MOTOROLA
SEMICONDUCTOR
TECHNICAL DATA

Order this document
by MPC601UMAD/AD

MPC601

Addendum to

PowerPC™ 601 RISC Microprocessor User's Manual

This addendum describes additions and corrections to the *PowerPC 601 RISC Microprocessor User's Manual*. For convenience, this information is organized according to the chapters in the user's manual; however, there is no attempt to provide a comprehensive list of text that is affected by this information. These changes will be incorporated in the first revision of the user's manual.

Section 1: Chapter 1, "Overview"

This section describes additional information and corrections to Chapter 1, "Overview."

Section Number	
1.1.5	In the first sentence in this section replace Terabyte with Petabyte.
1.3.8.4	Replace Figure 1-6 with the following:

MOTOROLA

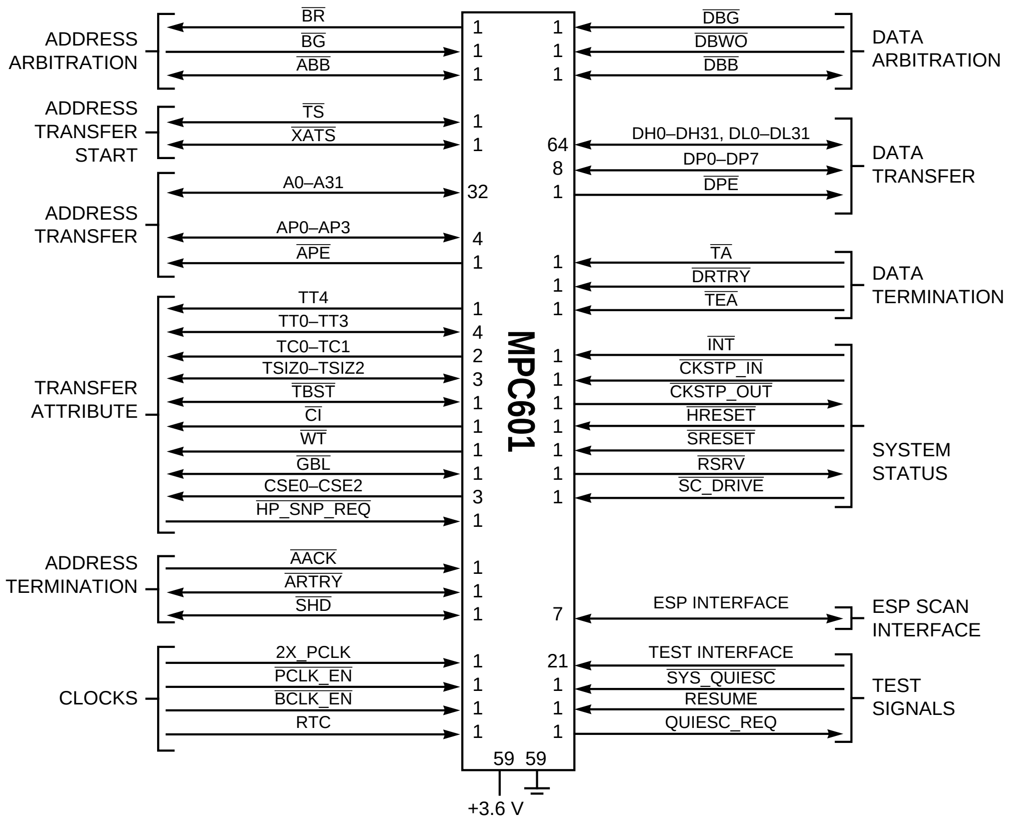


Figure 1-6. MPC601 Signal Groups

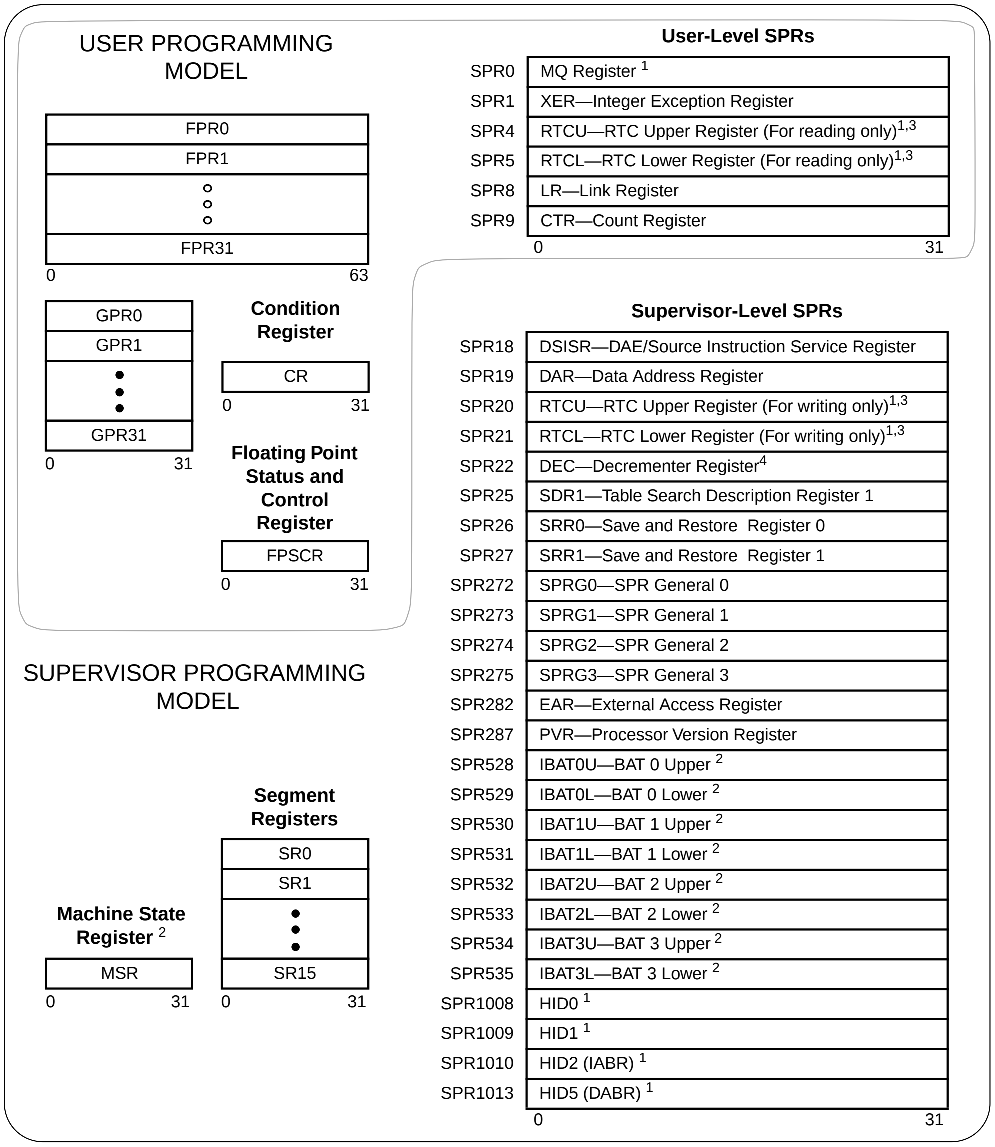
1.3.6.2 Replace the last sentence of the fourth paragraph with the following:

The processor ensures that the ITLB is consistent with the UTLB, and uses an LRU replacement algorithm when a miss is encountered.

Section 2: Chapter 2, “Registers and Data Types”

This section describes additional information and corrections to Chapter 2, “Registers and Data Types.”

2.1 Replace Figure 2-1 with the following:



User-Level SPRs

SPR0	MQ Register [1]
SPR1	XER—Integer Exception Register
SPR4	RTCU—RTC Upper Register (For reading only)[1,3]
SPR5	RTCL—RTC Lower Register (For reading only)[1,3]
SPR8	LR—Link Register
SPR9	CTR—Count Register
	0 31

Supervisor-Level SPRs

SPR18	DSISR—DAE/Source Instruction Service Register
SPR19	DAR—Data Address Register
SPR20	RTCU—RTC Upper Register (For writing only)[1,3]
SPR21	RTCL—RTC Lower Register (For writing only)[1,3]
SPR22	DEC—Decrementer Register[4]
SPR25	SDR1—Table Search Description Register 1
SPR26	SRR0—Save and Restore Register 0
SPR27	SRR1—Save and Restore Register 1
SPR272	SPRG0—SPR General 0
SPR273	SPRG1—SPR General 1
SPR274	SPRG2—SPR General 2
SPR275	SPRG3—SPR General 3
SPR282	EAR—External Access Register
SPR287	PVR—Processor Version Register
SPR528	IBAT0U—BAT 0 Upper [2]
SPR529	IBAT0L—BAT 0 Lower [2]
SPR530	IBAT1U—BAT 1 Upper [2]
SPR531	IBAT1L—BAT 1 Lower [2]
SPR532	IBAT2U—BAT 2 Upper [2]
SPR533	IBAT2L—BAT 2 Lower [2]
SPR534	IBAT3U—BAT 3 Upper [2]
SPR535	IBAT3L—BAT 3 Lower [2]
SPR1008	HID0 [1]
SPR1009	HID1 [1]
SPR1010	HID2 (IABR) [1]
SPR1013	HID5 (DABR) [1]
	0 31

[1] MPC601-only registers. These registers are not necessarily supported by other PowerPC processors.
[2] These registers may be implemented differently on other PowerPC processors. The PowerPC architecture defines two sets of BAT registers—eight IBATs and eight DBATs.The MPC601 implements the IBATs and treats them as unified BATs.
[3] The RTCU and RTCL registers can be written only in supervisor mode, in which case SPR20 and SPR21 are used.
[4] The DEC register can be read by user programs by specifying SPR6 in the **mfspr** instruction (for POWER compatibility).

Figure 2-1. Programming Model—Registers

2.1	The mechanism referred to for accessing SPRs is the set of Move to/from SPR instructions (**mtspr** and **mfspr**). These instructions are commonly used to access certain registers, while other SPRs may be more typically accessed as the side effect of executing other instructions.
2.1	The MSR register is 64 bits wide in 64-bit implementations and is 32 bits wide in 32-bit implementations.
2.2.4.1	Replace the first paragraph in this section with the following:
	In most integer instructions, when the Rc bit is set, the first three bits in CR0 are set by an algebraic comparison of the result to zero; the fourth bit of CR0 is copied from the XER[SO] bit. The **addic.**, **andi.** and **andis.** instructions set these four bits implicitly. These bits are interpreted as follows—if any portion of the result (the 32-bit value placed into the target register) is undefined, the value placed into the first three bits of CR0 is undefined.
2.2.5	The mechanism referred to for accessing SPRs is the set of Move to/from SPR instructions (**mtspr** and **mfspr**). These instructions are commonly used to access certain registers, while other SPRs may be more typically accessed as the side effect of executing other instructions.
2.2.5.3	In user-level access, RTCU and RTCL are read-only. The SPR numbers for the RTCU and RTCL differ depending upon whether the **mtspr** or **mfspr** instruction is used. For the **mfspr** instruction, RTCU is SPR4 and RTCL is SPR5. For the **mtspr** instruction, RTCU is SPR20 and RTCL is SPR21 (supervisor-level access only).
2.3.3	The MSR is not an SPR and should not be included in Table 2-15.
2.3.3	The RTCU and RTCL registers can be written to only in supervisor mode and the **mtspr** instruction requires a different SPR encoding. For the **mtspr** instruction, RTCU is SPR20 and RTCL is SPR21.
2.3.3.4	The PowerPC architecture defines the DEC register as supervisor-only access for both reads and writes. SPR22 is used for both reads and writes. The POWER architecture provides user-level read access, using SPR6. To ensure compatibility with subsequent PowerPC processors, the **mfspr** instruction should not be used in user-level.
2.3.3.10	The PVR is a read-only register that cannot be modified.
2.3.3.12.1	The HID0 register is set to x'8001 0080' by the hard reset operation. However, the state of the EMC bit depends on the results of the power-on diagnostics for the main cache array. This bit is set if the cache fails the built-in self test during the power-on sequence.
2.3.3.12.1	Checkstop enable bits can be set or cleared without restriction. If a checkstop source bit is set, it can be cleared; however, if the corresponding checkstop condition is still present on the next clock, the bit will be set again. A checkstop source bit can only be set when the corresponding checkstop condition occurs and the checkstop enable bit is set; it cannot be set via an **mtspr** instruction. That is, you cannot manually cause a checkstop condition.

2.3.3.12.2 Note that when HID1[8–9] = 10, the trap address of x'2000' has a base address indicated by the setting of MSR[IP]. This mode is valid for address comparisons and may produce unpredictable results when used with the HID single-instruction step mode.

2.4.3 Replace sentence 2 of paragraph 2 with the following:

All transfers of individual scalars between registers and storage are of double words. A subset of the 64-bit scalar (e.g., a byte) is not addressable in storage. As a result, to access any subset of the bits of a scalar, the entire 64-bit scalar must be accessed, and when a storage location is read, the 64-bit value returned is the 64-bit value last written to that location.

2.4.3 The following example shows how the byte ordering is changed from big- to little-endian mode by setting HID0[28] (*n* refers to the address):

```
<msr[ee] is off (zero) >

n       sync                  |Instructions
n+4     sync                  |accessed in
n+8     sync                  |big-endian mode
n+c     mtspr hid0(28)        |
n+10    sync                  |Instructions
n+14    sync                  |accessed in
n+18    sync                  |little-endian mode
```

The same instruction sequence can be used to go from little- to big-endian mode by clearing HID0[28].

Little-Endian Address Manipulation

In little-endian operations, the three least significant bits of an address are manipulated as described in Chapter 2, “Registers and Data Types,” to provide the appearance of a little-endian memory to the program for aligned loads and stores, as follows:

New_addr(29) <- EA(29) xor (word | half | byte)

New_addr(30) <- EA(30) xor (half | byte)

New_addr(31) <- EA(31) xor (byte)

The physical address used for a access generated by a load or a store to an operand that is less than a double word is modified as indicated. Addresses for aligned double word accesses and cache control operations are not modified since the endian mode has no effect on aligned accesses greater than one word.

The DAR and SRR0 will contain the program address (or the next sequential address, as appropriate) after all exceptions. If the processor is in little-endian mode, it will be a modified address. If the processor is in big-endian mode, the address is unmodified.

The T bit does not affect address manipulation or the detection of alignment exception conditions. Therefore I/O interface controller operations and BUID x'07F' segments receive the modified address. The **ecowx** and **eciwx** instructions are treated as no-ops if the T bit is set regardless of whether the MPC601 is in little-endian mode.

Because the MPC601 defines a cache block as 32 bytes, bits 27–31 of the address are not used for snooping. The program address should be specified, when an address is loaded into HID2 or HID5. That is, if the processor is in little-endian mode, a little-endian address should be specified, and if the processor is in big-endian mode, a big-endian address should be specified.

Little-Endian Alignment Exceptions

Additional alignment exception conditions can occur when the processor is in little-endian mode.

Load/store multiple operands (regardless of EA)

• **lmw**	**stmw**
• **lscbx*x***	**stswi**
• **lswi**	**stswx**
• **lswx**	

The new alignment exception conditions are prioritized with other alignment exceptions ahead of data access exceptions. For more information see Section 2.4.6.2 “Misaligned Scalars.”

Little-Endian Instruction Fetching

In little-endian mode, instructions are fetched in big-endian order; however, the instructions are swapped within a double word before being passed to the instruction queue, thus putting the instructions in little-endian order for execution. On exceptions, the MPC601 reports the correct effective address (as defined by the programming model or computed by a storage access instruction) regardless of the endian mode selected.

2.4.4 The second line of the program example is incorrect. Replace

```
double   b:    /* x'2122232422225262728' doubleword */
```

with the following:

```
double   b:    /* x'2122232425262728'   doubleword */
```

2.4.5 The MPC601 big- and little-endian mode operation differs from the PowerPC architecture in the following ways:

- Choice of big- or little-endian modes is provided through HID0[LM]—bit 28 of HID0. The PowerPC architecture defines two bits in the MSR for this purpose.

- The basic mode switching sequence requires three **sync** instructions followed by the **mtspr** access to HID0[28], followed by three more **sync** instructions. This sequence should be used whenever the state of this bit is changed.
- External and decrementer interrupts should be disabled before executing the sequence.
- The starting address of the sequence does not matter; however, the sequence cannot cross a protection boundary.
- In some cases the **mtspr** access to HID0[LM] can occur twice depending on the alignment of the instruction.
- In some cases not all of the **sync** instructions will actually be executed, depending on the starting address of the sequence.
- Although HID0[LM] can be switched dynamically, there are certain constraints (such as turning off translation and emptying the memory queues) that must be considered before the bit can be switched. Note that,when switching modes between tasks, this code sequence may not allow the MPC601 to operate at an optimal performance level.

Section 3: Chapter 3, "Addressing Modes and Instruction Set Summary"

This section describes additional information and corrections to Chapter 3, "Addressing Modes and Instruction Set Summary."

Section Number	
3.1.2	The first sentence in this section should include the **isync** instruction but should not include the **mtmsr** instruction.
3.3.4.2	In Table 3-9, in the descriptions of the Shift Left Word, Shift Right Word, and Shift Right Algebraic Word instructions the number of bits specified by **r**B should be **r**B(27–31) instead of **r**B(26–31).
3.4.3	The PowerPC architecture specifies that for the two floating-point convert to integer instructions, **fctiw** and **fctiwz**, both FPSCR(VXCVI) and FPSCR(VXSNAN) are set when the input operand is an SNaN. The MPC601 sets only FPSCR(VXCVI).
3.5.5	Add the following information:
	In future implementations, the load/store multiple instructions are likely to have greater latency and take longer to execute, perhaps much longer, than a sequence of individual load or store instructions that produce the same results.

3.5.6 Add the following information:

In future implementations, the integer move string instructions are likely to have greater latency and take longer to execute, perhaps much longer, than a sequence of individual load or store instructions that produce the same results.

3.5.7 Add the following information:

The paired use of the **lwarx** and **stwcx.** instructions allows programmers to emulate common semaphore operations such as test and set, compare and swap, exchange memory, and fetch and add.

The concept behind this part of the architecture is that a processor may load a semaphore from storage, compute a result based on the value of the semaphore, and conditionally store it back to the same location. The conditional store is performed based upon the existence of a reservation established by the preceding **lwarx** instruction. If the reservation exists when the store is executed, the store is performed and a bit is set in the condition register.

If the reservation does not exist when the store is executed, the target storage location is not modified and a bit in the condition register is cleared. The **lwarx** and **stwcx.** primitives allow software to read a semaphore, compute a result based on the value of the semaphore, store the new value back into the semaphore location only if that location has not been modified since it was first read, and determine if the store was successful.

If the store was successful, the sequence of instructions from the read of the semaphore to the store that updated the semaphore appear to have been executed atomically (i.e., no other processor or mechanism modified the semaphore location between the read and the update), thus providing the equivalent of a real atomic operation. However, other processors may have read from the location during this operation.

The reservation set by an **lwarx** instruction can be cleared by the following conditions:

- The processor having the reservation executes a store conditional instruction to any address.
- Another device executes any store instruction to any address in the 32-byte sector associated with the reservation.
- The processor with the reservation takes any exception.
- The processor with the reservation executes an **sc** instruction.
- The processor with the reservation executes a trap instruction that takes a program exception.

3.5.7 In a uniprocessor system, a program that modifies instructions it intends to execute must execute an **isync** instruction to ensure that all modifications are made visible to the instruction queue. Note that additional instructions are required for other PowerPC processors.

3.5.10.1 Replace the first sentence of this section with the following:

The steps for converting a floating-point value from the double-precision register format to single-precision memory format are as follows:

3.6.1.1 Note that the LI field is appended with b'00' prior to the addition.

3.6.1.2 Note that the BD field is appended with b'00' prior to the addition.

3.6.1.3 Note that the LI field is appended with b'00' prior to the addition.

3.6.1.4 Note that the BD field is appended with b'00' prior to the addition.

3.6.3 The PowerPC architecture defines the **bcctr** instruction with the “decrement and test CTR” ($BO_2 = 0$) option as an invalid form, and attempting to execute such an instruction causes boundedly undefined results. However, the MPC601 tests the count register for 0 and branches based on the result. Instruction fetching is directed to the address specified in the non-decremented version of the count register.

3.7.1 The RTCU and RTCL registers can be read in user level and can be written to in supervisor level. The SPR encodings for reading the RTCU and RTCL registers are 4 and 5, respectively (regardless of whether the processor is in user- or supervisor level). The SPR encodings for writing RTCU and RTCL are 20 and 21, respectively.

3.8.4 The essence of the **tlbie** instruction may be broadcast onto the MPC601 bus interface. This function is enabled by setting HID1[17].

Section 4: Chapter 4, "Cache and Memory Unit Operation"

This section describes additional information and corrections to Chapter 4, “Cache and Memory Unit Operation.”

4.8 Delete the next to the last paragraph in this section.

4.8.8 Replace the second paragraph with the following:

The **dcbi** instruction cannot be used to invalidate instructions in the cache of the MPC601. This instruction may have the effect of unmodifying data storage depending upon timing, exceptions, and other events.

4.11 In row #12 on page 4-28, “Four-beat write (quadword 2)” should update the sector status. The Current State column correctly contains an x but the Next State Column specifies no change. The next state should be M.

Section 5: Chapter 5, "Exceptions"

This section describes additional information and corrections to Chapter 5, “Exceptions.”

Section Number	
5.1	The last bullet in the entry for the instruction access exception in Table 5-1 should read as follows: If the K bits in the segment register and the PP bits in the PTE or BAT are set to prohibit read access, instructions cannot be fetched from this location.
5.1.1.3	The second bulleted item in this section should read as follows: SRR0 addresses either the instruction that would have completed or some instruction following it that would have completed if the exception had not occurred.
5.4.4	The following information should replace the appropriate bit descriptions for SRR1 in Table 5-12. 3 Cleared. Note that the PowerPC architecture defines this as set if the fetch access was to an I/O controller interface segment (SR[T]=1). Note that this condition causes SRR1[0–15] to be cleared in the MPC601. 10 Cleared
5.4.5	In early versions of the MPC601 (processor revision level x'0000'), the external interrupt is a level-sensitive signal and should be held active until reset by the interrupt service routine. Phantom interrupts due to phenomena such as crosstalk and bus noise should be avoided. In later versions of the MPC601 (processor revision level x'0001' and higher), the MPC601 is guaranteed to detect an external interrupt when the INT signal is held active for at least two clock cycles. The MPC601 is guaranteed to ignore the INT signal if it is held for less than one clock cycle.
5.4.6	The first DSISR value listed in Table 5-14 should be as follows: 000000000000 00 0 01 0 0101 ttttt ????? The following should be added to Table 5-14:

DAR	Set to the EA of the data access as computed by the instruction causing the alignment exception.

5.4.6.1.2	Replace the third bulleted item with the following: The **lwarx**/**stwcx.**/**lscbx** instructions that map into an I/O controller interface segment always cause a data access exception. However, if the instruction crosses a segment boundary an alignment exception is taken instead.
5.4.10	Note also that unlike exceptions that occur with memory accesses, loads and both loads and stores with update to I/O controller interface segments cause the target register to be updated, regardless of whether an exception is taken.

5.4.12 Replace this section with the following:

Run Mode/Trace Exception (x'02000')

The MPC601 defines an implementation-specific exception called the run mode exception. This exception is taken by the MPC601 under the following circumstances:

- Instruction address compare
- Branch target address compare
- Trace mode (MSR[SE] is set)—When an instruction clears MSR[SE], trace mode ends immediately. Note that other PowerPC processors implement a separate trace exception at vector x'00D00'.

Note that this exception may not be implemented by other PowerPC processors, and that this exception can be enabled and disabled using bits 8 and 9 in HID1; the exception is enabled when HID1[8,9] = b'01'. When this exception occurs, the registers are set as indicated in Table 5-24.

Table 5-24. Run Mode Exception—Register Settings

Register	Setting			
SRR0	Set to the address of the instruction that causes the run mode exception			
SRR1	Loaded from bits 0–31 of the MSR			
MSR	EE	0	SE	0
	PR	0	FE1	0
	FP1	0	EP	Value is not altered
	ME	Value is not altered	IT	0
	FE0	0	DT	0

The run mode is determined by the settings of HID1[1–3]. These settings are defined in Table 5-25.

Table 5-25. Run Modes Setting

HID1(1–3) Setting	Run Mode
000	Normal run mode
001	Undefined. Do not use.
010	Limited instruction address compare.
011	Undefined. Do not use.
100	Single instruction step
101	Undefined. Do not use.
110	Full instruction address compare
111	Full branch target address compare

Table 5-26 describes the run modes.

Table 5-26. Run Modes Description

Mode	Description
Normal run mode	No address breakpoints are specified and the MPC601 processes zero to three instructions per cycle.
Single instruction step mode	In single instruction step mode, the fetcher processes one instruction at a time. After an instruction is processed and the chip quiesces, the appropriate break action is performed. Note that this mode is distinct from the trace exception, which depends on the setting of MSR[SE].
Limited instruction address compare mode	The MPC601 runs at full speed until the EA of the instruction in the lowest position in the instruction queue (IQ0) matches the one specified in HID2. At this point the appropriate break action is performed. This is a limited compare in that branches and floating-point operations and the addresses associated with them may never be detected.
Full instruction address compare mode	In full instruction address compare mode, processing proceeds out of IQ0. When the EA in HID2 matches the EA of the instruction in IQ0, the appropriate break action is performed. Unlike the limited instruction address compare mode, all instructions pass through the IQ0 in this mode. That is, instructions cannot be folded out of the instruction stream.
Full branch target address compare mode	This mode is similar to full instruction address compare mode except that the branch target is compared against HID2. When addresses match, the appropriate break action is taken. This allows the programmer to see how a program got to an address. This mode can be used with **b**, **bc**, **bcr**, and **bcc** instructions.

When the trace exception is enabled, (MSR[SE] is set), a trace interrupt is taken after each instruction that completes without causing an exception or context change (such as an **sc**, **rfi**, or a load instruction that causes an exception). MSR[SE] is cleared when the trace exception is taken. In the normal use of this function, MSR[SE] is restored when the exception handler returns to the interrupted program using an **rfi** instruction.

Register settings for the trace mode are described in Table 5-27.

Table 5-27. Trace Exception—Register Settings

Register	Setting
SRR0	Set to the address of the next instruction to be executed in the program for which the trace exception was generated
SRR1	0–15 Cleared 16–31 Loaded from bits 16–31 of the MSR
MSR	EE 0 SE 0 PR 0 FE1 0 FP1 0 EP Value is not altered ME Value is not altered IT 0 FE0 0 DT 0

When a run mode or trace exception is taken, instruction execution resumes as offset x'02000' from the base address indicated by MSR[EP].

Section 6: Chapter 6, “Memory Management Unit”

This section describes additional information and corrections to Chapter 6, “Memory Management Unit.”

6.1.8 The first two bullet items under constraints enforced for instruction prefetching should be deleted.

6.7.6 The **dcbt**/**dcbtst** instruction branch of Figure 6-10 should say “Abort Access” instead of “Abort Translation.”

6.9.1.5.2 The Hash Value 2 shown in Figure 6-22 shows an extra 4 bits (1111) that should be deleted. The Hash Value 2 should be replaced with the following: “101 1000 0000 0001 1001.”

6.9.2 Steps 1 and 5 of the page table search operation imply that PTEs are read into the processor as single-beat read operations. In reality, the MPC601 performs a burst read operation at the PTE address (to load a sector of the on-chip cache) and optionally performs a second burst read operation to fill the cache line. However, the PTEs are read from the cache and compared with the virtual address information one at a time.

6.9.2 At the top of Figure 6-23, the fetch of a PTE is described as a single-beat read from PA. This should be replaced with a box showing that four PTEs are burst in at once, and four more PTEs may be burst in to fill a cache line.

Section 7: Chapter 7, "Instruction Timing”

This section describes additional information and corrections to Chapter 7, “Instruction Timing.”

7.1 Replace Figure 7-1 with the following:

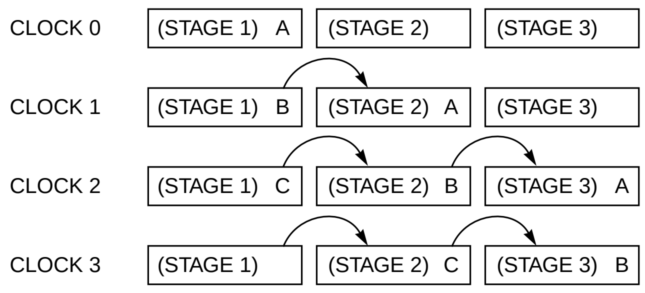


Figure 7-1. Pipelined Execution Unit

7.3.3	Replace the last two paragraphs in this section with the following:

Double-precision floating-point multiply instructions spend multiple clock cycles in the decode and execute stages of the FPU. However, the **fmul** instruction is broken down into two parts (which in the FPU pipeline appear to be two instructions). This allows the instruction to occupy two stages in the FPU simultaneously. The first part of the instruction can begin FPU execute 1 stage as the second part enters the decode stage. Likewise, when the first part of the instruction enters FPU execute 2 stage, the second part enters execute 1 stage.

This self-pipelining reduces the latency to five cycles and improves the throughput. For example, a series of **fmul** instructions would have a throughput of one instruction every two cycles.

Section 8: Chapter 8, "Signal Descriptions"

This section describes additional information and corrections to Chapter 8, “Signal Descriptions.”

Section Number

8.1 Replace Figure 8-1 with Figure 1-7 (shown on page Addendum-2 of this addendum).

8.2.4.1.2 Replace the first two entries in Table 8–1 with the following:

TT0	Special operations: This signal is asserted whenever a bus transaction is run in response to a **lwarx**/**stwcx.** instruction pair, a TLBI (translation lookaside buffer invalidate) operation, or either an **eciwx** or **ecowx** instruction.
TT1	Read (or write) operations: This signal indicates whether the transaction is a read (TT1 high) or a write (TT1 low). This assumes that the transaction is not address-only.

8.2.4.2.1 Replace the last paragraph of the State Meaning section with the following:

For external control instructions (**eciwx** and **ecowx**), TSIZ0–TSIZ2 are used to output bits 29–31 of the external access register (EAR), which are used to form the resource ID (TBST||TSIZ0–TSIZ2).

8.2.4.3.1 Replace the first paragraph of the State Meaning section with the following:

Asserted—Indicates that a burst transfer is in progress.

8.2.4.4 Replace the first sentence with the following:

The transfer code (TC0–TC1) consists of two output signals on the MPC601.

Replace the first entry in Table 8-4 with the following:

TC0	Assertion depends on whether the current transaction is a read or write operation; therefore, TC0 should be used with TT1. On a read operation, TC0 asserted indicates the transaction is an instruction fetch operation; otherwise, the read operation is a data operation. Asserting TC0 for write operations indicates the cache sector associated with a write is being invalidated; TC0 negated indicates the cache sector associated with a write is *not* being invalidated.

8.2.4.6 Substitute the following for the State Meaning entry:

Asserted—Indicates that a single-beat transaction is write-through, reflecting the value of the W bit for the block or page that contains the address of the current transaction. For burst writes, this indicates that the write is the result of a **dcbf** or **dcbst** instruction.

Negated—Indicates that a transaction is not write-through. For bursts it is negated for cast-outs and snoop pushes.

8.2.4.9 Add the following sentence to the first paragraph:

This pin must be enabled by setting HID0[31] if it is to be used.

The Timing Comments should read as follows: "Assertion/Negation—Must be valid throughout the entire address tenure."

8.2.6.1 Substitute the following for the Asserted information in the State Meaning section:

Asserted—Indicates that the MPC601 may, with the proper qualification, assume mastership of the data bus. The MPC601 derives a qualified data bus grant when $\overline{\text{DBG}}$ is asserted and $\overline{\text{DBB}}$, $\overline{\text{DRTRY}}$, and $\overline{\text{ARTRY}}$ are negated; that is, the data bus is not busy ($\overline{\text{DBB}}$ is negated), there is no outstanding attempt to retry the current data tenure ($\overline{\text{DRTRY}}$ is negated), and there is no outstanding attempt to perform an $\overline{\text{ARTRY}}$ of the associated address tenure.

8.2.7.2.2 Substitute the following for the next-to-last sentence in the State Meaning section:

Detected even parity causes a checkstop if data parity errors are enabled in the HID register.

8.2.9.1 Replace the First paragraph of the State Meaning section with the following:

Asserted—The MPC601 latches the interrupt condition if the MSR(EE) bit is set and ignores the interrupt condition otherwise. To guarantee that the MPC601 will take the external interrupt, the $\overline{\text{INT}}$ pin must be held active until the MPC601 takes the interrupt; otherwise, the MPC601 may or may not take an external interrupt, depending on whether MSR[EE] bit was set while the $\overline{\text{INT}}$ signal was held active.

8.2.9.6 Add the following sentence to the first paragraph:

Note that systems that do not use this signal should tie it low.

8.2.11 This section should be deleted.

8.2.12.3 Replace Figure 8-6 with the following:

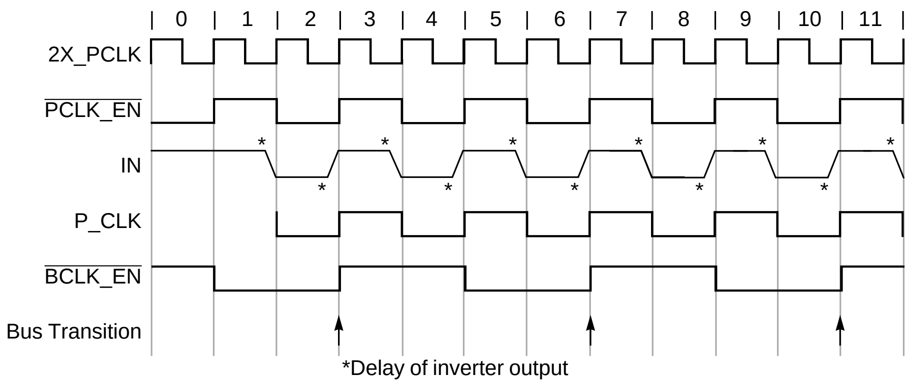


Figure 8-6. Generation of Bus Transactions—Logical Bus Clock = 1/2 P_CLK

Section 9: Chapter 9, "System Interface Operation"

9.1.2 Replace the last sentence in the second bulleted item with the following:

The update of the other sector can be disabled by setting bits in the HID0 register. HID0[DRF], bit 26, can be used to disable fetches and HID0[DRL], bit 27, can be used to disable loads and stores.

9.2 Replace the second paragraph with the following:

Figure 9-3 shows that the address and data tenures are distinct from one another and that both consist of three phases—arbitration, transfer, and termination. Address and data tenures are independent (indicated in Figure 9-3 by fact that the data tenure begins before the address tenure ends), which allows split-bus transactions to be implemented at the system level in multiprocessor systems. Figure 9-3 shows a data transfer that consists of a single-beat transfer of as many as 64 bits. Four-beat burst transfers of 32-byte cache sectors require data transfer termination signals for each beat of data.

9.3.2.1 Delete the second paragraph.

9.3.2.3 Substitute the following row in Table 9-3.

First transfer: two bytes	010	110	—	—	—	—	—	—	A	A

9.3.2.4 Replace the last sentence of the second paragraph with the following:

TC0 negated indicates the write is not invalidating any cache sector (for example, write-through or cache-inhibited write operations.)

9.3.3 The second sentence of the second paragraph should read as follows:

After $\overline{\text{ARTRY}}$ and $\overline{\text{SHD}}$ are asserted, they will be three-stated for two bus cycles and the system is responsible for precharging both $\overline{\text{ARTRY}}$ and $\overline{\text{SHD}}$ signals.

9.3.3 Add the following sentence to the end of the fourth bulleted item:

The override mode uses the $\overline{\text{HP\_SNP\_REQ}}$ signal to determine if the snoop queue is to be used. This mode is enabled by setting HID0[31].

9.4.2 Replace the first sentence of the third paragraph with the following:

The type of transaction initiated by the MPC601 depends on whether the code or data is cacheable and, for store operations, whether the cache is operated in write-back or write-through mode which software controls at either the page or block basis.

9.4.3 Replace the last sentence with the following:

$\overline{\text{ARTRY}}$ can also terminate a data bus transaction. For burst transactions, this $\overline{\text{ARTRY}}$ must occur no later than the cycle of the second $\overline{\text{TA}}$. For single-beat transactions, it must occur no later than the cycle following $\overline{\text{TA}}$. In either case, the $\overline{\text{ARTRY}}$ must be for the address bus tenure associated with the data bus tenure.

9.4.3.1 Replace Figure 9-10 with the following:

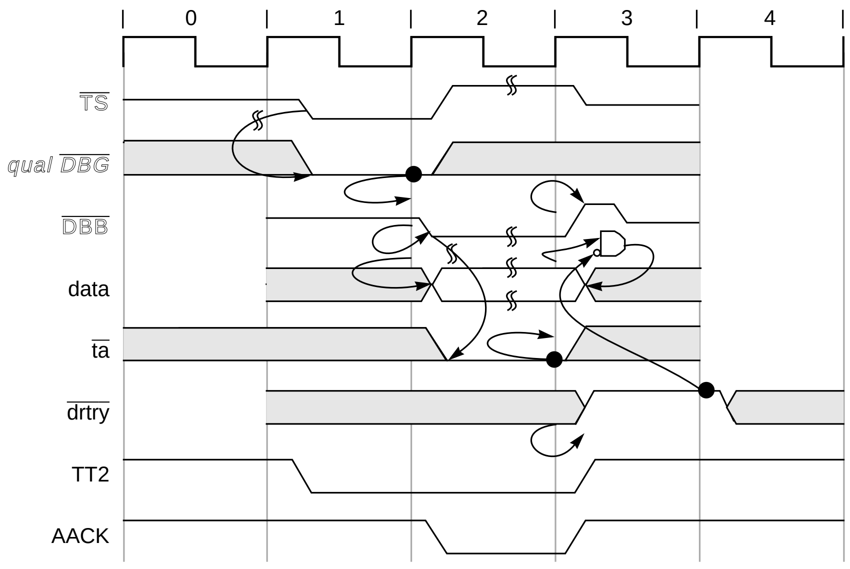


Figure 9-10. Normal Single-Beat Read Termination

Replace Figure 9-11 with the following:

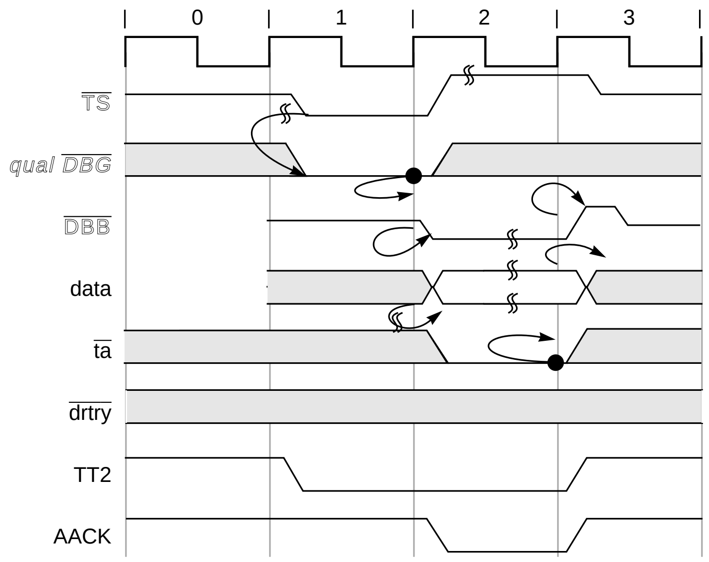


Figure 9-11. Normal Single-Beat Write Termination

9.5 The clock signals at the bottom of the figures in this section should be ignored.

Replace the first two paragraphs with the following:

This section shows timing diagrams for various scenarios. Figure 9-16 illustrates the fastest single-beat reads. This figure shows both minimal latency and maximum single-beat throughput. By delaying the data bus tenure, the latency increases, but, because of split-transaction pipelining, the overall throughput is not affected unless the data bus latency causes the third address tenure to be delayed.

Note that all bidirectional signals go to high-impedance between bus tenures.

9.6.4 Replace the last sentence in the fourth paragraph with the following

The MPC601 involved in this transaction, however, does not initiate any other I/O controller load or store operations once the first I/O controller interface operation has begun address tenure; however, if the I/O operation is retried, other higher-priority operations can occur.

9.6.4 Replace the last sentence of the last paragraph with the following:

If the $\overline{\text{TEA}}$ signal is not asserted with each tenure of a given I/O controller interface operation, the result of the assertion of $\overline{\text{TEA}}$ is unpredictable. The MPC601 may take a machine check exception or cause a checkstop condition.

9.10 Add the following note after step 5:

Note that steps 4 and 5 can occur in either order.

Section 10: Chapter 10, “Instruction Set”

This section describes additional information and corrections to Chapter 10, “Instruction Set.”

lmw Add the following information:

In future implementations, this instruction is likely to have greater latency and take longer to execute, perhaps much longer, than a sequence of individual load instructions that produce the same results.

lfsu The reference in the description of this instruction should be to Section 3.5.9.1, “Double-Precision Conversion for Floating-Point Load Instructions.”

lfsux The reference in the description of this instruction should be to Section 3.5.9.1, “Double-Precision Conversion for Floating-Point Load Instructions.”

lfsx The reference in the description of this instruction should be to Section 3.5.9.1, “Double-Precision Conversion for Floating-Point Load Instructions.”

lswi Add the following information:

In future implementations, this instruction is likely to have greater latency and take longer to execute, perhaps much longer, than a sequence of individual load instructions that produce the same results.

mffs This instruction is executed in the FPU rather than the IU.

lswx Add the following information:

Under certain conditions (for example, segment boundary crossings) the alignment error handler may be invoked. For additional information about alignment exceptions, see Section 5.4.6, “Alignment Exception (x’00600’)

In future implementations, this instruction is likely to have greater latency and take longer to execute, perhaps much longer, than a sequence of individual load instructions that produce the same results.

mfspr Replace Table 10-4 with the following:

Table 10-4. SPR Encodings for mfspr

SPR[1]			Register Name	Access
Decimal	SPR[5–9]	SPR[0–4]		
0	00000	00000	MQ	User
1	00000	00001	XER	User
4	00000	00100	RTCU[2]	User
5	00000	00101	RTCL[2]	User
6	00000	00110	DEC[3]	User
8	00000	01000	LR	User
9	00000	01001	CTR	User

Table 10-4. SPR Encodings for mfspr (Continued)

SPR[1]			Register Name	Access
Decimal	**SPR[5–9]**	**SPR[0–4]**		
18	00000	10010	DSISR	Supervisor
19	00000	10011	DAR	Supervisor
22	00000	10110	DEC[3]	Supervisor
25	00000	11001	SDR1	Supervisor
26	00000	11010	SRR0	Supervisor
27	00000	11011	SRR1	Supervisor
272	01000	10000	SPRG0	Supervisor
273	01000	10001	SPRG1	Supervisor
274	01000	10010	SPRG2	Supervisor
275	01000	10011	SPRG3	Supervisor
282	01000	11010	EAR	Supervisor
287	01000	11111	PVR	Supervisor
528	10000	10000	BAT0U	Supervisor
529	10000	10001	BAT0L	Supervisor
530	10000	10010	BAT1U	Supervisor
531	10000	10011	BAT1L	Supervisor
532	10000	10100	BAT2U	Supervisor
533	10000	10101	BAT2L	Supervisor
534	10000	10110	BAT3U	Supervisor
535	10000	10111	BAT3L	Supervisor
1008	11111	10000	Checkstop Register (HID0)	Supervisor
1009	11111	10001	Debug Mode Register (HID1)	Supervisor
1010	11111	10010	IABR (HID2)	Supervisor
1013	11111	10101	DABR (HID5)	Supervisor
1023	11111	11111	PIR (HID15)	Supervisor

[1]Note that the order of the two 5-bit halves of the SPR number is reversed compared with actual instruction coding.

If the SPR field contains any value other than one of these implementation-specific values or one of the values shown in Table 3-40, the instruction form is invalid.

SPR[0]=1 if and only if the register is being accessed at the supervisor level. Execution of this instruction specifying a defined and supervisor-level register when MSR[PR]=1 results in a privilege violation type program exception.

For **mtspr** and **mfspr** instructions, the SPR number coded in assembly language does not appear directly as a 10-bit binary number in the instruction. The number coded is split into two 5-bit halves that are reversed in the instruction, with the high-order 5 bits appearing in bits 16–20 of the instruction and the low-order 5 bits in bits 11–15.

SPR encodings for DEC, MQ, RTCL, and RTCU are not part of the PowerPC architecture.

[2]On the MPC601, the **mfspr** instruction for the RTCU and RTCL registers must use these encodings (SPR4 and SPR5, respectively) regardless of whether the processor is in supervisor or user mode. The **mtspr** instruction, which is supervisor-only for the RTCU and RTCL registers, must use the SPR20 and SPR21 encodings, respectively.

[3]Read access to the DEC register is supervisor-only in the PowerPC architecture, using SPR22. However, the POWER architecture allows user-level read access using SPR6. Note that the SPR6 encoding for the DEC will not be supported by other PowerPC processors.

mtspr Replace Table 10-5 with the following:

Table 10-5. SPR Encodings for mtspr

SPR[1]			Register Name	Access
Decimal	**SPR[5–9]**	**SPR[0–4]**		
0	00000	00000	MQ	User
1	00000	00001	XER	User
8	00000	01000	LR	User
9	00000	01001	CTR	User
18	00000	10010	DSISR	Supervisor
19	00000	10011	DAR	Supervisor
20	00000	10100	RTCU[2]	Supervisor
21	00000	10101	RTCL[2]	Supervisor
22	00000	10110	DEC[3]	Supervisor
25	00000	11001	SDR1	Supervisor
26	00000	11010	SRR0	Supervisor
27	00000	11011	SRR1	Supervisor
272	01000	10000	SPRG0	Supervisor
273	01000	10001	SPRG1	Supervisor
274	01000	10010	SPRG2	Supervisor
275	01000	10011	SPRG3	Supervisor
282	01000	11010	EAR	Supervisor
528	10000	10000	BAT0U	Supervisor
529	10000	10001	BAT0L	Supervisor

Table 10-5. SPR Encodings for mtspr (Continued)

SPR[1]			Register Name	Access
Decimal	SPR[5–9]	SPR[0–4]		
530	10000	10010	BAT1U	Supervisor
531	10000	10011	BAT1L	Supervisor
532	10000	10100	BAT2U	Supervisor
533	10000	10101	BAT2L	Supervisor
534	10000	10110	BAT3U	Supervisor
535	10000	10111	BAT3L	Supervisor
1008	11111	10000	Checkstop Register (HID0)	Supervisor
1009	11111	10001	Debug Mode Register (HID1)	Supervisor
1010	11111	10010	IABR (HID2)	Supervisor
1013	11111	10101	DABR (HID5)	Supervisor
1023	11111	11111	PIR (HID15)	Supervisor

[1]Note that the order of the two 5-bit halves of the SPR number is reversed compared with actual instruction coding.

If the SPR field contains any value other than one of these implementation-specific values or one of the values shown in Table 3-40, the instruction form is invalid.

SPR[0]=1 if and only if the register is being accessed at the supervisor level. Execution of this instruction specifying a defined and supervisor-level register when MSR[PR]=1 results in a privilege violation type program exception.

For **mtspr** and **mfspr** instructions, the SPR number coded in assembly language does not appear directly as a 10-bit binary number in the instruction. The number coded is split into two 5-bit halves that are reversed in the instruction, with the high-order 5 bits appearing in bits 16–20 of the instruction and the low-order 5 bits in bits 11–15.

SPR encodings for DEC, MQ, RTCL, and RTCU are not part of the PowerPC architecture.

[2]On the MPC601, the **mfspr** instruction for the RTCU and RTCL registers must use these encodings (SPR4 and SPR5, respectively) regardless of whether the processor is in supervisor or user mode. The **mtspr** instruction, which is supervisor-only for the RTCU and RTCL registers, must use the SPR20 and SPR21 encodings, respectively.

[3]Read access to the DEC register is supervisor-only in the PowerPC architecture, using SPR22. However, the POWER architecture allows user-level read access using SPR6. Note that the SPR6 encoding for the DEC will not be supported by other PowerPC processors.

stmw	Add the following information: In future implementations, this instruction is likely to have greater latency and take longer to execute, perhaps much longer, than a sequence of individual store instructions that produce the same results.
stswi	Add the following information: In future implementations, this instruction is likely to have greater latency and take longer to execute, perhaps much longer, than a sequence of individual store instructions that produce the same results.
stswx	Add the following information: In future implementations, this instruction is likely to have greater latency and take longer to execute, perhaps much longer, than a sequence of individual store instructions that produce the same results.
10.3	The **tlbiex** instruction has been removed from the PowerPC architecture. and should be deleted from Table 10-6.

Section 11: Appendixes

This section describes additional information and corrections to the appendixes.

Section Number	
App. A	The **slbiex** and **tlbiex** instructions, which are not implemented in the MPC601, have been removed from the PowerPC architecture.
App. C	The **slbiex** and **tlbiex** instructions, which are not implemented in the MPC601, have been removed from the PowerPC architecture.
App. D	The MPC601 takes an illegal instruction error exception for instructions that the PowerPC architecture defines as reserved, except for those POWER instructions that are implemented on the MPC601.
F.3.2	Replace **Round Integer(frac,gbit,rbit,xbit,round_mode)** with the following:

Round Integer(frac,gbit,rbit,xbit,round_mode)

In this example, u represents an undefined hexadecimal digit. Comparisons ignore the u bits.

```
inc ← 0
If round_mode= b’00’ then
  Do
    If sign || frac[64] || gbit || rbit || xbit = b’u11uu’ then inc ← 1
    If sign || frac[64] || gbit || rbit || xbit = b’u011u’ then inc ← 1
    If sign || frac[64] || gbit || rbit || xbit = b’u01u1’ then inc ← 1
  End
If round_mode= b’10’ then
  Do
      If sign || frac[64] || gbit || rbit || xbit = b’0u1uu’ then inc←1
      If sign || frac[64] || gbit || rbit || xbit = b’0uu1u’ then inc ← 1
      If sign || frac[64] || gbit || rbit || xbit = b’0uuu1’ then inc ← 1
    End
```

```
If round_mode= b’11’ then
  Do
    If sign || frac[64] || gbit || rbit || xbit = b’1u1uu’ then inc ← 1
    If sign || frac[64] || gbit || rbit || xbit = b’1uu1u’ then inc ← 1
    If sign || frac[64] || gbit || rbit || xbit = b’1uuu1’ then inc← 1
  End
frac[0–64] ← frac[0–64] + inc
FPSCR[FR] ← inc
FPSCR[FI] ← gbit | rbit | xbit
Return
```

App. H The following appendix should be added to the user’s manual.

Appendix H
MPC601 as a PowerPC Microprocessor

The MPC601 processor is the first implementation of the PowerPC architecture. It offers a reliable platform for software and hardware developers to make products compatible with subsequent processors in the PowerPC family. In addition, the MPC601 provides extensions to the PowerPC architecture that allow it to function as a bridge from the POWER architecture. This appendix describes the POWER extensions as well as other differences between the MPC601 and the PowerPC architecture (*PowerPC Architecture, First Edition*). These differences can be categorized as follows:

- POWER extensions—Additional functionality not defined in the PowerPC architecture. For example, the MPC601 implements many POWER instructions that do not have PowerPC equivalents.
- Variances—MPC601 functionality that is implemented differently than as described in the PowerPC architecture. For example, there are several differences between the MPC601 MMU implementation and that specified by the PowerPC architecture. In general, these variances are not visible from the user level.
- Implementation-dependent extensions—These include features that are not part of but are allowed by the PowerPC architecture. For example, the MPC601 provides a set of implementation-dependent registers (HIDs) to control hardware features such as parity checking and instruction address breakpoint that are beyond the specifications in architecture. Software should take appropriate precautions to control use of these features.
- PowerPC optional features—These include optional features defined in the PowerPC architecture that are implemented in the MPC601.

This appendix does not describe performance trade-offs allowed by the PowerPC architecture. For example, some implementations may provide more support for alignment than others and therefore they may require different amounts of assistance in the interrupt handler.

This appendix also does not describe variances built into the PowerPC architecture to provide some latitude in PowerPC implementations for handling reserved, invalid, and undefined conditions. These aspects are left intentionally undefined. Note that while the MPC601's treatment of such aspects may be predictable, taking advantage of that behavior may cause software incompatibilities with other PowerPC implementations.

Where applicable, a reference is given to the portion of the user's manual that describes that functionality.

Also, the tables in this appendix indicate the level of architecture at which the MPC601 diverges. These levels are as follows:

- PowerPC user instruction set architecture—Defines the base user-level instruction set, user-level registers, data types, floating-point exception model, memory models for a uniprocessor environment, and programming model for uniprocessor environment.

  The tables in this appendix identify differences with this part of the architecture by listing "User" in the "Level" column of the tables.
- PowerPC virtual environment architecture—This describes the memory model for a multiprocessor environment, defines cache control instructions, and describes other aspects of virtual environments. Implementations that conform to the PowerPC virtual environment architecture also adhere to the PowerPC user instruction set architecture, but may not necessarily adhere to the PowerPC operating environment architecture.

  The tables in this appendix identify differences with this part of the architecture by listing "Virtual environment" in the "Level" column of the tables.
- PowerPC operating environment architecture—This defines the memory management model, supervisor-level registers, synchronization requirements, and the exception model. Implementations that conform to the PowerPC operating environment architecture also adhere to the PowerPC user instruction set architecture and the PowerPC virtual environment architecture definition.

  The tables in this appendix identify differences with this part of the architecture by listing "Operating environment" in the "Level" column of the tables.

H.1 POWER Extensions

Table H-1 lists POWER functionality supported by the MPC601 that is not defined in the PowerPC architecture. POWER extensions include additional functionality not defined in the PowerPC architecture.

Table H-1. POWER Extensions

Difference	Reference	Level	Type
The MQ register, provided for POWER compatibility, is not part of the PowerPC architecture.	Section 2.2.5.1, "MQ Register (MQ)"	User	POWER
The MPC601 implements the real-time clock feature, including POWER registers RTCU and RTCL, to provide a time reference rather than the time base feature defined by the PowerPC architecture.	Section 2.2.5.3, "Real-Time Clock (RTC) Registers"	Virtual and operating environment	POWER and Variance
In the MPC601 processor, the decrementer implementation uses the separate 7.8125 MHz RTC for its base frequency. Other PowerPC processors base the decrementer on the processor clock.	Section 2.3.3.4, "Decrementer (DEC) Register"	User and virtual environment	POWER and Variance
The decrementer register (DEC) in the MPC601 allows user-level read access, which is not provided in the PowerPC architecture.	Section 2.3.3.4, "Decrementer (DEC) Register"	User and operating environment	POWER
Because MPC601 supports the POWER registers, MQ, RTCU, and RTCL, instruction encodings to access them, **mtspr** and **mfspr**, are also provided.	Section 3.7.1, "Move to/from Special Purpose Register Instructions"	User and operating environment	POWER
The MPC601 provides a group of instructions for compatibility with POWER. The relationship between the MPC601 and the PowerPC instruction sets are shown in Appendix C. The PowerPC architecture defines these instructions as reserved.	Appendix C, "PowerPC Instructions Not Implemented in MPC601"	—	POWER

H.2 Variances

Variances include PowerPC functionality that is implemented in the MPC601 with some differences. In general, these variances are not visible from the user level. Table H-2 lists variances to the PowerPC architecture.

Table H-2. Variances

Difference	Reference	Level	Type
The VXSOFT, VXSQRT, and NI bits (bits 21, 22, and 29, respectively) are not implemented in the MPC601 processor.	Section 2.2.3, "Floating-Point Status and Control Register (FPSCR)”	User and operating environment	Variance
If the Floating-Point Convert to Integer Word (**fctiw**) instruction results in a conversion exception, FPSCR[XCVI] is set causing FPSCR[VX] to be set. The PowerPC architecture specifies that when **fctiw** causes FPSCR[XCVI] to be set, FPSCR[XX] is not altered. The MPC601 may set both FPSCR[XX] and FPSCR[XCVI] in some circumstances.	Section 3.4.3, "Floating-Point Rounding and Conversion Instructions”	User and operating environment	Variance
The architecture requires that both FPSCR[VXCVI] and FPSCR[VXSNAN] be set when the source operand of a **fctiw** is an SNAN. MPC601 sets only FPSCR[VXCVI].	Section 3.4.3, "Floating-Point Rounding and Conversion Instructions”	User and operating environment	Variance
PowerPC architecture defines the following bits in the machine state register (MSR) not implemented in the MPC601: **Bit Description** -- 13 Power management enable (POW) 15 Interrupt little-endian mode (ILE) 22 Branch trace enable (BE) 30 Recoverable exception (RE) 31 Little-endian mode (LE)	Section 2.3.1, "Machine State Register (MSR)”	Operating environment	Variance
The MPC601 provides a bit in an implementation-specific register (HID0) for selecting between big- and little-endian modes. PowerPC architecture defines MSR[LE] for this purpose.	Section 2.4.3, "Byte and Bit Ordering”	Operating environment	Variance
The number, function, content, and format of the BAT registers implemented by the MPC601 is different than that specified by the PowerPC architecture.	Section 2.3.3.11, "BAT Registers”	Operating environment	Variance
The MPC601 clears the reservation bit set by the execution of an **lwarx** instruction when taking any type of exception. The PowerPC architecture defines that the reservation be cleared for a subset of exceptions.	Section 3.5.7, "Memory Synchronization Instructions”	User and operating environment	Variance
Because the MPC601 does not implement the MSR[RE] bit (recoverable exception bit), the operating system must use other criteria to determine if it is possible to recover from an asynchronous, imprecise interrupt.	Section 5.4.1, "Reset Exceptions (x’00100’)”	Operating environment	Variance
Instruction access exceptions due to instruction fetches from I/O controller interface segments do not set the SRR1[3] bit as defined in the PowerPC architecture. This condition clears SRR1[0–15].	Section 5: "Chapter 5, "Exceptions" of this addendum	Operating environment	Variance

Table H-2. Variances (Continued)

Difference	Reference	Level	Type
The non-IEEE mode bit, FPSCR[NI], is reserved in the MPC601. All floating-point results are consistent with IEEE standards.	Section 5.4.7.1, "Floating-Point Enabled Program Exceptions”	Operating environment	Variance
The MPC601 maps I/O controller interface error conditions to I/O controller interface exceptions instead of to the data access exception vector specified by the PowerPC architecture.	Section 5.4.10, "I/O Controller Interface Error Exception (x’00A00’)”	Operating environment	Variance
Unlike exceptions that occur with memory accesses, loads, loads with update, and stores with update to I/O controller interface segments cause any target registers to be updated, regardless of whether an exception is taken.	Section 5: "Chapter 5, "Exceptions" of this addendum	Operating environment	Variance
The MPC601 does not implement the trace exception as a separate exception as is defined in the PowerPC architecture (x’00D00’). The MPC601 vectors trace exceptions to the run-mode/trace exception (x’02000’).	Section 5.4.12, "Run Mode/Trace Exception (x’02000’)”	Operating environment	Variance
The MPC601 allows access to the I/O controller interface regardless of the setting of MSR[DT]. The PowerPC architecture does not allow these accesses when MSR[DT] is cleared.	Section 6.1.3, "Address Translation Mechanisms”	Operating environment	Variance
The MPC601 does not implement the PowerPC **tlbsync** instruction, but instead requires the use of a **sync** instruction to synchronize the completion of a broadcast **tlbie** instruction.	Section 10.3, "Instructions Not Implemented by the MPC601”	Operating environment	Variance
PowerPC architecture defines a "Guarded" memory attribute used to protect volatile memory. This attribute is associated with each virtual page (guarded bit in the page table entry) and with physical memory. The MPC601 provides a similar function using the "Caching Inhibited" memory attribute.	Section 6.1.8, "Effects of Instruction Prefetch on MMU”	Operating environment	Variance

H.3 Implementation-Dependent Extensions

Implementation-dependent extensions include features that are not part of, but are allowed by, the PowerPC architecture. Note that there are a number of such extensions that are described throughout the user’s manual, and there is no attempt to list them exhaustively in this appendix. Table H-3 provides a brief list of key implementation-dependent extensions supported by the MPC601.

Table H-3. Implementation-Dependent Extensions

Difference	Reference	Level	Type
The MPC601 includes the following implementation-specific registers: HID0, HID1, HID2, HID5, HID15. These may or may not be included in future implementations.	Section 2.3.3.12.1, "Checkstop Sources and Enables Register—HID0," through Section 2.3.3.12.5, "Processor Identification Register (PIR)—HID15"	Operating environment	Implementation -dependent extensions
Because the MPC601 automatically handles all floating-point data types, the MPC601 floating-point assist exception defined in the PowerPC architecture (x'00E00') would never be taken by the MPC601, and therefore it is not implemented.	—	Operating environment	Implementation -dependent extension
The MPC601 supports an additional exception called the run mode exception in addition to the exceptions defined by the PowerPC architecture.	Section 5.4.12, "Run Mode/Trace Exception (x'02000')"	Operating environment	Implementation -dependent extension
The MPC601 includes a feature that supports 256-Mbyte translation capability. This is enabled when the T-bit is set and the BUID field is = x'07F' in the appropriate segment register(s).	Section 6.5.2.1, "I/O Controller Interface Address Translation: T=1 in Segment Register"	Operating environment	Implementation -dependent extension

H.4 Options to the PowerPC Architecture

Table H-4 lists options to the PowerPC architecture supported by the MPC601. Note that because these are optional, they may not be supported by all PowerPC processors, just as the MPC601 does not support other optional features supported by the architecture.

Table H-4. Options to the PowerPC Architecture

Difference	Reference	Level	Type
The MPC601 processor implements the external access register (EAR), which is optional to the PowerPC architecture. Note that only four bits (28–31) are implemented in the MPC601, whereas the PowerPC architecture defines six bits.	Section 2.3.3.9, "External Access Register (EAR)"	Operating environment	Optional
The MPC601 implements the **eciwx** and **ecowx** instructions that are optional to the PowerPC architecture but are required for use with the EAR register.	Section 3.9, "External Control Instructions"	User	Optional

Literature Distribution Centers:
USA: Motorola Literature Distribution; P.O. Box 20912; Phoenix, Arizona 85036.
EUROPE: Motorola Ltd.; European Literature Centre; 88 Tanners Drive, Blakelands, Milton Keynes, MK14 5BP, England.
JAPAN: Nippon Motorola Ltd.; 4-32-1, Nishi-Gotanda, Shinagawa-ku, Tokyo 141 Japan.
ASIA-PACIFIC: Motorola Semiconductors H.K. Ltd.; Silicon Harbour Center, No. 2 Dai King Street, Tai Po Industrial Estate, Tai Po, N.T., Hong Kong.

MOTOROLA