded application oriented processors of the 400 family and is entitled *IBM PowerPC Embedded Virtual Environment*. The 400 family processors support all storage control instructions and the Time Base functionality defined in the original *Virtual Environment Architecture*. Future implementations can have support for multi-processor memory coherence.



Book 3 specifies the *Operating Environment Architecture*, which defines features to permit operating systems to allocate and manage storage to handle exceptions, and to support I/O devices in support of multiprocessor operating systems. Again, changes have been made to enhance embedded usage capability. The IBM PowerPC Embedded Operating Environment is similar to the base architecture but many differences exist. Changes include the addition of a DCR address space, a dual-level interrupt structure with critical and non-critical interrupt sources, additional cache management instructions, extended timer facilities, an MMU better suited for embedded usage, and enhanced debug capabilities. Because these features are available only to "privileged" programs, application code is still compatible regardless of Book 3 variations.

Book 4, the *PowerPC Implementation Features* defines all implementation specific aspects of the architecture. Any code for this book will likely need to be revised or removed when porting across PowerPC families.

PowerPC Book E combines the architecture specification of the original four architecture books, plus changes and enhancements, to define a standard evolutionary path for future PowerPC implementations that will provide greater code portability.

Note: Through out this document, a reference to the 600 family of PowerPC implementations generally also includes the 700 implementations. When a specific distinction between the 600 and 700 families is necessary, it will be clearly indicated.

2 Memory Management Unit

The primary function of the Memory Management Unit (MMU) in a PowerPC processor is the translation of effective (logical, virtual) addresses to physical (real) addresses for instruction and data storage accesses. The secondary function of the MMU is to provide access protection and storage attribute control on a memory sub-region basis. These functions support demand-paged virtual memory. There are many aspects of memory management that are implementation-dependent. The 400 series of processors have MMUs that are very similar to each other, and which are distinct from the 6xx/7xx MMU design. Operating system code used for processor initialization, exception handling, device drivers, and physical memory management will have significant differences. Application code that uses only the Book 1-architecture features will not be impacted.

2.1 600/700 Family MMU Overview

The 600/700 family MMU manages memory by dividing the address range of memory into blocks, segments and pages. Blocks are used for defining large regions of memory to have the same access protection and storage attributes. Memory blocks are created by using instruction or data Block Address Translation (BAT) registers to define memory blocks sized between 128KB and 256MB.

If a memory block is not defined for a given effective address, address translation is handled by the page address translation logic, which supports virtual paging. Virtual paging is done using a software constructed page table and hardware managed TLB.

The 4K-byte pages are a fixed in size and define the granularity of memory management when blocks are not used. Effective page addresses are translated to interim virtual addresses using the segment registers, and a page table stored in RAM translates the virtual address to a physical address. Segments are 256MB effective address memory regions that are mapped to virtual segments by using the Segment

Registers (SR0-SR15). The virtual segment address defined in the SR is used to access a Page Table Entry (PTE). SR registers are modified using the **mtsr**, **mfsr**, **mtsrin** and **mfsrin** instructions.

The PTE defines the physical address to be used. The MMU automatically scans the page table for a matching entry each time an execution unit presents an address. When a match is found, the entry is cached in the MMU managed TLB. The 603-processor software is assisted by table search functions, which are controlled by using the table search registers. The **tlbie**, **tlbia** and **tlbsync** instructions are provided for software TLB management. Table 1 – 600/700 Family MMU Features shows a summary of the 600/700 family MMU features.

Table 1 – 600/700 Family MMU Features

Feature Category	603 specific	Feature
Address ranges	No	<ul style="list-style-type: none"> • 2³² bytes of effective address • 2⁵² bytes of virtual address • 2³² bytes of physical address
Page size	No	4 Kbytes (fixed)
Segment size	No	256 Mbytes
Block address translation	No	<ul style="list-style-type: none"> • Range of 128 Kbytes to 256 Mbytes in size by powers of 2 • Implemented with IBAT and DBAT registers
Memory protection	No	<ul style="list-style-type: none"> • Segments selectable as no-execute • Pages selectable as user/supervisor and read-only • Blocks selectable as user/supervisor and read-only
Page history	No	Referenced and changed bits defined and maintained by hardware
Page address translation	No	<ul style="list-style-type: none"> • Translations stored as PTEs in hashed page tables in memory • Page table size determined by mask in SDR1 register
TLBs	No Yes	Instructions for maintaining optional TLBs (tlbe instruction in 603) 64-entry, two-way set associative ITLB and DTLB
Segment descriptors	No	Stored in segment registers on-chip
Page Table search support (Note: table search support varies, the 603 and 603e use software table searching as described. The 604, 740 and 750 have hardware assisted table searches.)	Yes	<ul style="list-style-type: none"> • Three MMU exceptions defined: ITLB miss exception, DTLB miss on load exception, and DTLB miss on store (or C = 0) exception • IMISS and DMISS registers (missed effective address) • HASH1 and HASH2 registers (PTEG address) • ICMP and DCMP register (for comparing PTES) • RPA register (for loading TLBS) • tlbli rB instruction for loading ITLB entries • tlbld rB instruction for loading DTLB entries • Shadow registers for GPR0-3 (can use r0-r3 in table search handler without corruption of r0-r3 in context that was previously executing)

2.2 400 Family MMU Overview

The 400 family MMU divides the address range of memory into pages when address translation is enabled, and two bounded-regions when address translation is disabled. It has a unified 64-entry fully associative TLB array. Page entries for either instruction or data accesses can be placed anywhere in the TLB. Pages are individually sizeable from 1K byte to 16M bytes. Since each can be sized as needed, fewer TLB entries are needed and the amount of TLB entry swapping is eliminated, or greatly reduced

compared to a 600 family implementation. TLB effective address matching is optionally additionally qualified by the TID field having to match the contents of the PID (Process ID) register.

When address translation is enabled (i.e., virtual-mode), the MMU divides the effective address range into pages. Pages are individually controllable regarding address translation and access protection. Eight page sizes are available between 1KB and 16MB. Page size and address translation is controlled by the entries in the TLB array. The TLB is completely managed by software, creating and deleting TLB entries, assisted by three table search instructions; **tlbre**, **tlbwe**, **tlbsx**.

If a page table is used, there is no hardware assistance for lookup of page table entries or management of the Referenced and Changed bits. Virtual paging is done using a software constructed page table and software creation of entries in the TLB. Software can quickly check for the existence of, or find existing entries by using the table search instruction, **tlbsx**. Effective addresses without a matching TLB entry are automatically recorded in the SRR0 register for instructions or in the DEAR for data loads and stores, before the TLB miss exception handler is executed.

Effective page addresses are translated to interim virtual addresses by combining the 32-bit effective address with the 8-bit PID register value to form a 40-bit value which the fully associative array hardware of the TLB will attempt to match with the tag portion of a TLB entry. If a match is found in a TLB entry, the data portion of the TLB entry provides the physical address to be used.

The access protection of multiple pages can be controlled using the Zone Protection Register (ZPR). Each TLB entry can be assigned to one of 16 zones, which then control the protection for all such TLB entries.

For the 403 family implementation, with address translation disabled (real-mode) read-only accesses can be defined for two regions of memory by using the PBL1-PBU1 and PBL2-PBU2 register pairs. The region can be either inclusive or exclusive of the address range defined by the register pairs. Protection against writes is enabled using the MSR[PE] (Protection Enable) bit. The PE bit is automatically negated upon the occurrence of any interrupt, allowing the interrupt handler write-access to all of memory. Control of speculative fetching is controlled using the SGR (Storage Guarded) register. This read-only access feature has been removed from the architecture and is not implemented on the 401 or 405 processors.

2.3 MMU Differences

The 400 family does not have the segmented memory concept, and therefore, no segment registers or tables as defined in the PowerPC architecture and as provide by the 600 family processors. It also doesn't have hardware assistance for TLB management by looking up page table entries in a page table. Instead, it uses a software managed TLB, therefore virtual-paging software for the 400 family will have to provide TLB entry management as well as the page table management functions. 600 family TLB management code being ported has to be modified to work with the 400 family processor's unified TLB, and to use the **tlbsx** instruction.

When porting code from the 400 family, realize that the 600 family does not have the concept of zones, so page table entry creation software must be modified to eliminate any functions related to the TLB entries ZSEL field and the ZPR register.

Guarding against speculative memory accesses is controlled by the SGR in the 400 family versus using the WIMG bits of either the BAT register or the page table entry in the 600 family processors.

3 Cache Differences

The PowerPC architecture does not impose a specific L1 cache organization. To ensure portability among PowerPC implementations, programmers should assume that all implementations possess separate instruction and data caches. All IBM 600 and 400 family processors have separate (non-unified) instruction and data caches.

For the 401 and 403 implementations, the **iccci** instruction invalidates individual cache blocks. For the 405, **iccci** invalidates the entire instruction cache. Therefore, the portion of cache management code initializing the cache prior to first usage can run unchanged when ported from a 401/403 to a 405. Although the additional **iccci** instructions would be redundant, they are harmless. The 600 family processors invalidate the entire instruction cache by asserting the HID0[ICFI] bit. When porting code from 600 family to 400 family, use of the **iccci** instruction is required as well as removal of references to HID0, which doesn't exist.

Another cache management instruction, **icbt** (instruction cache block touch), varies in execution privilege among the 400 family and does not exist at all in the 600 family. For 401 and 403 processors it is a privileged instruction that may only be executed when in supervisor mode. For the 405, user-mode software may also use it thereby making it available to applications for performance enhancement.

For the 401 and 440 processors, cache blocks are individually lockable, use of this feature will have to be discontinued if porting to other members of the 400 or 600 families. The 600 family can lock the entire instruction or data cache.

The 400 family provides a capability to read the cache. For the 401 and 403, the CDBCR register is used along with the **icread**, **icbldr** and **dcread** instructions. The 405 uses the CCR0 register. The 600 family does not have any capability to read caches via code, however for debugging purposes, it can be read through the JTAG port.

4 Exception Differences

This section describes the differences in exception processing between the 400 and 600 family of PowerPC implementations. In some cases, the only difference is that the exception vector address varies for the same type of exception causing event. In others, exception vector address locations are used for different purposes, or the causing event may vary slightly with different detailed causal information. Finally, some exceptions are completely unique to a specific family and/or implementation.

The 400 family locates the interrupt vectors using the EVPR register that provides the high-order 16 bits of the exception handler routine addresses. The 600 family has only two possible vector locations, either 0x000n_nnnn or 0xFFFFn_nnnn, selected by using the MSR[IP] bit. The restricted location options of the 600 family affects the system memory map, typically requiring location of RAM starting at address 0x0000_0000 and ROM covering the addresses starting at 0xFFFF0_0000. This allows exception handlers resident in ROM during hardware initialization and then the subsequent use of RAM based exception handlers. When porting code to the 400 family the 600 family choices for exception vector locations are available via the flexibility of the EVPR.

4.1 Exception Vector Offsets With Different Usage

The exception vector offset of at 0x0100 is used for the critical interrupt exception type of the 400 family while it is used as the system reset exception type in a 600 family implementation. The critical interrupt is a 400 family implementation specific extension to the architecture that has no equivalent in the 600 family.

The exception vector located at 0x1000 is used for the programmable interval timer (PIT) function of the 400 family, and is used for instruction translation miss event in the 600 family. Additionally in the 600 family, the exception vector offsets of 0x1010 and 0x1020 are used for the fixed interval timer (FIT) and watchdog timer respectively. In the 400 family, this address space remains part of the 600 family instruction-translation-miss vector.

The exception vector located at 0x1100 is used for the data TLB miss interrupt of the 400 family. The 600 family has separate vectors for load vs. store and for data translation misses. Vector offset 0x1100 is used for load misses and 0x1200 is used for store misses.

The exception vector located at 0x1200 is used for the instruction TLB miss function of the 400 family, but is used for the data-store-translation miss event for the 600 family implementations.

When porting code from the 600 family, be aware that the 400 family has a unique specific vector offset of 0x2000 that is used for debug events. This offset is not used at all for exception vectors in the 600 family, therefore this memory address range could have been used for other purposes. Any code or data located here would need to be moved so that at least a word starting at 0x2000 would be available for a branch instruction to the exception handler.

4.2 Exception addresses specific to the 400 family

Table 2 – 400 Family Unique Exception Vectors

Offset	SRR#	Type of Interrupt	Cause
0100	2/3	Critical Interrupt Pin	Critical Interrupt pin
1000	0/1	Programmable Interval Timer	Posting of an enabled Programmable Interval Timer interrupt in the Timer Status Register (TSR)
1010	0/1	Fixed Interval Timer	Posting of an enabled FIT interrupt in the TSR
1020	2/3	Watchdog Timer	Posting of an enabled first time-out of the watchdog timer in the TSR
1100	0/1	Data TLB Miss	Valid matching entry for the EA and process ID of an attempted data access is not found in the TLB.
1200	0/1	Instruction TLB Miss	Attempted execution of an instruction at an address and process ID for which a valid TLB entry does not exist.

2000	2/3	Debug	Debug Events when in Internal Debug Mode
FFFFFFFC	N/A	System Reset	System Reset pin, enabled 2nd watchdog time-out

4.3 600 Family Specific Exceptions

The 600 family MSR[RI] bit indicates if the interrupt is recoverable, i.e. if execution of the exception causing instruction can be continued. Before any 600 family exception handler attempts to return to the non-interrupt code, it should insure the RI bit indicates recoverability.

The floating-point not enabled exception is 600 family specific since none of the 400 family have floating point units.

The Instruction Address Breakpoint at offset 0x1300 is analogous to the 400 family debug exception at 0x2000. The interrupt handler routine needs to be changed to work on the 400 family; in part to handle the additional debug events that are possible.

The 600 family System Management interrupt at 0x1400 does not have an equivalent exception in the 400 family.

Table 3 - PowerPC 600 Family Unique Exception Vectors

Offset	SRR#	Type of Interrupt	Cause
0100	0/1	System Reset	A system reset is caused by the assertion of SRESET or HRESET. HRESET causes branch to 0xFFFF0_0100
0800	0/1	Floating-Point unavailable	Attempt to execute a floating point instruction when MSR[FP]=0 (Also available on 405 and 440 with APU)
0900	0/1	Decrementer	Most significant bit of the decremter (DEC) register transitions from 0 to 1. MSR[EE] must be set.
0D00	0/1	Trace	MSR[SE]=1 or when currently completing instruction is a branch and MSR[BE]=1
0E00	0/1	Floating Point Assist	May be used by some PowerPC implementations for floating-point assist exceptions
1000	0/1	Instruction Translation Miss	Effective address for an instruction fetch cannot be translated by the ITLB
1100	0/1	Data Load Translation miss	Effective address for a data load operation cannot be translated by the DTLB
1200	0/1	Data Store Translation Miss	Effective address for a data store operation cannot be translated by the DTLB or a DTLB hit occurs, and the change bit in the PTE must be set due to a data store

			operation
1300	0/1	Instruction Address Breakpoint	Address bits 0-29 in the IABR matches the next instruction in the completion unit and the IABR enable bit (bit 30) is set to 1
1400	0/1	System Management Interrupt	SMI input signal is asserted and MSR[EE] = 1

5 Time Base Differences

The PowerPC architecture defines a 64-bit incrementing register for use as a time base and all implementations have the prescribed 64-bits except the 403GA which uses 56 bits.

The frequency of incrementing the time base register is implementation dependent. The 600 family processors time base is incremented at 1/4 the bus clock rate. The 400 family time base is incremented at either the execution unit clock rate or the user provided external clock rate. The 403GCX core rate can be twice that of the system bus clock rate and the time base is incremented at this core clock rate.

The instructions for writing to the time base are not implementation dependent, thus code written to set the time base on a 32-bit PowerPC implementation will work correctly on a 64-bit implementation whether running in 64 or 32-bit mode.

The time base read method is implementation dependent. The 600 family, 401x2 and 405 time base is read using different SPR numbers than those used for writing the 32-bit TBU and TBL registers. User privilege-level access is read-only using the **mftb** instruction to read TBL and **mftbu** to read TBU. Supervisor privilege-level programs can write the registers using the **mttbl** and **mttbu** to write the TBL and TBU registers, respectively.

Access to the time base registers in the 401M1 core and 403 use 4 time base registers. Instead of the 600 family's TBU and TBL registers, these processors have the functionally equivalent registers named TBHI and TBLO which can be read and written by supervisor privilege-level programs. User privilege-level programs may only read the time base using the registers TBHU and TBLU. All time base register access is performed using the **mtspr** and **mfspr** instructions. The 401M1 core supports use of either the 403 family method or the 600 family method of access.

The Book E architecture does not have a **mftb** instruction. The 440 core is Book E compliant and so uses the 403 method of access.

6 Supported Endian Modes

The PowerPC architecture specifies big-endian as the normal memory access order. The 401 also supports true little-endian and PowerPC little-endian memory access ordering. The 403 and 600 family add only PowerPC little-endian. The 405 supports big-endian and true little-endian operation.

7 Floating-Point Support

The 600 family has a floating-point unit to perform floating-point operations with hardware assistance. The 400 family handles floating-point instructions by using the integer unit to provide emulation via program functions that are invoked when a floating-point instruction is encountered. The functions are invoked by having an illegal instruction exception handler that recognizes the opcode and invokes the corresponding floating-point routine. A higher performance approach is to directly call subroutines of a floating-point emulation run-time library, but binary program compatibility is sacrificed.

8 APU Support

The 401 and 405 cores have support for an Auxiliary Processing Unit (APU) which is tightly coupled to the processors execution unit. It can be used, for example, to add a floating-point operation unit to speed floating-point math operations. Or a Multiply with Accumulate (MAC) unit for DSP type instructions.

The 401 cores do not have the ability to load or store data to the APU. The 405 core provides load and store operations using specific instructions for moving opcodes and parameter to the APU. These are defined as extensions to the PowerPC architecture defined instruction set.

9 Debug Registers

Debug facilities are implementation dependent. In general, the 400 family implementations all provide the ability to cause a debug event if either a instruction or data address of interest is accessed. The 600 family provides only an instruction address breakpoint controlled by using the IABR.

The 401 provides a single instruction address and a single data address debug event generation ability. The are controlled using the DBCR and DBSR registers.

The 403 provides two instruction address and two data address debug event abilities, again controlled using the DBCR register and getting status via the DBSR register. However there is also a "first event" counter and event sequencing ability.

The 405 has four instruction address registers which can be used individually, or to define an inclusive or exclusive address range. It has two data address registers that can also provide discrete address matching or a range of data movement address matching capability. Additionally two registers are provided to further qualify the generation of a debug event based on the value of the data being loaded or stored. The debug functionality is controlled using the DBCR0 and DBCR1 registers. Status is indicated in the DBSR register.

10 Power Management

The 400 and 600 families implement somewhat similar power management functionality, but they use different control mechanisms.

The 400 family uses the MSR[WE] Wait-state Enable bit to control place the processor in the wait state. When an exception is taken, the Wait State is removed.



The 600 family uses a the MSR[POW] bit to control the enabling and disabling of power management. Exceptions automatically disable power management when the interrupt is taken. Its original state is saved in SRR1.

The 700 family provides for programmable power states; full power, doze, nap and sleep. Software selects these modes by setting one (and only one) of the three power saving mode bits in the HID0 register. A hardware interrupt causes the transfer of program flow to an interrupt handler that then invokes the appropriate power saving mode. There is also the ability to use a decremter register to enter the nap or doze mode for a predetermined amount of time and then return to full power mode using a decremter interrupt.

11 Miscellaneous Processor Specific Registers

- PVR - The PVR value is unique for each processor implementation

11.1 600 Family Specific Registers

- SPRG0-SPRG3 - Only the 600 family has these general-purpose SPR registers, which the program can use as needed. Conventionally, for the 600 family processors SPRG0 is reserved as register containing the value for a stack pointer to be used by the first-level exception handler. SPRG1 is then used in first-level exception handlers to save a GPR, which is then used as a stack pointer for saving other registers by copying the content of SPRG0 to the saved GPR. The EABI specifies that GPR1 is used as the stack pointer and is always valid, so saving it using by means of an SPRG, or any other method, is not necessary when running EABI compliant programs.
- HID0 – Hardware Implementation Dependent register 0 is used for enabling and determining checkstop sources, bus parity control, cache control and configuration, and power management. Its functionality varies with each 600 family implementation.
- HID1 – Is used to configure the PLL and is accessed with **mtspr** and **mfspr**. Its functionality varies with each 600 family implementation.
- EAR – The External Access Register is an optional SPR in the PowerPC Operating Environment Architecture (OEA) that controls access to the external control facility. It identifies the target device for the external control facility, which communicates with special external devices. The 400 family has no equivalent facility.
- PIR – The Processor ID Register is optional register in the OEA that can be used by the operating system to assign an ID to a processor. It is also used when the processor communicates with I/O devices.

12 Conclusion

The PowerPC Architecture provides a high degree of code portability between different implementations even when the implementations have significant differences in their intended applications. Porting code between implementations is a straightforward and manageable job that is primarily limited to the kernel

and other supervisory mode code. Applications typically require almost no changes in order to be moved to a different target processor than that which they were originally developed for.

Porting between processors can be greatly facilitated by the use of structured programming methods, which then minimizes the amount of code needing modification. For example, if all access to the Time Base is done using a single function which is used by the entire application and operating system, only that function needs to be modified when porting to another PowerPC processor.

13 References

The following references contain more information regarding the PowerPC architecture and specific implementation details.

- *The PowerPC Architecture: A Specification for a New Family of RISC Processors*, IBM 5/94. Published by Morgan Kaufmann Publishers, Inc. San Francisco.
- *Programming Microprocessor Family: The Programming Environments*, IBM and Motorola 9/94. Available on-line at <http://www.chips.ibm.com/techlib/products/powerpc/manuals/>
- *The IBM PowerPC Embedded Environment: Architectural Specifications for IBM PowerPC Embedded Controllers*. Available from <http://www.chips.ibm.com/techlib/products/powerpc/manuals/>
- *Book E: PowerPC Architecture Enhanced for Embedded Applications*. Available from http://www.chips.ibm.com/products/powerpc/booke_rm.pdf
- *PowerPC 740 and PowerPC 750 User's Manual*, IBM 6/98. Available on-line at <http://www.chips.ibm.com/techlib/products/powerpc/manuals/>
- *PowerPC 403GCX Embedded Controller User's Manual*, IBM 5/97. Available on-line at <http://www.chips.ibm.com/techlib/products/powerpc/manuals/>

© International Business Machines Corporation, 1999

All Rights Reserved

* Indicates a trademark or registered trademark of the International Business Machines Corporation.

** All other products and company names are trademarks or registered trademarks of their respective holders.

IBM and IBM logo are registered trademarks of the International Business Machines Corporation.

IBM will continue to enhance products and services as new technologies emerge. Therefore, IBM reserves the right to make changes to its products, other product information, and this publication without prior notice. Please contact your local IBM Microelectronics representative on specific standard configurations and options.

IBM assumes no responsibility or liability for any use of the information contained herein. Nothing in this document shall operate as an express or implied license or indemnity under the intellectual property rights of IBM or third parties. NO WARRANTIES OF ANY KIND, INCLUDING BUT NOT LIMITED TO THE IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE ARE OFFERED IN THIS DOCUMENT.

