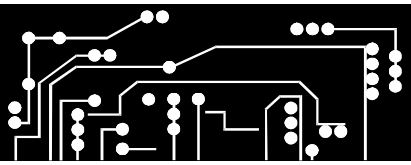
PowerPC Embedded Processors
Application Note



IBM PowerPC™ 440 Microprocessor Core Programming Model Overview

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

October 4, 2001

Abstract – This application note gives an overview of the programming model of the IBM PowerPC 440 processor core. It is of interest to software developers creating both application and operating system programs. The IBM PowerPC 440 is an implementation of the PowerPC Book E architecture. The programming model defines the instruction set, operations and registers available for use in both system and application level programs. The Book E architecture was defined in cooperation with Motorola and other PowerPC interests to provide a standardized enhancement for embedded applications to the previous PowerPC architecture, now referred to as the 'classic' PowerPC architecture. References to the PowerPC architecture infer that the feature exists in both the classic and Book E implementations.

1 Overview

The IBM PowerPC 440 (PPC440) is an implementation of the PowerPC Book E architecture; therefore, its programming model is in part, described in the Book E architecture specification. The term 'architecture' refers to the formal description of the operation and facilities of a processor core implementation. A 'programming model' is the programmer view of how that implementation is used by a program. This application note covers the following aspects of the programming model:

- Data types
- Register types and usage
- Memory model
- Instruction set
- Program Flow
- Interrupts
- Timers
- Cache

The PPC440 is a 32-bit implementation of the Book E architecture meaning there are 32 bits in each internal register and that there are 32 address bits available for memory references. However, the MMU produces 36-bit physical addresses of which 2^32 can be referenced by a program using a particular MMU entry.

The PowerPC architecture specifies that two levels of privilege for program execution: user-mode and supervisor-mode. In user-mode, access to most registers including system control registers is denied. It is intended that most application programs be executed in user-mode so that they are protected from errant register changes made by other user-mode programs.

Book E provides for binary compatibility for 32-bit PowerPC application programs. Thus, existing application code, which uses only the instructions defined by the User Instruction Set Architecture book of the classic PowerPC architecture, will execute without modification on PowerPC Book E implementations. Book E specifies that binary instruction compatibility not necessarily be provided for privileged (supervisor-mode) PowerPC instructions.

2 Data Types and Alignment

The PowerPC architecture defines scalar (integer) data type sizes as shown in Table 1.

Table 1 - PowerPC Scalar Data Types

Data type	Size (bytes)
Byte	1
Halfword	2
Word	4
Doubleword	8
Quadword	16
String	1 – 128

Although not a data type used by the load or store integer instructions, a quadword is defined to have a size of 16 bytes. The doubleword type is available only in 64-bit implementations and is shown for completeness. Since the PPC440 is a 32-bit implementation, it is not available in its programming model.

3 Register Types and Usage

The PowerPC architecture defines five register types: General Purpose, Special Purpose, Device Control, Machine State, and Condition. All register, address bus and data bus numbering in IBM implementations of the PowerPC architecture follow the convention of labeling the most significant bit as bit 0 (zero).

Table 2 lists the registers available to programs executing in user-mode.

Table 2 - PowerPC 440 User-mode Registers

Register	Type	Used for:
GPR0 – GPR31	GPR	General-Purpose Registers
CR	CR	Condition Register
XER	SPR	Integer Exception Register
LR	SPR	Link Register
CTR	SPR	Count Register
SPRG4 – SPRG7	SPR	Special Purpose Registers General
USPRG0	SPR	User Special Purpose Register 0
TBU	SPR	Time Base – Upper 32-bits
TBL	SPR	Time Base – Lower 32-bits

The PPC440 has a register file of thirty-two 32-bit wide general-purpose registers (GPR). GPRs are used as the target and source for all arithmetic and logical operations, as well as for load and store accesses to storage. In order to use a value that is in storage, it must be read into a GPR using one of the load

instructions. Likewise, to save a value to memory, a store of a GPR's contents would be performed using one of the store instructions.

The GPRs are also used as targets and sources for most of the instructions that read and write the other register types. In other words, access to Special Purpose Registers, Device Control Registers, the Machine State Register, and the Condition Register is through GPRs

Only the Condition Register (CR) is of the condition register type. CR reflects the result of arithmetic and logical instruction execution into four bits: Equal, Greater Than, Less Than and Summary Overflow. These bits can then be used to control program flow with conditional branches. The CR consists of eight 4-bit CR fields, CR0 – CR7. Having multiple fields improves performance by allowing the combined testing of the results from multiple compares into a single conditional branch. It also helps to avoid blocking by the PPC440's multiple execution units which otherwise might occur due to contention for access. An explicit form of each arithmetic and logical instruction enables updating of the CR0 condition register field. The "record" form of instructions is specified by appending a "." (a period), to the basic instruction mnemonic. The CR is accessible to code executing in either the user or supervisor-mode.

The Special Purpose Register (SPR) type is for registers used in configuring and monitoring of processor resources and instruction execution results. Most SPRs may not be accessed while in user-mode. Table 2 shows the SPRs that user-mode programs may access either directly or indirectly. SPRs are directly accessed using the **mtspr** (move to special purpose register) and **mfspr** (move from special purpose register) instructions. Privileged SPRs are used to control additional processor resources such as the debug facilities, timers, interrupts, memory management and caches.

Certain SPRs may be indirectly accessed as a side effect of the execution of instructions. For example, the Integer Exception Register (XER) is an SPR that is updated with arithmetic status information such as carry and overflows after execution of certain versions of integer arithmetic and shift instructions.

The Link Register (LR) is an SPR used for support of subroutine execution. For example, to go to a subroutine, use the **bl** (branch and link) instruction to branch to the subroutine. This places the address of the instruction that follows the **bl** into LR. Later, when it is time to return from the subroutine, executing the **blr** (branch to link register) instruction uses the contents of the LR to branch back to the main program. If nested subroutines are called, the Link Register (LR) will need to be saved before each call to the next subroutine and then restored before branching back to calling subroutine.

The Count Register (CTR) is an SPR that is used either as a loop counter that is decremented and tested by certain branch instructions, or as to specify the target address for the **bcctr** (branch conditional to CTR) instruction. The **bcctr** instruction can therefore be used to branch to any 32-bit address. The CTR is written from a GPR by using the **mtspr** instruction. Extended mnemonics are shorthand versions of actual instructions that assemblers recognize in order to simplify writing assembly language programs. For example **mtspr ctr,r4** is equivalent to the extended mnemonic **mtctr r4**. Similar instructions are available for accessing most of the SPRs.

The Special Purpose Registers General (SPRG4-SPRG7, USPRG0) are a special case of SPRs. They are accessible by both user-mode as well as supervisor-mode programs. However, SPRG4-SPRG7 can only be read in user-mode. USPRG0 can be either read or written in user-mode. These registers are provided for general purpose, system-dependent software use. An example is use by an operating system to save the values of GPRs during interrupt handlers or to pass read only values to user-mode programs. The **mfspr** instruction transfers a value into a GPR from an SPR. Conversely the **mtspr** instruction is used to write a value to USPRG0.

The Time Base is a 64-bit incrementer that is an architecturally required resource for all PowerPC implementations. Book E standardizes implementation access methods and instructions. The Time Base consists of two cascaded 32-bit counters. The Time Base Upper (TBU) register accesses the upper 32-

bits and the Time Base Lower (TBL) register accesses the lower 32-bits. User-mode programs may only read the time base registers using the **mfspr** instruction. Information that is more detailed will be presented in the timer discussion.

The Machine State Register (MSR) is its own unique machine state register type. Access to it is only allowed while executing in the supervisor-mode. It controls the state of the configuration and state of the processor. The MSR can be written from a GPR using the **mtmsr** (move to machine state register) instruction. The contents of the MSR can be read into a GPR using the **mfmsr** (move from machine state register) instruction. The MSR[EE] bit can be set or cleared atomically using the **wrtee** (write external enable) or **wrteei** (write external enable immediate) instructions. The MSR contents are also automatically saved, altered, and restored by the interrupt-handling mechanism.

Device Control Registers (DCR) are implementation specific integrated peripheral control and status registers that are attached to the Device Control bus. DCRs may be used to control various on-chip system functions, such as the operation of on-chip buses, peripherals, and certain processor core behaviors. They are on-chip registers that exist architecturally and physically outside the PPC440 core, and thus are not specified by the Book-E Enhanced PowerPC Architecture. Rather, PowerPC Book-E defines the existence of DCR address space and the instructions used to access the DCRs, the **mtdcr** (move to device control register) and **mfdcr** (move from device control register) instructions.

All other registers in the PPC440 are privileged and may only be accessed while a program is executing in supervisor-mode. An attempt to execute a privileged instruction or to access a privileged register while in user-mode causes a Privileged Instruction exception type Program Interrupt to occur.

4 Memory Model

The PPC440 uses 32-bit effective addresses that are translated to 41-bit virtual addresses and then to 36-bit real (physical) addresses, which provides a 64 GB real address space. The Memory Management Unit may not be disabled, so address translation and attribute controls are always in effect. It implements address mapping and attribute control for up to 64 simultaneously active memory page definitions.

Maintenance of TLB entries is completely under software control as there is no hardware assisted page table lookups. System software determines the TLB entry replacement strategy and the format and use of any page table information. A TLB entry contains all of the information required to identify the page, specify the address translation, control the access permissions, and designate the storage attributes.

Eight memory page sizes (1KB, 4KB, 16KB, 64KB, 256KB, 1MB, 16MB, 256MB) are simultaneously supported, such that at any given time the TLB can contain entries for any combination of page sizes. Each TLB entry provides separate user-mode and supervisor-mode read, write, and execute permission controls for the memory page associated with the entry. Each region also has write-through, caching inhibition, speculative access, and byte order (endianness) storage attribute controls. Four user-definable storage attribute controls are also available for use as desired by a PPC440 based design.

TLB entries can be specified that create 1:1 mappings from effective (virtual) to real (physical) addresses. This can be useful for simplified management of a system's memory map, or when virtual memory page swapping techniques are not needed. An attempt to access an address for which no TLB entry exists causes either an Instruction (for fetches), or Data (for load/store accesses), TLB Error exception.

The TLB management instructions are used to read and write entries of the TLB array, and search the TLB array for an entry that translates a given virtual address. The TLB is managed by using the privileged instructions listed in table 3.

Table 3 - PowerPC 440 MMU Control Instructions

Instruction	Used for:
tlbre	Reading a TLB entry
tlbsx[.]	TLB search – finds an entry for the specified address
tlbsync	Synchronizes multiprocessor TLBs – treated as no-operation
tlbwe	Writing a TLB entry

5 Instruction Set

The PowerPC Book E architecture defines the required instructions, their mnemonics and operation so that porting code among various PowerPC implementations is straightforward. Some instructions in the architecture are optional; therefore, implementations do not have to support them. For example, the PPC440 does not have support for floating-point in the processor core. Therefore, unless an auxiliary processor that implements floating-point instructions is attached, the floating-point registers are not available.

In the PPC440's 32-bit implementation of Book E architecture, all instructions are 32-bits in size. The first six bits, b0 – b5 define the primary opcode. The remaining 26 bits are used for operands which define source and target registers, immediate values, address offsets, etc. All instructions execute in 1 cycle except for divides, multiplies (3 cycle latency and single cycle throughput) and load/store multiples (1 cycle per word if cached). Branch and condition register logical instructions can execute in zero cycles if folded, i.e. overlapped with a previous instruction's execution. Since the PPC440 has a dual-issue instruction pipeline, more than one instruction can be completed per cycle.

The PowerPC Book-E architecture provides for allocated instructions, which are instructions available for implementation-dependent and/or application-specific purposes. Specific allocated instructions are not defined as part of the PowerPC Book-E architecture. The PPC440 does implement a number of them, which architecturally, are considered allocated instructions, since they use opcodes that are within the allocated class of instructions. However, all of the allocated instructions that are implemented within the PPC440 core are "standard" for IBM's 4xx family of PowerPC embedded controllers and so, are not unique to the PPC440.

Table 4 shows the PPC440 instruction types organized by functional categories.

Table 4 - PowerPC 440 Instruction Categories

Functional Category	Instruction Types
Integer Storage Access	load, store
Integer Arithmetic	add, subtract, multiply, divide, negate
Integer Logical	and, andc, or, orc, xor, nand, nor, xnor, extend sign, count leading zeros
Integer Compare	compare, compare logical
Integer Trap	trap
Integer Rotate	rotate and insert, rotate and mask
Integer Shift	shift left, shift right, shift right algebraic
Branch	branch, branch conditional, branch to link, branch to count
Condition Register Logical	crand, crandc, cror, crorc, crnand, crnor, crxor, crxnor
Register Management	Move to/from SPR, move to/from DCR, move to/from MSR, write to external interrupt enable bit, move to/from CR

System Linkage	System call, return from interrupt, return from critical interrupt
Processor Synchronization	Instruction synchronize
Cache Management	Data allocate, data invalidate, data touch, data zero, data flush, data store, instruction invalidate, instruction touch
TLB Management	Read, write, search, synchronize
Storage Synchronization	Memory synchronize, memory barrier
Allocated Arithmetic	Multiply-accumulate, negative multiply-accumulate, multiply halfword
Allocated Logical	Detect left-most zero byte
Allocated Cache Management	Data congruence-class invalidate, instruction Congruence-class invalidate
Allocated Cache Debug	Data read, instruction read

6 Program Flow Control

Programs can alter the sequential execution of instructions using the branch instructions. These instructions unconditionally or conditionally branch to a specified address. Conditional branch instructions test a condition code (LT, GT, EQ, SO) set in the CR by a previously executed instruction and branch accordingly. Conditional branch instructions can also decrement and test the Count Register (CTR) as an additional condition to a branch determination. The target address for a branch either is a displacement from the current instruction address, an absolute address, or contained in the LR or CTR. All branch instructions can optionally save a return address (the address of the instruction after the branch) in the Link Register (LR).

Table 5 lists the branch instructions in the PPC440. In the table, the syntax “[l]” indicates that the instruction has both a “link register update” form, which updates LR with the address of the instruction after the branch, and a “non-link register update” form. Similarly, the syntax “[a]” indicates that the instruction has both an “absolute address” form, in which the target address is formed directly using the immediate field specified as part of the instruction, and a “relative” form in which the target address is formed by adding the specified immediate field to the address of the branch instruction.

Table 5 - Program Flow Instructions

Instruction	Used for:
b[l][a]	Branch unconditionally
Bc[l][a]	Branch conditionally
bcctr[l]	Branch conditionally including CTR condition test
bclr[l]	Branch conditionally to address in Link Register

As a form of program flow control, the storage synchronization instructions allow software to enforce ordering amongst the storage accesses caused by load and store instructions. By default, loads and stores are “weakly-ordered” in that the processor is architecturally permitted to perform loads and stores out-of-order with respect to their sequence within the instruction stream. However, if a storage synchronization instruction is executed, then all storage accesses needed by instructions prior to the synchronizing instruction must be performed prior to any storage accesses needed by instructions that come after the synchronizing instruction. Table 6 shows the storage synchronization instructions in the PPC440.

Table 6 - Storage Synchronization Instructions

Instruction	Used for:
msync	Memory synchronize – ensures preceding instructions complete
mbar	Memory barrier – ensures preceding loads/stores complete

7 Register Access

All register types other than the GPR type are examined and modified by being read and written to/from a GPR.

Table 7 shows the CR register access instructions **mtcr** and **mfcr** that are used to move the contents of a GPR to and from the CR register. The **mcrx** instruction is used to move the value in the XER register into the CR for later usage. The contents of a CR field may be copied to another CR field using **mcrf**.

Table 7 - CR Register Access and Modification Instructions

Instruction	Used for:
mcrf	Move Condition Register Field to another CR field
mcrxr	Move to Condition Register field from XER
mfcr	Move From Condition Register to a GPR
mtcrf	Move to Condition Register Field from a GPR

Table 8 shows the instructions, which are used to access DCRs. DCRs, are used to read status information from and write configuration settings to peripherals. DCRs are read into a GPR using the **mfdcr** instruction. DCRs are written from a GPR using the **mtdcr** instruction.

Table 8 - DCR Register Access Instructions

Instruction	Used for:
mfdcr	Move from a DCR to a GPR
mtdcr	Move to a DCR from a GPR

Table 9 shows the instructions used to access the MSR. The MSR contains processor status and operational controls. The MSR is read into a GPR using the **mfmsr** instruction. The MSR is written from a GPR using the **mtmsr** instruction. The MSR[EE] bit, which enables external interrupts, may be directly manipulated using the **wrtee** and **wrteei** instructions

Table 9 - MSR Register Access Instructions

Instruction	Used for:
mfmsr	Move From Machine State Register to a GPR
mtmsr	Move To MSR from a GPR
wrtee	Write External Enable - enable/disable external interrupts
wrteei	Write External Enable with an Immediate value

Table 10 shows the instructions used to access SPRs. SPRs are read into a GPR using the **mfspr** instruction. SPRs are written from a GPR using the **mtspr** instruction.

Table 10 - SPR Register Access Instructions

Instruction	Used for:
mfspr	Move From Special Purpose Register to a GPR
mtspr	Move To Special Purpose Register from a GPR

8 Interrupts

Exceptions, which are also known as interrupts, can if enabled, cause the processor to interrupt the current instruction execution sequence. They are the result of either program execution or external events.

There are two classes of Interrupts: critical and non-critical. Critical interrupts have a higher priority than non-critical interrupts and are caused by things like debug events, machine checks, the critical interrupt level assigned to an external interrupt pin, and the watchdog timer. Non-critical interrupts are caused by instruction execution; external interrupts assigned to the non-critical interrupt class, and the programmable and fixed interval timers.

Interrupts are grouped in to types and each type has its own interrupt handler address offset. The Interrupt Vector Prefix Register (IVPR) defines the base address of the interrupt handlers by specifying the upper 16-bits of the interrupt vector address.

The interrupt processing registers are shown in Table 11. They are the Save/Restore Registers (SRR0–SRR1), Critical Save/Restore Registers (CSRR0–CSRR1), Data Exception Address Register (DEAR), Interrupt Vector Offset Registers (IVOR0–IVOR15), Interrupt Vector Prefix Register (IVPR), and Exception Syndrome Register (ESR). Only a subset of these registers are used by any given interrupt type.

Table 11 – Interrupt processing registers

Register	Used for:
SRR0	Return address from non-critical interrupts
SRR1	Saved machine state register (MSR) for non-critical interrupts
CSRR0	Return address for critical interrupts
CSRR1	Saved machine state register (MSR) for critical interrupts
DEAR	Address of data access that caused an exception
ESR	What was the cause of the access exception
IVPR	The high order bits of Interrupt vector handlers location in memory
IVOR0-IVOR15	Vectors offset for the base address in IVPR

9 Timers

The PPC440 contains a Time Base and three timers: a Decrementer (DEC), a Fixed Interval Timer (FIT), and a Watchdog Timer. The Time Base is a 64-bit counter which increments at either equal the processor core clock rate or as driven by a separate timer clock input to the time base. The DEC is a 32-bit register that is decremented at the same rate at which the Time Base is incremented. The FIT generates periodic interrupts based on the transition of a selected bit from the Time Base. Similar to the FIT, the Watchdog Timer also generates a periodic interrupt based on the transition of a selected bit from the Time Base. All these timers are monitored using the Timer Status Register (TSR) and controlled using the Timer Control Register (TCR). The TCR contains the interrupt enables, watchdog period selection, and time-out event configuration.

These timer registers have specific instructions for access. The Time Base is accessed via two 32-bit special purpose registers, TBL which is the low-order 32-bits and TBU which is the high-order 32 bits of time base. The following code provides an example of reading the Time Base.

```
loop:
  mfspr  Rx,TBU    # read TBU into GPR Rx
  mfspr  Ry,TBL    # read TBL into GPR Ry
  mfspr  Rz,TBU    # read TBU again, this time into GPR Rz
  cmpw   Rz,Rx     # see if old = new
  bne    loop      # loop/reread if rollover occurred
```

The comparison and loop ensure that a consistent pair of values is obtained in case of a carry from the TBL to the TBU.

10 Processor Control

The PPC440 core provides several registers for general processor control and status. Access to these registers is privileged.

The Machine State Register (MSR) controls interrupts, address translation and other processor functions.

The Processor Version Register (PVR) indicates the specific implementation of a processor. The PVR is a read-only register typically used to identify the specific processor core and chip implementation. Software can read the PVR to determine the processor core and chip hardware features. The PVR can be read into a GPR using **mfspr**.

The Processor Identification Register (PIR) indicates the specific instance of a processor in a multi-processor system. The PIR is a read-only register that uniquely identifies a specific instance of a processor core, within a multi-processor configuration. This enables software to determine which processor it is running on. This capability is important for operating system software within multiprocessor configurations. The PIR can be read into a GPR using **mfspr**.

The Core Configuration Register 0 (CCR0) controls a number of special processor specific functions, including data cache and auxiliary processor operation, speculative instruction fetching, trace, and the operation of the cache block touch instructions. The CCR0 is written from a GPR using **mtspr**, and read into a GPR using **mfspr**.

The Reset Configuration (RSTCFG) is a read-only register that reports the values of certain fields of the translation look aside buffer (TLB) as supplied at reset.

11 Cache

The PPC440GP provides separate instruction and data cache controllers and arrays. These allow concurrent access and therefore minimize pipeline stalls. The instructions shown in table 12 and table 13 are provided to control the operation of the data and instruction caches. They are used to fill, flush, invalidate, and zero data cache blocks. There are also instructions to assist in debugging that can be used to read the tag and data arrays.

Table 12 - Data Cache Management Instructions

Instruction	Used for:
dcba	Data Cache Block Allocate – treated as no operation
dcbf	Data Cache Block Flush – copy block to storage and mark invalid
dcbi	Data Cache Block Invalidate – block marked invalid
dcbst	Data Cache Block Store – copy block to storage and mark unmodified
dcbt	Data Cache Block Touch – pre-read data into cache block
dcbtst	Data Cache Block Touch for Store – pre-read data for future storage
dcbz	Data Cache Block Zero – set cache block to zero

Table 13 – Instruction Cache Management Instructions

Instruction	Used for:
icbi	Instruction Cache Block Invalidate – mark block invalid
icbt	Instruction Cache Block Touch – pre-read instructions into cache

12 Conclusion

The PowerPC 440 provides a high performance implementation of the Book E architecture while providing backwards compatibility with existing PowerPC application programs. The Book E instruction set is enhanced to support real-time embedded control and supervisory applications.

13 References

The following references are presented in a suggested reading order progressing from a comprehensive overview through the standards documents from most PowerPC specific to most general.

- *The PowerPC 440 Core,* IBM, September 1999. This technical white paper discusses the architecture and implementation of the first Book E compliant microprocessor core from IBM. http://www.chips.ibm.com/products/powerpc/cores/440_wp.pdf

- *Book E – PowerPC Architecture Enhanced for Embedded Applications*, IBM, March 2000. http://www-3.ibm.com/chips/techlib/techlib.nsf/pages/main

Printed in the United States of America, October 4, 2001

IBM Microelectronics Division
1580 Route 52, Bldg. 504
Hopewell Junction,
NY 12533-6351

The IBM home page can be found at http://www.ibm.com

The IBM Microelectronics Division home page can be found at http://www.chips.ibm.com