POWER2 fixed-point, data cache, and storage control units

by D. J. Shippy
T. W. Griffith

The POWER2™ fixed-point, data cache, and storage control units provide a tightly integrated subunit for a second-generation high-performance superscalar RISC processor. These functional units provide dual fixed-point execution units and a large multiported data cache, as well as high-performance interfaces to memory, I/O, and the other execution units in the processor. These units provide the following features: dual fixed-point execution units, improved fixed-point/floating-point synchronization, new floating-point load and store quadword instructions, improved address translation, improved fixed-point multiply/divide, large multiported D-cache, increased bandwidth into and out of the caches through wider data buses, an improved external interrupt mechanism, and an improved I/O DMA mechanism to support multiple-streaming Micro Channels.®

Introduction

The POWER2™ processor is a next-generation RISC design which has significantly improved performance over that of previous designs with the addition of multiple floating-point and fixed-point functional units. To support the data and instruction demands of this processor, wider data buses, larger caches, and longer cache lines have been implemented. The POWER2 fixed-point unit (FXU), data cache unit (DCU), and storage control unit (SCU) provide functions and a system structure similar to those of the original RISC System/6000® (RS/6000) processor [1], but have improved in the following areas: an additional fixed-point execution unit, improved fixed-point/floating-point synchronization, new floating-point load and store quadword instructions, improved address translation, improved fixed-point multiply/divide, a multiported D-cache, larger caches and longer cache line size, increased bandwidth into and out of the caches through wider data buses, an improved external interrupt mechanism, and an improved I/O DMA mechanism to support multiple-streaming Micro Channels®.

This paper is organized as follows. First is a system overview, followed by a description of the dual FXU execution units. Next, POWER2 address translation is discussed, and the data cache control and directory unit are described. This is followed by a discussion of the DCU and the multiported data cache array macro. Finally, the SCU and the memory and I/O interfaces are discussed.

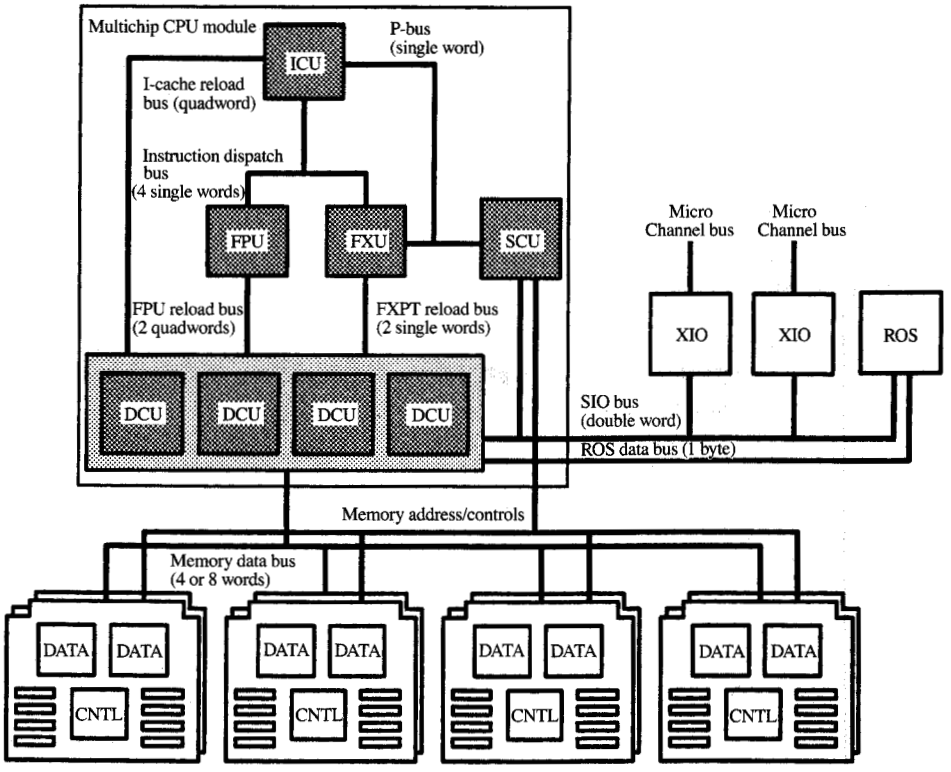

Figure 1

POWER2 system configuration with eight-word memory bus.

System overview

Figure 1 shows the POWER2 system. The FXU receives instructions from the instruction cache unit (ICU) through a four-word interface. The FXU is tightly coupled to the SCU by the processor bus (P-bus), which is used for cache-miss requests and I/O load/store operations. The SCU interfaces with I/O through the system I/O bus (SIO), and with memory through a split address/control and data bus. Both the FXU and SCU control the four DCU chips. To support the data demands of the multiple execution units as well as the data demands of multiple data-streaming Micro Channel devices, the POWER2 system provides large caches, long cache lines, and multiple wide data buses. The data cache consists of a four-way set-associative, dual-port 256KB cache with a 256-byte line. The instruction cache consists of a two-way set-associative 32KB cache with a 128-byte line. The memory bus interface to the DCU chips is either four or eight words wide. All cache and DMA operations use this bus. In addition, wide buses from the DCU to the ICU, FXU, and FPU have been implemented. There are two quadword interfaces to the FPU (quadrupling the data bandwidth over that of the original RS/6000 design [2]), two single-word interfaces to the FXU (double that of the original design), and a quadword interface to the ICU (double that of the original design). In addition, an I/O cache that provides DMA prefetch capability and a two-word system I/O bus are provided to the I/O control chips which generate the Micro Channel.

Fixed-point unit

The fixed-point unit (FXU) decodes and executes all instructions, except branches and floating-point arithmetic. Branches never leave the instruction cache unit (ICU), and floating-point arithmetic instructions are executed by the floating-point unit (FPU). Fixed- and floating-point

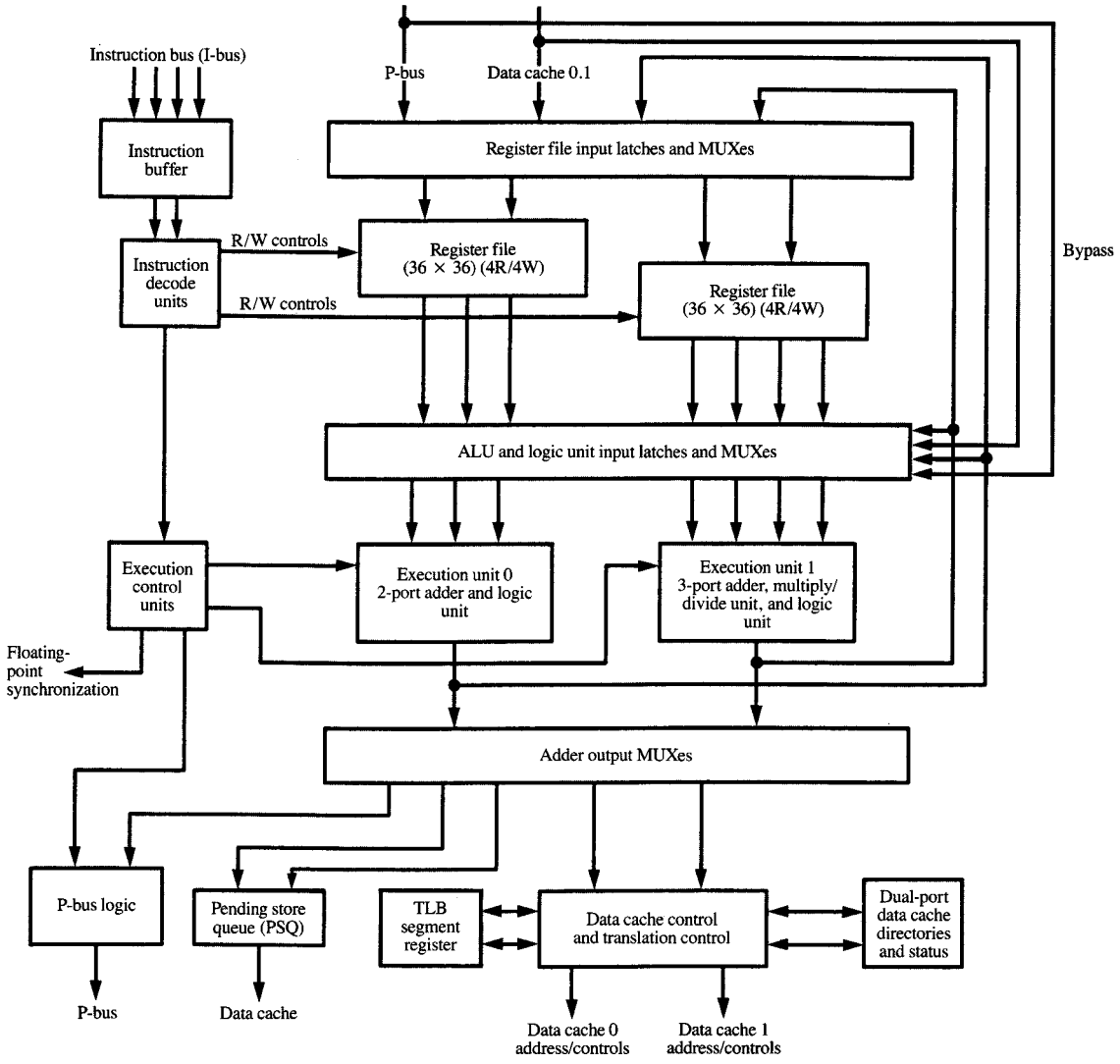

Figure 2

Fixed-point unit high-level block diagram.

instructions are dispatched by the ICU to the FXU and FPU simultaneously, and are executed synchronously in the FXU and FPU. In addition, the FXU contains the address translation, data protection, and data cache directories for both fixed- and floating-point load and store instructions.

Figure 2 shows a high-level block diagram of the FXU. The FXU receives four instructions from the ICU. The instruction buffer unit queues and dispatches instructions to two decoding units. Each decoding unit takes the primary and extended opcode fields and combines them into a single 10-bit field. This 10-bit field is used for decoding of all instructions for execution. At the end of the decode cycle, this combined opcode is latched for use during the execute cycle. The decoding unit also controls the general-purpose registers (GPRs). The architecture calls for thirty-two 32-bit GPRs. There are two sets of these registers, one for each execution unit. The hardware keeps these registers consistent with each other. The decoding unit decodes the instructions and manages their dispatch to the two execution units. The execution units are identical, except that only execution unit 1 may do multiply and divide. For load/store operations, the address translation logic converts virtual addresses to real

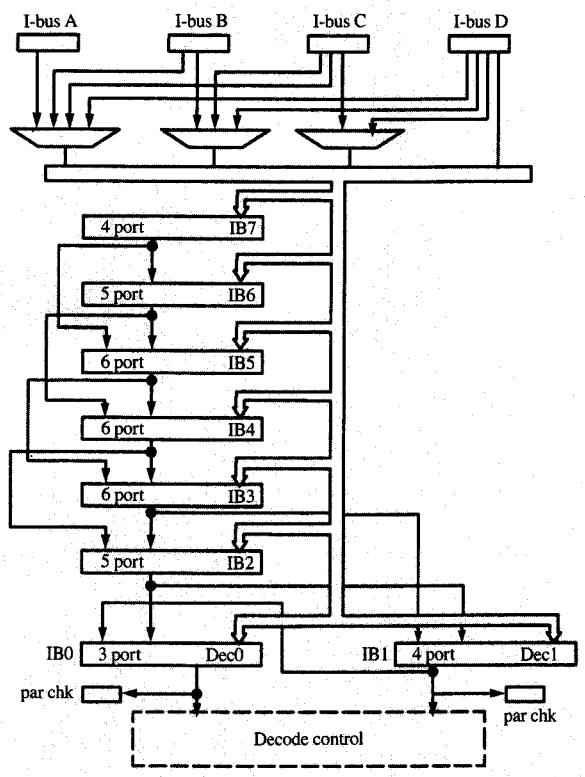

Figure 3

Fixed-point unit instruction buffer dataflow.

addresses, and the data cache control unit controls the data cache and directory. The P-bus logic interfaces with the other processor chips.

• *Instruction buffer*

Eight instruction buffers are used to queue instructions prior to decoding and execution, as shown in **Figure 3**. The I-cache dispatches instructions to the FXU and the FPU via the four-instruction (4 × 36 bits)-wide I-bus (I-bus A, B, C, and D). If these instructions are marked valid on the buses and the FXU has room to accept them, the FXU latches these instructions into the buffers in a FIFO manner. Associated with every instruction is a set of three tag bits that provide further information about the instruction.

Since both floating- and fixed-point instructions are dispatched on the I-bus, there are "holes" on the bus (from the perspective of the FXU) where there are valid floating-point instructions. To speed up the loading of instructions into the buffers, multiplexors are used in the dataflow to remove these holes. The instruction buffers are also designed to allow no holes to occur. This means that valid instructions in the pipeline may not be separated by invalid buffers. On each cycle, valid instructions are moved toward the bottom of the pipeline to occupy vacated buffers. This prevents invalid instructions from being fed to the execution unit.

The FXU tells the ICU how many buffers it has free, i.e., whether it has room to accept 0, 1, 2, 3, or 4 sequential instructions. It also tells the ICU whether it has room enough for four target instructions.

On each cycle the FXU moves valid instructions and their tags off the I-bus and into the buffers. The valid bit from the ICU is further qualified by the status of pending branches, canceled ICU instructions, and other related conditions to create the real valid bit for instruction. A valid bit is reset and the instruction canceled when the instruction is canceled by the ICU in the cycle after its dispatch, when it was conditionally dispatched and the branch is subsequently taken, or when an interrupt has occurred.

• *Instruction decoding*

There are two decoding units, which have the following responsibilities: decoding instructions; reading the general-purpose registers (GPRs); managing GPR bypass controls, sign extension and inverter controls, immediate field bypasses, and execute-bypass controls; and managing dispatch to the two execution units. The decoding units are identical. Each unit takes the primary and extended opcode fields and combines them into a single 10-bit field, which is then used for decoding of all instructions for execution. At the end of the decode cycle, this combined opcode is latched for use during the execute cycle.

During the decode cycle, three (or four for execution unit 1) locations are read from the GPR according to the address specified in the RS, RA, and RB fields of the instruction being decoded. This information is made available to the bypass multiplexors above the S, A, B, and U latches. If there are no holdoffs or bypasses, the data are latched in the S, A, B, and U latches for use during the execute cycle.

If the data required during the execute cycle are not in the GPR, a bypass of the GPR may be necessary. **Figure 4** shows the fixed-point GPR1 bypass dataflow. The three types of bypasses are the ALU bypass, the local load and P-bus bypass, and cache bypass. The ALU bypass occurs when an RR operation is dependent on the RR operation immediately preceding it. For example,

```
                          1     2     3     4
AND   R3,R1,R2      | DEC | EX  | WB  |
A     R4,R3,R5            | DEC | EX  | WB  |
```

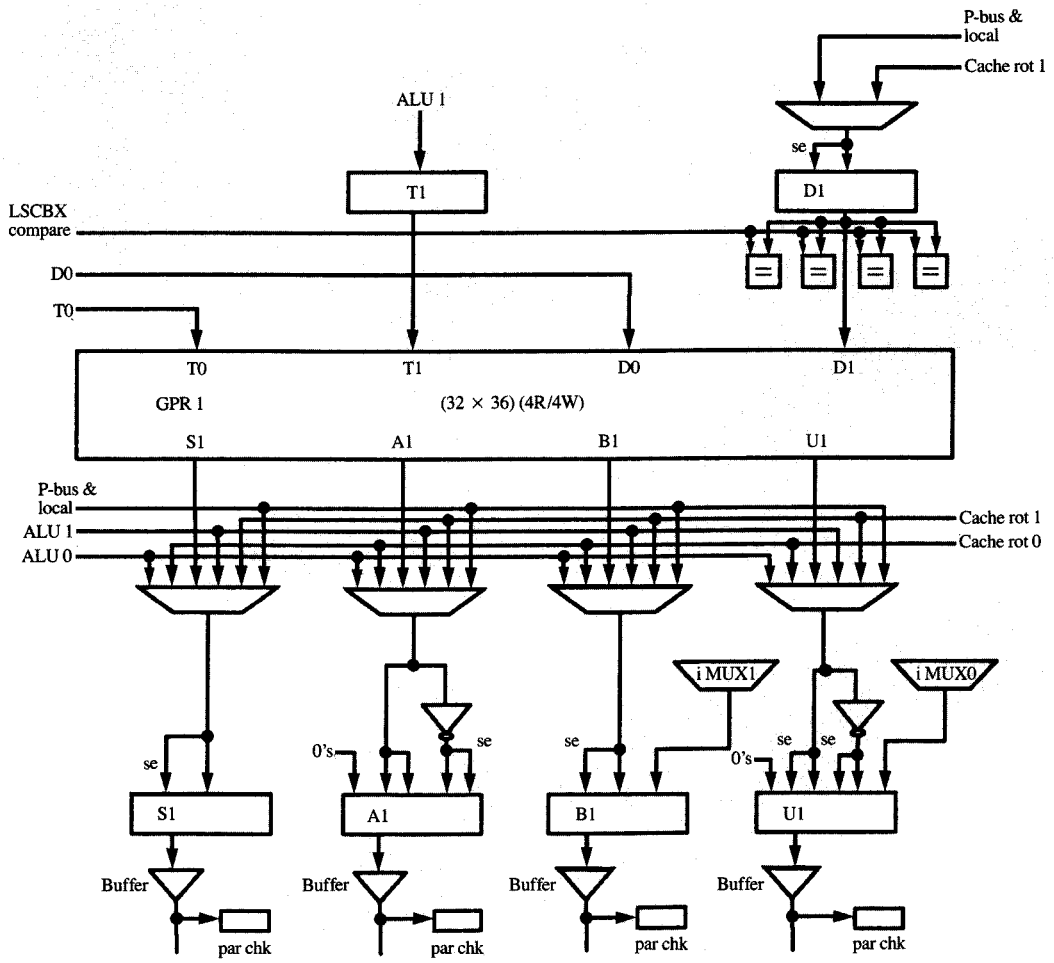

Figure 4

Fixed-point unit GPR1 bypass dataflow.

Because the data for the AND are not in the register file during cycle 2, the data must be bypassed during cycle 2 for the ADD. (There are no cycles lost in a bypass of this kind.) The local load/P-bus bypass occurs when a load/store to I/O or an MFSPR (Move From Special-Purpose Register)-type instruction is followed by a dependent operation:

```
                          1     2     3     4     5
MFSPR R3,LINK       | DEC | EX  | AC  | WB  |
A     R4,R3,R5            | DEC | EX  | EX  | WB  |
```

In this case, the data are bypassed from the P-bus or local load bus. (If a load or store to I/O caused the bypass, it may take several cycles for the data to be available.)

A cache bypass occurs when a load to memory space occurs:

```
                          1     2     3     4     5
LD    R3,R8,R9      | DEC | EX  | CA  | WB  |
A     R4,R3,R5            | DEC | EX  | EX  | WB  |
```

In this case, the data are bypassed from the cache data bus. (If there is a cache miss, it may be several cycles before data are ready.)

The decode units also manage the dispatching of instructions to the two execution units. In particular, register dependencies are resolved, as are ops such as string ops and load and store multiples, which are dispatched to both execution units.

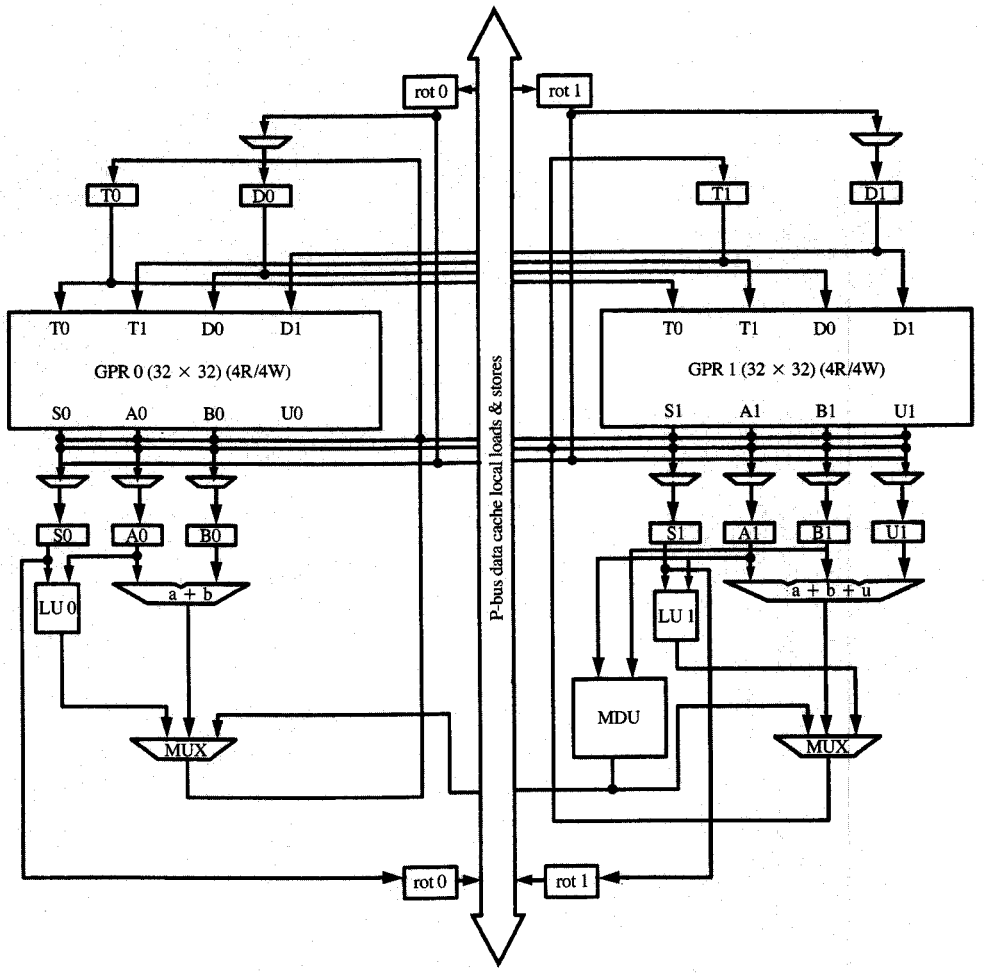

Figure 5

Fixed-point dual execution unit dataflow.

- *Execution control unit*

The execution control unit (sometimes referred to as the pipeline control unit), is responsible for load and store execution, holdoffs, and writeback of GPRs.

There are two execution units, which are identical except that only the B execution unit may do multiply and divide operations (see **Figure 5**). Also, some of the special operations such as cache operations and all privileged operations may execute only in the A execution unit.

The execution control unit must calculate an effective address based on the decoding of the instruction being executed, for example,

```
L      RT,RA,RB    | DEC | EX | CA | WB |
```

The effective address (E/A) in this case is RA + RB, and the data are stored in RT. The operands for the E/A are read during decoding and latched in the A and B latches. During execution, the A and B latches feed the adder, and the E/A is made available to the cache directory logic and the TLB lookup logic. If the load was an update form and no interrupts occurred, the E/A is also written into the GPR at the address specified by the RA field in the instruction. The data are latched in the T-latch at the end of execution, and are written in the GPR once it is determined that no interrupts will occur.

If the data were available in the cache (e.g., no TLB miss occurred and no data cache miss occurred), the data

			Bus_Data_Fetched_On
MemWordAddress	(E/A[0···29])	xxxa	0
	(E/A[0···29] +4)	bcde	1
	(E/A[0···29] +8)	fghi	0
	(E/A[0···29] +12)	jklm	1
	(E/A[0···29] +16)	nopx	0
	(E/A[0···29] +20)	xxxx	1

where a through p are the 16 bytes we are interested in and x is don't care data

```
LSI1    R28, RA, 12     * 16 bytes: loads R28, R29, R30, R31
Add2    RT, R28, RB     * dependent on a register loaded by LSI
Add3                    * nondependent
Add4                    * nondependent
```

where RA is some non-word-aligned address.

CYCLE	1	2	3	4	5	6	7	8
DEC	LS1/AD2	AD2/AD3	AD2/AD3	AD2/AD3	AD4	AD4		
EX		LS1/LS1	LSI/LSI	LS1/–	AD2/AD3	AD2/AD3	AD2/AD3	AD4
CA			xxxabcde	fghijklm	nopxxxxx			
HOLD					H H	H H		
FORMAT				abcdefgh	ijklmnop			
WRITE					abcdefgh	ijklmnop		

Notes
CA: normal cache access cycle.
FORMAT: formatting of data for write into the GPRs.
WRITE: data are written into the GPRs.

Figure 6

Construction of an unaligned load string from unaligned storage.

are available to be latched in the D-latch at the end of the cache access (CA) cycle. Data are written into the GPR during the write-back (WB) cycle. The address is latched during the decoding cycle from the RD field in the instruction. This address is held until used or canceled.

Multicycle loads are handled as a series of single-cycle loads using both execution units. The opcode is loaded into each execution unit, and the two load units are joined together to do two loads per cycle. Rotation of the words is handled with a doubleword rotator. **Figure 6** shows an example which does an unaligned load string from unaligned storage.

During the initial cycle of the multiword load, 4 is added to the E/A in execution unit 1. During all subsequent cycles, 8 is added to each E/A to obtain a new E/A. Thus, the two units work in unison fetching odd/even pairs until all fetching is done. As far as the cache is concerned, each storage unit is treated as if it were a single-cycle load by the TLB and D-cache directory logic, and thus works as described above in the load section. During the WB cycle, the address is incremented by 2 for use during the next WB cycle of the multicycle load. This loop continues until all registers requested have been loaded.

The LSCBX (load string compare byte) instruction works like a multicycle load as described above, with one enhancement. This instruction loads the requested data until a match occurs between a byte in the loaded data stream and a byte in the FXU XER register. Once a match is found, no more requests are made to memory.

LSCBX requests the number of bytes specified in the XER (bits 16–23) until a match occurs between data being loaded and XER bits 24–31. In this case, the XER is set up for a 12-byte (three-word) load, and the comparison is made for X'AA'. The match is found in the first word.

LSCBX cannot access memory locations past a lock bit boundary (lock bit boundaries are 128 bytes long), until all compares up to the boundary have been completed. Thus, LSCBX does not access memory locations past a compare byte. This prevents unnecessary data storage interrupts.

Note that the comparison is actually made during the WB cycle. This is implemented with 8-byte comparators between the D-latches and the GPR (see Figure 4). This creates pipelining effects which cause some bytes past the compare byte to be undefined in the register file.

- *Execution unit*

The fixed-point execution unit performs the data transformations required by fixed-point RR operations, as well as the computation of the effective address for all storage references. It also provides data steering signals (register and MUX selects) to other sections of the chip during the execution of **move to** and **move from** special-purpose register instructions.

The FXU chip contains two execution units (Figure 5). Each unit contains one adder and one logic-unit (LU) functional block. The second execution unit also contains the multiply-and-divide logic (MDU) functional block. All multiply and divide operations are executed by the second execution unit, since there is no hardware in the first execution unit for these ops. The second execution unit also contains a three-leg adder to allow the simultaneous execution of dependent adder ops. When executing two instructions in parallel that have a common register as their target, the one in the second execution unit overwrites the result of the first execution unit, as long as the second execution unit does not receive a cancel.

Each execution unit is fed by its own copy of the S-latch, A-latch, and B-latch (see Figures 4 and 5), which generally correspond to the [RS], [RA], and [RB] operands referenced in the instruction word. All results of data transformation instructions are routed through an ALU MUX. Each ALU MUX drives the result data bus, from which data may be steered through bypass MUXes to the S-latch, A-latch, or B-latch, in addition to being latched by the register file input register. Store data from the fixed-point unit pass through each execution unit directly to the FXU pending-store queue (PSQ) register, and do not use the ALU MUX result data bus.

Each execution unit operates as a slave to the execution control unit and the decode units. The primary control interface is a 10-bit opcode derived by the decode unit from the 32-bit instruction word. These control vectors are decoded by each execution unit to determine the actions to be taken during the current execute cycle. A similar 10-bit opcode which represents the instruction currently in the decode phase of the pipeline is also decoded by each execution unit in order to set up latches which must provide timing-critical control signals early in the next execute cycle.

Each execution unit manages the MQ and XER registers as well as the XICR bus (by which the condition register CR on the I-cache chip is updated with execution results) for all instructions executed by the fixed-point chip.

The results of fixed-point instructions which update the CR are developed and placed on each execution unit's XICR bus. The only exceptions to this are the instructions LSCBX and RAC. The execution and condition code generation for those instructions are controlled by the execution control unit and the translation unit, respectively, which signal the results to the execution units.

The data transformation circuits in the first execution unit may be divided into two subunits:

1. Two-leg 32-bit carry-lookahead adder unit.
2. Logic unit, comprising a rotator, a count-leading-zeros unit, and a Boolean/mask/merge unit.

The data transformation circuits in the second execution unit may be divided into three subunits:

1. Three-leg 32-bit carry-lookahead adder unit.
2. Logic unit, comprising a rotator, a count-leading-zeros unit, and a Boolean/mask/merge unit.
3. A 36 × 36 multiply array and associated control logic for performing a converging divide algorithm.

The adder unit for the first execution unit is a 32-bit two-input carry-lookahead adder with one carry in; it produces a 32-bit sum, two carry-outs, and a zeros-and-ones detect signal. The ones detect signal is not used in this implementation. The adder receives its two inputs from the A0-latch and the B0-latch; the carry-in signal comes from a latch in the XGA RLM. The execution control unit loads the [RA] and [RB] operands into these registers during the decode cycle of an adder operation. If the operation calls for subtraction (e.g., negate, subtract, compare), the execution control unit loads the Boolean inverse of [RA] into the A0-latch. In such cases, the execution unit sets the carry-in bit to the adder during the execute cycle.

The adder for the second execution unit is a 32-bit three-input carry-save adder with one carry-in, followed by a 32-bit two-input lookahead adder with one carry-in. It produces a 32-bit sum with four carry-outs and a zeros-and-ones detect signal. A three-leg adder is used in the second execution unit so that sequential operations can be executed in one cycle, such as

```
A R1,R2,R3
A R4,R1,R5
```

The multiply/divide unit (MDU) in POWER2 has been enhanced over that of the original RS/6000 [3]. The multiply array allows for two-cycle multiplications for all multiply instructions (MUL, MULS, MULI), an improvement over the earlier RS/6000, which took three to five cycles for a multiply. The two divide instructions (DIV, DIVS) are implemented with an adaptation of the Anderson–Earle–Goldschmidt–Powers converging division algorithm for adapting floating-point numbers to fixed-point arithmetic. The DIVS and DIV instructions execute in 13 to 14 cycles; the DIV instruction may require three extra cycles if the algorithm converges from above. This is an improvement over the earlier design, which took 19 to 20 cycles for a multiply. When the divisor for the DIV instruction is the most negative number (0X80000000), two extra cycles are required.

The MDU is available only on the second execution unit. During decoding, multiply and divide instructions that come into the first execution unit are passed to the second execution unit, where they execute in the next cycle. The first execution unit may continue to execute for one cycle while the multiply or divide instruction is executing on the second unit. After the first cycle, only the second execution unit can execute until the multiply or divide is complete.

The POWER2 fixed-point divide algorithm is based on a floating-point converging algorithm. This algorithm starts with a table lookup to generate the first n good bits; it then iterates to produce the quotient. The remainder is then computed by multiplying the quotient by the divisor and subtracting it from the dividend.

• *Synchronization of fixed-point and floating-point units*
Synchronization between the FXU and FPU ensures the integrity of the association between data and the instruction that operates upon the data. For example, on a floating-point load instruction, it ensures that the data fetched by the FXU are loaded into the correct floating-point register (FPR). In both the POWER and POWER2 implementations, data integrity is maintained by synchronizing on all floating-point loads; a floating-point load executes in the FXU during the same cycle in which the rename stage in the FPU is selecting a new physical register for the load's target register.

Synchronization also helps preserve precise interrupts by ensuring that the FPU does not execute an interruptible operation (IOP), or subsequent instructions, before the FXU indicates that the execution may proceed. POWER implementations use two mechanisms to preserve precise interrupts [3]. An interruptible instruction latch in the FPU ensures that the FPU never executes an IOP ahead of the FXU. The FXU may not execute an IOP until the instruction reaches the FPU rename stage. A counter, indicating the relative execution positions of the FXU and FPU, limits how far either unit can be ahead of the other. The counter-based synchronization scheme relies on the FXU and FPU seeing all instructions on the IBUS.

In POWER2 implementations, the FXU does not see FPU arithmetic operations, and the FPU does not see FXU arithmetic operations. Therefore, a queueing scheme was devised to allow precise interrupts. As in POWER, the FPU may not execute IOPs ahead of the FXU. However, the synchronization has been relaxed to allow the FXU to execute all operations, except the floating-point loads, ahead of the FPU [4]. Thus, the FXU can execute all operations except floating-point loads ahead of the FPU and the FPU can execute all operations except IOPs ahead of the FXU. As a result, the POWER2 FXU can execute further up the instruction stream and, under certain conditions, provide data to the FPU in fewer cycles.

Address translation and data protection unit

The POWER2 address translation has also been improved over that of the original RS/6000. In the new scheme, only one memory reference is required for fetching page table entries, rather than the two memory references required by the earlier design. A second difference is that page table entries for sequential pages can be cached; formerly, page tables could not be cached. More details on address translation follow.

POWER2 is a register-intensive load/store architecture. Data transfers between CPU and memory occur via load/store operations only. For example, there are no instructions that take the contents of a memory location and add it to the contents of another memory location. To support the high data rates required by the pipelined processor, a high-speed data cache is placed next to the CPU. Most load/store operations can be served by the cache without degrading the FXU pipeline. Data not in the cache are fetched from the main memory. If the data are not in main memory, a page fault is taken, and the data are retrieved from mass storage (hard disk).

The major features of the storage mechanism are the following:

- Page size is 4 KB (2^{12} bytes).
- Maximum real memory size is 4 GB (2^{32} bytes, one million pages).
- Presumed minimum real memory size is 16 MB (2^{24} bytes, 4K pages).
- Virtual memory size is 2^{52} bytes.
- Number of segments is 2^{24} (16M).
- Number of transaction IDs is 2^{16} (64K).
- Hardware support for special segments (physical lock management on a 128-byte line).
- Automatic granting of locks in special segments.
- Memory-mapped I/O into I/O segments.

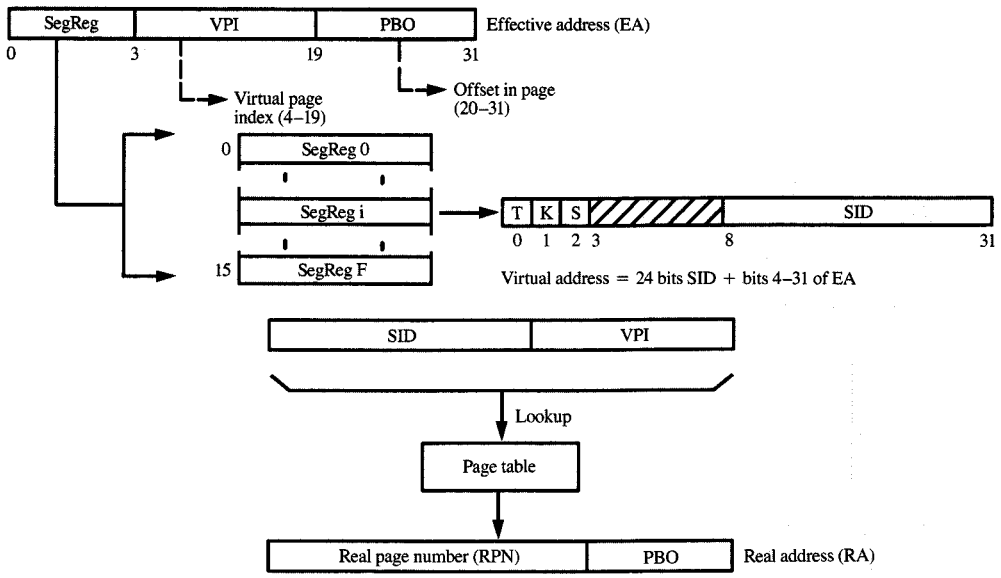

Figure 7

Address translation overview.

An overview of the address translation scheme is shown in **Figure 7**. Address translation is enabled by two bits in the machine state register (MSR)—one for data address translation, MSR(DR), the other for instruction address translation, MSR(IR). Both are independent bits and may be set differently. When translation is off (MSR = 0), the segment register is accessed only to determine whether it is an I/O segment for data storage accesses. If the T-bit in the segment register is zero, the effective address is the real address, and its numerical value is the address of a byte in main memory. If the T-bit is one, the effective address is sent to I/O.

However, if address translation is enabled (MSR = 1), the 32-bit EA is converted to a 52-bit virtual address as follows:

1. Use bits 0–3 of the effective address to identify one of the sixteen segment registers (SR[0 · · · 15]).
2. Concatenate the 24-bit segment ID (SID) field of the accessed segment register with bits 4–31 of the EA.

This 52-bit virtual address is then converted into a 32-bit real address (RA) via the hashed page table (HTAB or HPT).

The hashed page table (HTAB) contains a maximum of 2^{19} hash table entry groups (HTEGs). The HTEGs are addressable elements within the HTAB, and each HTEG contains eight page table entries (PTEs). Hashing the virtual address produces a pointer to the first of two HTEGs that could contain the translation for the virtual address. If the translation is not found in the initial HTEG, the virtual address is rehashed and a secondary HTEG is searched.

As mentioned above, each HTEG contains eight PTEs. Each PTE is composed of a two-word entry. The two-word entry contains fields to specify the segment ID (SID), the abbreviated virtual page index (AVPI), the real page number (RPN), page protection bits (pp), the reference bit (f), and the change bit (c). The organization of the hashed page table and the content of the page table entries are shown in **Figure 8**.

The translation between virtual address and real address is defined by the HTAB, and conceptually this table is searched by the address relocation hardware to translate every reference. However, for performance reasons the hardware keeps a translation lookaside buffer (TLB) which holds PTEs that have recently been used. A TLB is organized like a PTE; hence, it can be considered as a

cache that contains a subset of the page table entry. The TLB is searched before referring to the page table in storage. As a consequence, when software makes changes to the page table, it must issue the appropriate instructions to invalidate the TLB and thereby maintain the consistency between the TLB and the page tables.

When there is a TLB miss (i.e., no matching entry in the TLB), the HTAB mask and HTAB org in storage description register 1 (SDR1), the SID in the segment register, and the effective address are used to calculate the address of the first PTE group.

The FXU searches through the first group of PTE entries until a matching entry is found. A matching entry is one for which the valid bit is active, the SID in the segment register matches the SID of the PTE (bits 1–24; word 0), and bits 4–8 of the EA match the AVPI of the PTE (bits 27–31; word 0). If a matching entry is found, the RPN (bits 0–19) contained in word 1 of the PTE is concatenated with the offset (bits 20–31) of the EA to form the 32-bit word real address. However, if no match is found in the first set of eight PTEs, a secondary HTEG address is hashed, and the search is repeated as described above.

All eight PTEs in the secondary HTEG are searched to find a matching entry. If no matching PTE is found, the translation fails, a page fault occurs, and a data/instruction storage interrupt is generated.

Since POWER2 contains two FXU execution units, a translation miss from execution unit 0 must always be resolved before a miss from execution unit 1 can be resolved (this occurs when a load/store is received simultaneously from both execution units). Furthermore, any interrupt caused by unit 0 does not interfere with the translation of a miss for unit 1 and vice versa.

• *Translation lookaside buffer (TLB)*

The data flow for the data TLB is shown in **Figure 9**. The data TLBs are dual ports, two-way set-associative with 256 entries per set, and each entry contains two words (word 0 and word 1 of the PTE). An automatic hardware reload of TLB entries on a miss and HPT update of reference/change bits and data locking bits is included. The fixed-point unit performs all TLB reloading and HPT updating for the instruction cache unit, since it contains the only path to the data cache.

Data cache control and directory unit

The data cache control and directory unit is responsible for controling loads and stores for the FXU and FPU. Design features include increased data path bandwidth and D-cache support for multiple capacities and line sizes in a nonblocking store-back design.

The D-cache pipeline for a fixed-point load begins in the execution cycle with the access of the directory and status

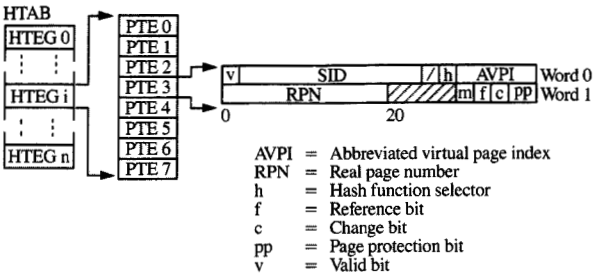

Figure 8

Hashed page table and page table entries.

arrays. The cache address tag and the TLB real page number are compared (along with the associated control, TLB hit, and cache valid bits) to form the cache *late selects*. Also in this cycle, the D-cache address is launched from the FXU and captured in the DCU. In the next cycle, the cache access cycle, the D-cache is read and the FXU's late-select signal instructs the DCU multiplexer to send the desired data to the FXU. This is the only two-chip crossing path in the entire processor complex. The data which arrive at the FXU are formatted and latched in the D-latch (where data can update the GPRs) and may be bypassed to the execution cycle input latches.

The data cache control and directory unit supports two design points. The high-performance design point incorporates a 256-byte line, 256KB D-cache with an eight-word memory interface. The low-cost design point incorporates a 128-byte line, 128KB D-cache with a four-word memory interface. Both design points implement a four-way set-associative D-cache; therefore, the corresponding cache sets are 64KB and 32KB, respectively. The longer line size improves performance on sequential data accesses. However, since a cache set is 16 pages in size, the lower four bits of the page address are required to index into the cache. This scheme places the following restriction on the operating system: Any data referenced with *translation on*, and then again with *translation off*, must keep the cache address portion of the virtual and real addresses equal. This aliasing restriction eliminates the chance of the same data being located in the cache in two locations. Memory bus bandwidth is augmented by a store-back D-cache design with two change bits per line. To handle a cache store-back operation, a 256/128-byte store-back buffer is implemented to hold the data until the memory bus is available.

With the increased computing power of POWER2, the data path bandwidth has been increased to prevent the

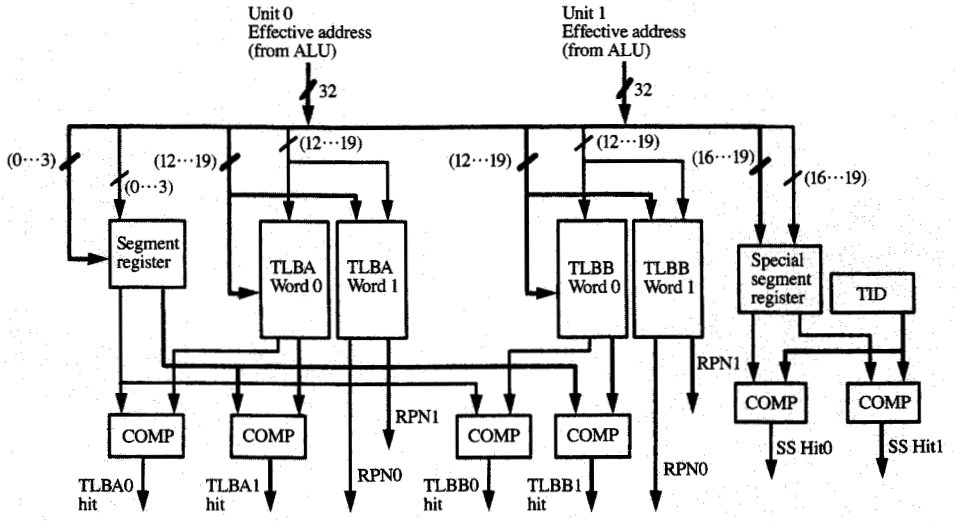

Figure 9

Data translation lookaside buffer organization.

data access from becoming a bottleneck. The D-cache logic path is fully dual-ported from the directory arrays and D-TLB to the D-cache itself. This allows the processor to execute two load/store instructions per cycle. A three-port adder in the EA generation path provides the capability to execute two update-form load/store instructions in parallel. Each data port to the FXU is a single word wide, allowing two independent data accesses. The new floating-point quadword load and store instructions, matched with two quadword-wide buses to the FPU, give the processor the ability to move four doublewords per cycle into the FPU. The D-cache custom array is capable of aligning data on a doubleword boundary, so quadword accesses need only be on a doubleword boundary.

The POWER2 D-cache is a nonblocking design; the D-cache can still be accessed on one port while the other port resolves a cache miss. A second miss blocks all accesses. Each port functions similarly to the POWER single-port design, with many of the same dataflow structures duplicated for the additional port. The following sections describe the data flow in more detail.

• *Load dataflow*

Figure 10 shows the load FXU/DCU dataflow. The D-cache logic, including directories, D-cache, and status array (not shown) is fully dual-ported. The design is dynamic; either port may be driven from E-unit 0 or E-unit 1. This is a nonblocking D-cache. Data can still be accessed from the D-cache with one outstanding "miss."

• *Fixed-point store dataflow*

Figure 11 shows the fixed-point store FXU/DCU dataflow. Two fixed-point stores can be executed per cycle and placed in the fixed-point pending store queue (XPSQ). One or two entries from the XPSQ can be written into the D-cache per cycle using any available port. The XPSQ is a non-overrunnable queue and has the first priority in clearing entries. Data are always transferred from the FXU to the DCU in the first cycle after executing a store and are placed in a fixed-point store data register (XSDR) on the DCU. Whenever a load/cache op is executed, it is compared to all entries in the XPSQ to check for a match. All compares are done using the effective address, and are performed down to the word level, bits 14–29.

• *Floating-point store dataflow*

Figure 12 shows the floating-point store FXU/DCU/FPU dataflow. Two floating-point stores can be executed per cycle and placed in the floating-point pending store queue (FPSQ). One or two entries from the FPSQ can be written into the D-cache per cycle using any available port. The FPSQ is an overrunnable queue capable of stopping execution of floating-point stores. The FPSQ has lower

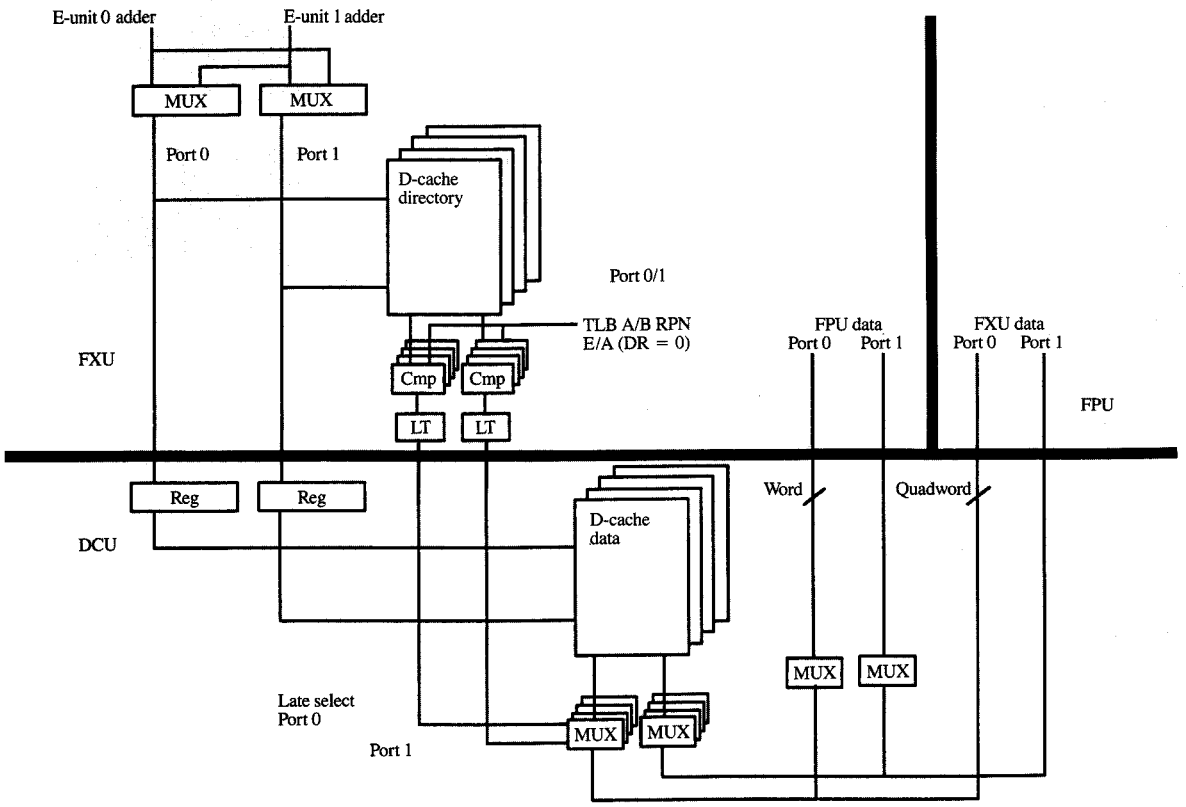

Figure 10

Dual load dataflow.

priority than the XPSQ. Data can be transferred from the FPU to the DCU after the FXU receives a data-ready from the FPU, depending upon bus availability. Data are placed in a floating-point store data register (FSDR) on the DCU. Whenever a load/fixed-point store/cache op is executed, it is compared to all entries in the FPSQ to check for a match. All compares are done using the effective address, and are performed down to the word level, bits 14–29.

- *Reload and store-back dataflow*

Figure 13 shows the reload and store-back dataflow. When the data are not found in the D-cache, a *reload* operation moves data from memory to the D-cache. If the D-cache destination for the new data contains data which have previously been modified, a *store-back* operation moves the modified data to memory.

The D-cache reload function is accomplished through a third (reload) port on the D-cache array. The DCU is given a reload command specifying the address and set. As memory data arrive, the data are written into the D-cache in the second half of the cycle. Load-through data are bypassed from the memory data latch and sent to the FXU or FPU in the first data cycle of all loads. If an ECC error is detected for bypassed data, the FXU or FPU will retry the request; the second data cycle will contain corrected data from the D-cache.

D-cache reloads are based on a true LRU algorithm with the memory bus delivering eight (four) words of data per cycle to fill a 256 (128)-byte cache line in eight cycles. The cache line is fetched in a wraparound fashion in which the first eight (four) words from memory contain the referenced data. The memory data are loaded directly into the D-cache on the reload port. This additional cache port provides minimal processor performance loss during a reload operation.

The D-cache store-back function supports half-line granularity by maintaining one change bit per half line. The

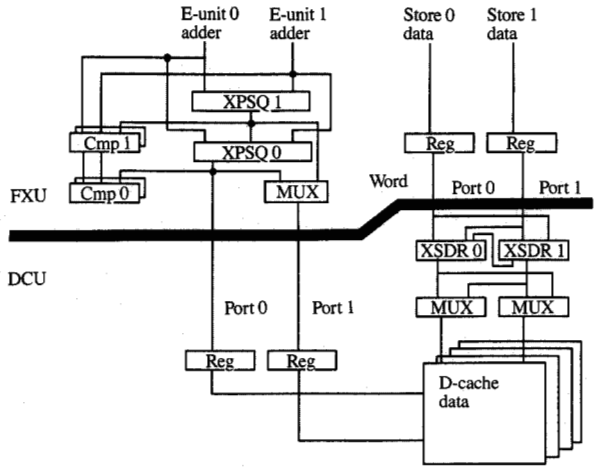

Figure 11

Dual fixed-point store dataflow.

DCU contains two cache line store-back buffers, SBB0 and SBB1. The two buffers allow optimal performance on reloads. SBB0 is used to postpone write functions to the memory. The FXU cache control will pass the reload command to the SCU immediately on a miss with one outstanding store-back in SBB0. The SCU will perform reads before writes and postpone store-back operations to give additional reload performance.

SBB1 has the additional capability of being able to be written from the XPSQ and the FPSQ. The XPSQ and FPSQ are not checked before a reload command is issued to the SCU. Once the reload has been given to the SCU, the control logic moves the replacement line to SBB1. The control logic checks the XPSQ and FPSQ against the replacement line. If there is a match, the stores are done to SBB1. SBB1 is then moved to SBB0 if it is available. If not, the control logic holds until SBB0 is unloaded to memory and then moves SBB1 to SBB0.

Data cache unit (DCU)

The DCU consists of four identical chips, which provide a four-way set-associative multiport store-back cache. The DCU supports two design points. The first consists of a 256-byte line, 256KB D-cache with an eight-word memory interface. The second consists of a 128-byte line, 128KB D-cache with a four-word memory interface. As shown in **Figure 14**, the DCU also provides several buffers for cache and DMA operations, as well as error detection/correction and bit steering for all data sent to and received from memory.

The DCU provides a 128-byte instruction reload buffer (IRB) for transferring instruction cache lines to the ICU, as well as store-back buffers for data cache operations. Data cache reload buffers are built into the data cache array macro. The DCU also provides an I/O cache for DMA operations. This cache holds up to four I/O cache lines and is controlled by the SCU. The following section describes the data cache array macro. The other DCU functions are described in more detail in the section on the SCU.

- *Data cache array macro*

The data cache array is a four-way set-associative 64KB dual-port array, with support for half-line store-back operations, as well as support for quadword access on a doubleword boundary. To meet the demands of the dual execution units, the data cache array macro has been enhanced over the previous cache designs [5]. The cache is a multiported design which uses a virtual multiport technique [6] and a standard single-port cell macro. This technique has kept the size of the array small while providing multiple ports. Other features of the array are line zeroing, port swapping, unaligned access, and an array built-in self test (ABIST).

The data cache array macro has three unique ports. There are two 36-bit read/write ports (Port 0 and Port 1) and one 72-bit write-only port (CRB port). The cache also has a 288-bit read-only port designed for storing back cache lines.

The virtual multiport technique provides a full three-port array to the outside logic, while internally it pipelines three sequential cycles within one processor cycle. The first two read/write cycles are always performed. The third cycle is for the CRB port and is only performed during cache reloads. The CRB port is used for loading data from the memory bus into the cache. Memory data are loaded one word per cache macro per cycle in a four-word memory system, and two words per cache macro per cycle in an eight-word system.

A port-swap feature minimizes the delay for a read to port 1 when preceded by a write to port 0. The feature swaps the port operations to guarantee that a port 1 read will never follow a port 0 write. This swap allows the RAM to take advantage of the faster array recovery following a read, and to start the port 1 access earlier than if it had followed a write cycle. The port-swap circuitry identifies when port 0 is writing, and then simply reverses the internal port clocks. Forcing port 0 to occur second, whenever a write occurs, allows it to become the priority port during the double-write case. The reload write maintains priority over the two execution unit ports.

The cache supports both aligned and unaligned accesses. The cache can read (write) from (onto) the cache-to-processor buses on doubleword boundaries. Because each

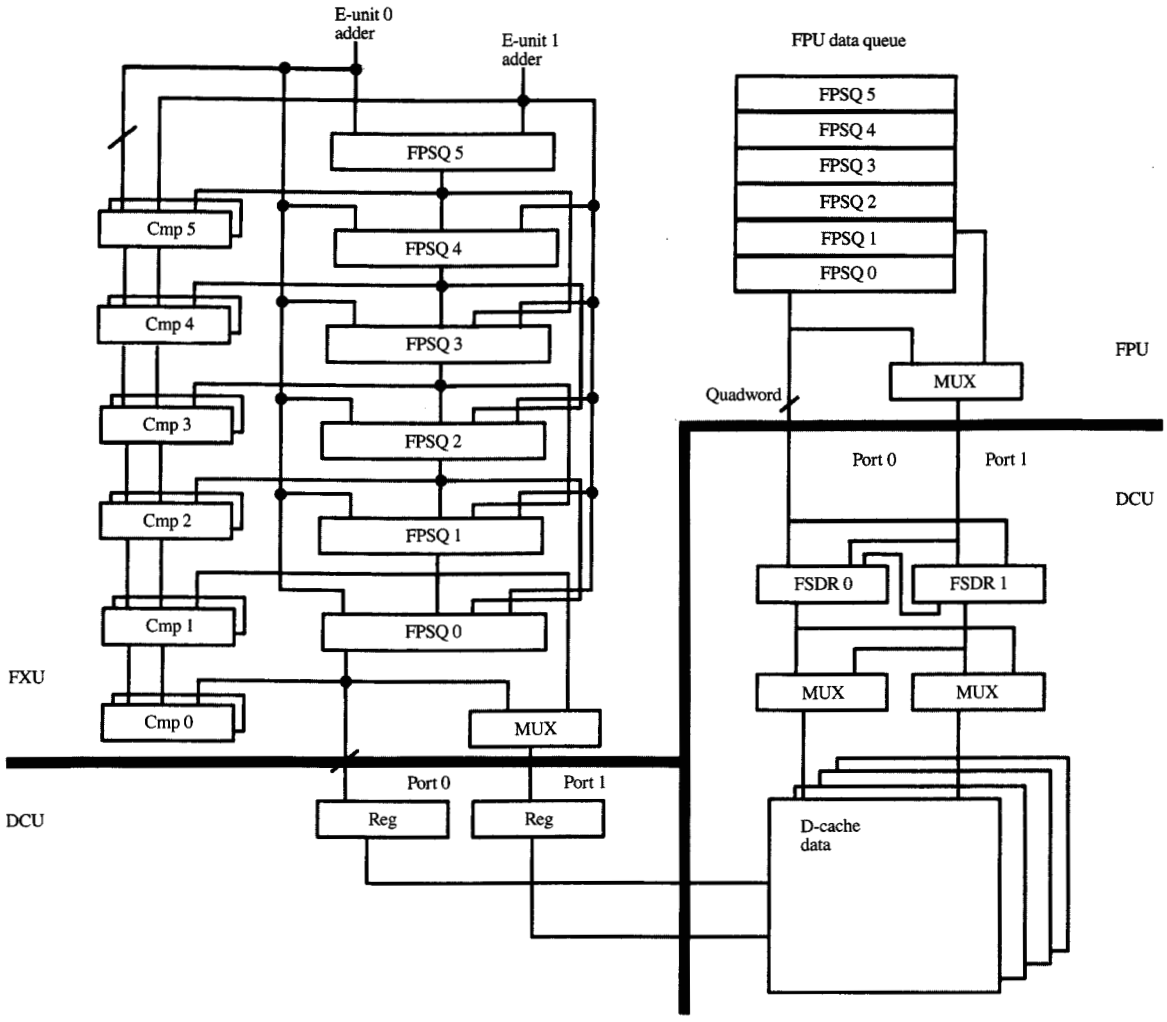

Figure 12

Dual floating-point store dataflow.

DCU chip provides four bytes of a quadword, each array requires access to data on either a word or half-word boundary. The cache is organized to read/write "aligned data," such as a word (bytes A1, B1, C1, D1), or data on a half-word boundary (see **Figure 15**). In this case, bytes C and D from word 1 can be merged with bytes A and B from word 2 to form the word C1|D1|A2|B2. The RAM increments the address for half of the bytes and then swaps the data between the upper and lower bytes for proper alignment. The last word on the line is not valid for this function; therefore, data misaligned across cache lines require two RAM accesses.

The virtual multiport cache is designed to be logically equivalent to a real multiport array. This requires a compare-bypass feature for the two read/write ports to guarantee that the execution unit receives the last data written when the ports simultaneously read and write the same address. Because the port-swapping feature will force the read access to occur first, an address comparator and data-in/data-out multiplexor are included to identify when an address collision has occurred so that it can bypass written data to the previously read port. The comparator not only identifies when the addresses are equal, but also when they are adjacent along a cache line. This is necessary

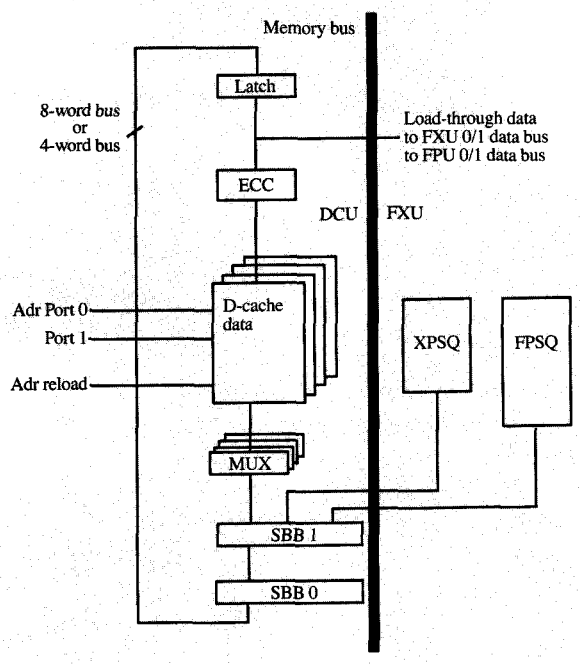

Figure 13

Reload and store-back dataflow.

when one access is aligned and the other is misaligned. Although the addresses are different, portions of the two accesses may overlap, and the comparator must be able to bypass half of the bytes during a read/write cycle.

To permit the software to initialize lines in the cache, a mechanism is provided by which the FXU can zero-out cache lines. This feature allows lines in the cache to be initialized without requiring the line to be transferred from memory. This initialization is significantly faster than a series of stores with zeros for data.

Storage control unit (SCU)

The main function of the SCU is to control the communication between the processor complex and the other system units: I/O control units, main memory unit, and the IPL read-only storage (ROS) unit. The SCU interfaces with the FXU and ICU processor chips across the P-bus, with I/O and ROS over the SIO bus, and with main memory using the memory address and control buses. Each of these interfaces has a unique set of control signals. In addition to managing these interfaces, the SCU contains logic for external interrupts and the performance monitor.

Figure 16 shows a high-level block diagram of the SCU. The SCU logic consists of the following areas: P-bus interface, SIO bus interface, memory interface, ROS interface, performance monitor, and external interrupts. The memory interface is further broken down into cache reload and store-back operations, memory scrub operations, error handling, and bit steering.

- *P-bus interface*

The P-bus interface supports three types of operations: memory, I/O load/store, and move. Memory addresses are moved from the P-bus into the P-bus memory queue and then moved out to main memory via the memory row/column address generation logic. I/O load/store operations to the SCU are used to read and write SCU registers, DCU registers, and I/O registers. The only SCU registers which can be read and written from the P-bus are the performance monitor registers, bank configuration registers, external interrupt registers, SCU control registers, and error registers. For I/O load/store operations to the DCU and I/O registers, the 128-byte PIO buffer is used to move data between the P-bus and SIO bus. Move operations transfer the interrupt-level control register (ILCR) to/from the P-bus in a single cycle.

- *Memory interface*

The memory interface is a high-speed, synchronous, split address/data bus which allows the processor, as well as I/O devices, to access main memory. The two primary changes to the memory interface are support for both a four-word and an eight-word memory-to-D-cache interface, capable of transfer rates of more than 2000 MB/s, and support for three cache line sizes. The memory interface also improves performance through its memory request queuing schemes and reload/store-back strategies. Like POWER, this implementation enhances reliability with memory scrubbing, ECC, and bit steering.

- *D-cache line size support*

The POWER memory interface supports 64-byte lines for the I/O and instruction caches and 128-byte lines for the D-cache. The POWER2 design supports both 128-byte and 256-byte D-cache lines while providing both 64-byte support for the POWER2 I/O cache line and 128-byte support for the I-cache. A new protocol was required not only for four-cycle and eight-cycle transfers, but also for two-cycle transfers. In addition, a new real-to-DRAM address translation was required by the SCU for the eight-word system. This translation is generated in a single cycle, as is described later.

- *Memory configuration*

The memory interface supports both a lower-cost four-word configuration and a high-performance eight-word configuration. The four-word interface maintains

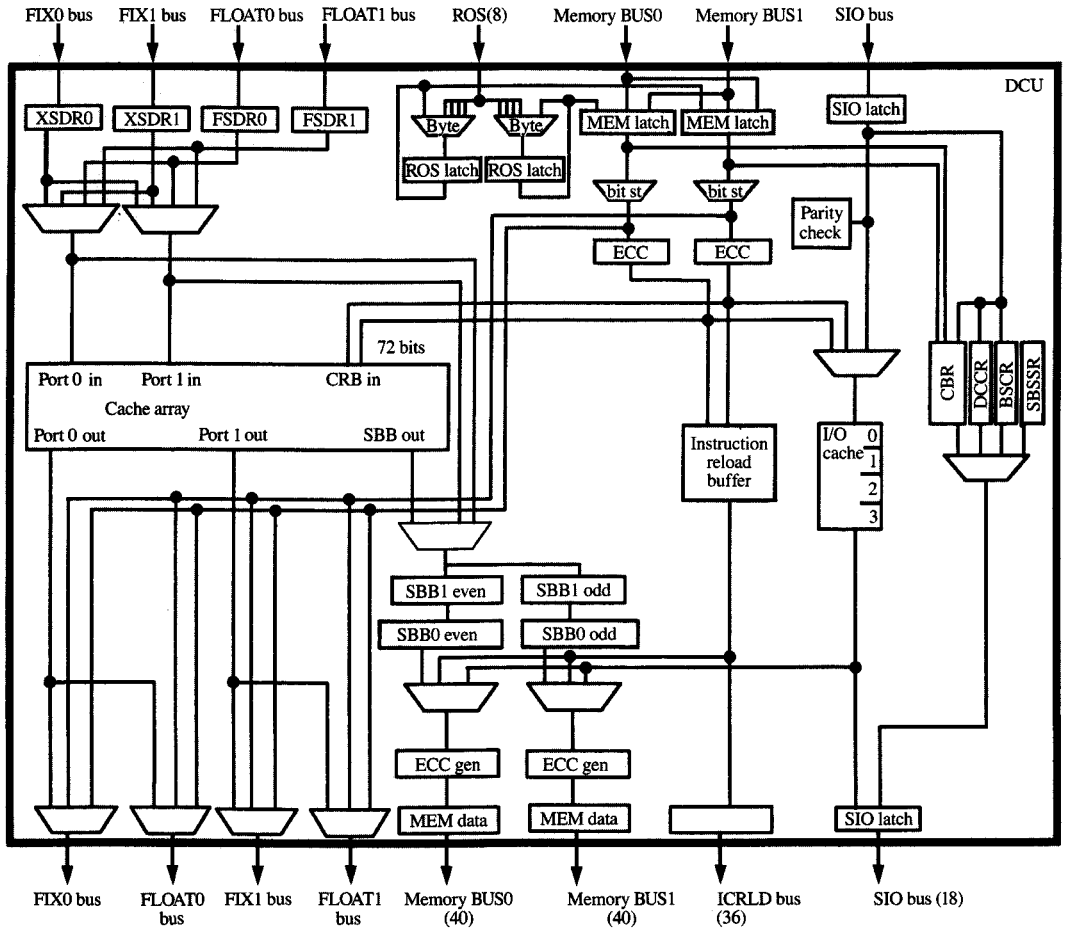

Figure 14

DCU dataflow.

compatibility with the existing memory cards and POWER's I/O subsystem, creating a stable interface for debugging the POWER2 processor chips. The high-performance eight-word interface supports the improved processing capabilities of POWER2. This unique memory interface selects the mode by detecting the number of installed memory cards.

Memory cards are two words wide; a minimum configuration consists of two cards. When two additional cards are installed, the memory interface becomes an eight-word bus. The design provides the customer an opportunity to buy a system with a minimal set of memory cards. With no changes to the planar, hardware, or software, the customer can add memory cards, providing both a wider data bus and a larger memory. The wider data bus doubles the memory performance.

The hardware automatically detects the number of memory cards present and establishes the width of the memory data bus. The on-card sequencer (OCS) monitors a memory card detect signal to determine how many memory cards are present. The OCS interfaces with the common on-chip processor (COP) logic [7] in the processor chips over a COP bus to configure the system. During IPL, the OCS initializes a mode latch in the FXU, SCU, and DCU that the processor chips use to determine a memory transfer's data width and number of cycles.

- *Memory request queues and controls*

To improve storage bandwidth and latency, the SCU queues and prioritizes memory requests and controls memory access. The SCU maintains three memory request queues. The first queue holds up to three processor

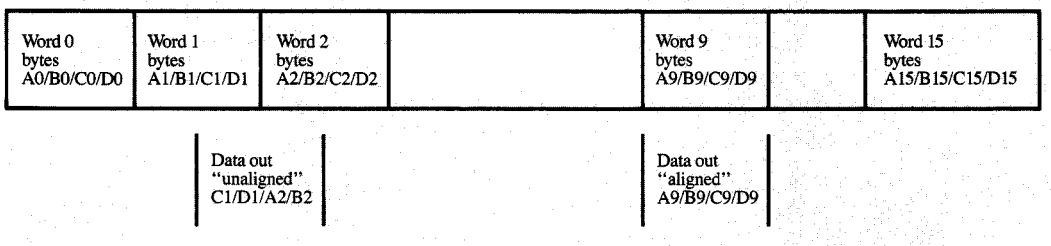

Figure 15

Aligned and unaligned data access across the cache line.

requests, the second holds two I/O DMA requests, and the third holds one memory scrub request. Three corresponding address generators create the address and the bank selects for the next request on the queue. The SCU arbitrates for the memory bus in parallel with the address generation. The SCU's memory arbiter grant logic selects one of three requests. The memory arbiter prioritizes the requests in the following order: DMA requests are highest in priority, followed by processor requests, followed by memory scrub operations. If back-to-back DMA requests are active along with processor requests, the arbiter grants the two DMA requests first, followed by one processor request.

While the arbiter is generating the bus grant, the bank select logic determines which one of 16 memory banks to activate. This logic compares the upper address bits of the real address with the base address bits in the bank configuration registers. The number of bits compared depends on the size of the field in the configuration register. If the addresses match, the bank select for that register is activated, and the transfer is completed.

The memory interface control reduces latency on back-to-back requests. By allowing two memory operations to be pending at any given time, the memory card begins to process the second request before the first is complete.

- *Cache reloads and storebacks*

When cache misses occur, cache reload and store-back operations move instructions and data between memory and the ICU, DCU, and FXU. These operations are jointly executed by the FXU, DCU, ICU, and SCU. D-cache miss performance is improved by implementing a load-through path for reloads, a store-back buffer which allows reloads to occur in parallel with a store-back operation to the buffer, and a high-priority reload feature. The I-cache miss sequence routes the data through the DCU to the ICU, reducing the pin count and providing the DCU's error detection and correction (ECC) coverage.

D-cache miss and store-back requests are initiated by the FXU and are sent as a processor request to the SCU. The SCU controls the four-word or eight-word transfers from memory to the DCU. As shown at the top of Figure 5, data pass through the bit-steering logic before being sent along two data paths. The ECC logic path goes to the D-cache. The load-through path bypasses the D-cache, sending data directly to the FXU and FPU data buses. When a new line of data is brought into the DCU, the word that satisfies the request is brought in first, minimizing latency. When the end of the line is reached, the first word of the line and the remaining sequential words are fetched until all eight four-word or eight-word data packets arrive in the DCUs.

The high-priority reload operation provides a performance advantage for all D-cache miss operations that require the cast-out of a "dirty" line. For these operations, two events must occur: Data from the D-cache must be written back to memory and data from memory must be stored into the D-cache. From a programmer's viewpoint, the data returned from memory are highest in priority. The data written back to memory are no longer needed. The high-priority reload design hides the cache line store-back penalties on the memory bus. When the SCU queues the cache reload and store-back requests, the reload and store-back addresses are monitored. If the addresses are not for the same cache line, the reload is given higher priority. The store-back operation must then wait to access the memory bus until there are no reload requests pending.

I-cache miss requests are initiated by the ICU and are sent as a processor request to the SCU. The SCU controls the four-word or eight-word data transfers from memory to the 128-byte I-cache reload buffer in the DCU. The SCU

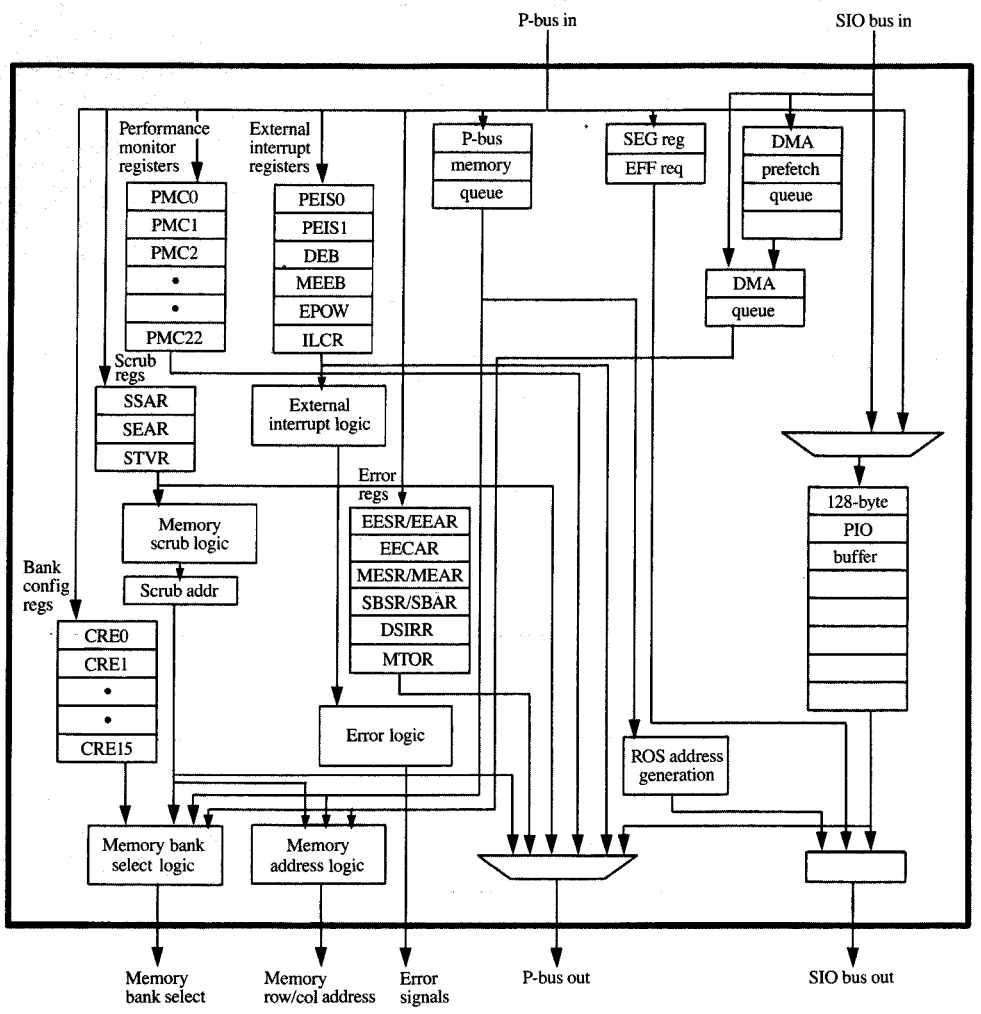

Figure 16

SCU high-level diagram.

controls the order in which data are loaded (quadword 0 or quadword 1 first) and when data are sent on the I-cache reload bus. The I-cache reload memory data pass through the DCU's bit steering and ECC logic. To reduce latency, the first quadword of the memory data includes the instruction requested by the ICU. A wraparound load of the instructions is performed.

- *Memory scrubbing*

To reduce the chances of an unrecoverable failure, the processor hardware provides a software-controlled memory-scrubbing function that attempts to find and correct single-bit errors before they become double-bit errors. The software uses three registers, located in local I/O space, to control the scrub function: the scrub start address register (SSAR), the scrub end address register (SEAR), and the scrub timer value register (STVR).

The scrub sequence consists of three memory transfers: a read operation to detect errors, a write operation to correct the errors, and a read operation to verify that the data have been corrected. If no errors are detected during the initial read operations, the subsequent write and read operations are not executed. When an error is detected, the hardware records the type of error (soft, hard, or uncorrectable) and the address where it was detected.

• *Error detection and correction*

The ECC logic allows the DCU to correct all single-bit errors and to detect all double-bit errors. The system memory bus is divided into either eight or four ECC words; each DCU chip receives either one or two words per data transfer. The ECC word contains 32 data bits, seven check bits, and one spare bit. Each word is encoded with a modified Hamming code when written to memory and is checked for errors when read from memory. If a single-bit error is detected, the DCU corrects the data and writes the ECC *syndrome* (eight-bit code which indicates which bit failed) into a register. This register and the corresponding failing address register and the corresponding failing address register in the SCU can be read by software to isolate the memory failure to a particular memory bit.

• *Bit steering*

Bit steering improves reliability by providing a "hot stand-by" bit if a memory bit fails. The SCU enables bit steering when a hard error in memory is detected through ECC and memory scrubbing. The SCU sets one of the bit-steering configuration registers (BSCR) to indicate the data word position to which the spare bit should be steered. Each DCU contains 16 BSCR registers, one for each memory bank. Each 8-bit BSCR contains the ECC syndrome. The DCU steers the spare bit into any data or check bit position within the ECC word during transfers to memory or from memory.

• *Read-only storage*

The ROS, located on the system planar, provides the code and data required to initialize the system and perform various disgnostic tests. This type of memory is typically separate from the larger main memory discussed earlier. For example, the POWER ROS interface used a separate address and data bus. Packaging the POWER2 on a multichip module limits the processor to 512 functional signals, leaving few signals available for the ROS interface. POWER2 uses the SIO bus as a ROS address bus, eliminating the need for a unique bus.

POWER2 reserves the upper 1 MB of the system address space for ROS. When the SCU detects an address in this range, it arbitrates for the SIO bus and generates the ROS address. The SCU controls the transfer of data from the ROS to the DCUs. When a full memory bus width (four or eight words) has been received, the data are written to either the I-cache or the D-cache. The fact that the data come from ROS is transparent to the ICU and FXU.

• *System I/O bus*

The SIO bus is a dedicated internal bus used for communication between the processing units and the I/O control units. The bus contains a total of 98 signal I/Os. The 86 bidirectional signals consist of a 72-bit multiplexed address and data bus (which includes eight bits of parity), and an eight-bit control bus with one parity bit and five control tags (address valid, data valid, acknowledge, processor lock, and checkstop). The other 12 unidirectional signals (four bus requests, four bus grants, and four busy signals) are used for SIO bus arbitration, and by the I/O control units to hold off I/O transfers.

The SIO bus supports the following transfers: I/O loads and stores, DMA block transfers, and I/O interrupt requests to the processing unit. All DMA transfers and I/O store transfers are *single-envelope*; the current transaction in progress must be completed before a new request is honored. All I/O load transfers have a disjoint reply packet. The processor issues the I/O load request for one of the I/O control units and then releases the SIO bus while it waits for the load data. When the load data are ready, the I/O control unit requests the SIO bus and transfers its data to the processor.

• *System I/O direct memory access*

The POWER2 I/O system overcomes many of the POWER bottlenecks in moving data between main memory and I/O devices, such as disk controllers and LAN adapters. Improvements include increased I/O transfer rates, support for more I/O controllers, I/O prefetch, and an I/O cache. First, a DMA sequence is described.

Data are moved between memory and I/O devices using 64-byte DMA read and write requests on the SIO bus. These operations are initiated by sending a command and address to the SCU over the SIO bus. Data are routed through the I/O buffers in the DCU. The SCU uses a round-robin buffer selection scheme to choose which I/O buffer will be used for the operation. For DMA write requests, the SCU receives the 32-bit real address and 64 bytes of data from the I/O control unit. It then loads the data into an I/O buffer two words at a time and unloads the data to memory either eight or four words at a time.

The POWER2 SIO bus supports up to four I/O control units; each unit manages four to eight Micro Channel channels (or slots). Each channel can transfer up to 80 MB/s using the Micro Channel Streaming Data protocol [8].

To support the increased I/O bandwidth which results from the multiple I/O control units, the POWER2 DCU contains an I/O cache. The POWER implementation of asingle I/O buffer [5] cannot sustain the high data transfer rates on Micro Channel. The POWER2 I/O cache improves both read and write performance. The I/O cache consists of I/O buffers in which data are prefetched for DMA read requests from I/O bus units. This removes the memory-card-DRAM latency for DMA data and provides a continuous stream on the SIO bus. For prefetch operations, the SCU fetches the 64 bytes requested as well

as the next sequential 64 bytes. For each new 64-byte request, the SCU unloads the data that were previously prefetched and fetches the next 64 bytes in parallel.

In addition, the I/O cache can buffer several I/O cache lines during a stream of DMA writes, hiding the latency of memory bus interference from the CPU. When access to the memory bus is obtained, the data are written in parallel with the loading of new cache lines.

- *External interrupt logic*

The external interrupt structure provides a mechanism for some external event, such as an I/O device requiring service, to break the normal flow of instructions. POWER2 provides a new high-performance external interrupt mechanism that incorporates a hardware high-priority detect, a priority mask, and a minimal set of single-cycle instructions.

The POWER external interrupt structure has two primary bottlenecks. First, the 64 interrupt bits are masked on an individual basis. When an interrupt occurs, the software interrupt handler iteratively loops through each bit of the interrupt register until the highest-priority bit set is found. Second, the interrupt register is mapped into I/O space, requiring a segment register to be set up every time a load or store to this register occurs. These I/O load/store instructions are inherently slow operations because of the setup and the handshake between the FXU and SCU.

To improve this scheme, the POWER2 external interrupt structure incorporates in hardware the high-priority detection that was previously handled in software. Additionally, the instructions used to interface to the interrupt logic have been changed from slow I/O load/stores to single-cycle move operations which operate on a set of special-purpose registers. A data field in one of these control registers provides the capability to set, reset, and update the external interrupt hardware.

- *Performance monitor*

The SCU implements a centralized performance monitor making possible a wide variety of POWER2 performance measurements [9]. The monitor consists of 22 software-accessible counters that monitor activity in each of the eight chips that make up the POWER2 processor. A performance monitor control register in the SCU selects the events to be monitored. Such data could not be obtained using an external monitor, since the processor is packaged on a multichip module where only the memory and SIO buses can be probed.

Summary

One of the primary goals of POWER2 was to improve performance by adding more execution units than those found in the original RS/6000, but still maintain a balanced system that avoids bottlenecks in the cache, memory, and I/O interfaces. POWER2 has achieved this goal in the FXU, DCU, and SCU by the addition of a fixed-point execution unit, a larger multiported data cache, an improved bus organization, and an improved I/O interrupt and DMA subsystem.

Acknowledgments

The authors would like to acknowledge several people who contributed to the POWER2 FXU, DCU, and SCU chips. Larry Thatcher, David Ray, Alex Spencer, Warren Maule, Roger Bailey, Mir Ali, Bert Williams, and Jennifer Le worked on the FXU logic design, and Joaquin Fentanes, Jr. worked on the physical design. Geordie Braceras worked on the data cache array macro. Larry Howell, Gary Countryman, Tao Brown, and Robert Wagner worked on the DCU logic design, and Mike Chung worked on the physical design. Doug Moran, Kurt Feiste, and Hakim Mosleh worked on the SCU logic design, and Adrienne Kokoszka worked on the physical design. These teams documented much of the technical detail presented in this paper. Ed Silha and John O'Quin were hardware and software architects for the entire POWER2 chip set.

References

1. H. B. Bakoglu, G. F. Grohoski, and R. K. Montoye, "The IBM RISC System/6000 Processor: Hardware Overview," *IBM J. Res. Develop.* **34,** 12–22 (1990).
2. G. F. Grohoski, "Machine Organization of the IBM RISC System/6000 Processor," *IBM J. Res. Develop.* **34,** 37–58 (1990).
3. G. F. Grohoski, J. A. Kahle, L. E. Thatcher, and C. R. Moore, "Branch and Fixed-Point Instruction Execution Units," *IBM RISC System/6000 Technology*, Order No. SA23-2619, IBM Corporation, 1990, pp. 24–32; available through IBM branch offices.
4. T. N. Hicks, R. E. Fry, and P. E. Harvey, "POWER2 Floating Point Unit: Architecture and Implementation," *IBM J. Res. Develop.* **38,** 525–536 (1994, this issue).
5. William R. Hardell, Jr., Dwain A. Hicks, Lawrence C. Howell, Jr., Warren E. Maule, Robert Montoye, and David P. Tuttle, "Data Cache and Storage Control Units," *IBM RISC System/6000 Technology*, Order No. SA23-2619, IBM Corporation, 1990, pp. 44–51; available through IBM branch offices.
6. Geordie Braceras et al., "A 200 MHz Internal/66 MHz External 64KB Embedded Virtual Three-port Cache SRAM," *ISSC Digest of Technical Papers*, February 1994, IEEE, Piscataway, NJ, pp. 262–263.
7. Ion M. Ratiu, "Pseudorandom Built-In Self-Test," *IBM RISC System/6000 Technology*, Order No. SA23-2619, IBM Corporation, 1990, pp. 74–77; available through IBM branch offices.
8. James O. Nicholson, "Micro Channel Features," *IBM RISC System/6000 Technology*, Order No. SA23-2619, IBM

Corporation, 1990, pp. 52–55; available through IBM branch offices.
9. E. H. Welbon, C. C. Chan-Nui, D. A. Hicks, and D. J. Shippy, "The POWER2 Performance Monitor," *IBM J. Res. Develop.* **38,** 545–554 (1994, this issue).

Received August 3, 1993; accepted for publication May 23, 1994

David J. Shippy *IBM Systems Technology & Architecture Division, 11400 Burnet Road, Austin, Texas 78758* (shippy@ibmoto.com). Mr. Shippy has a B.S. degree in electrical engineering from the University of Kentucky and an M.S. degree in computer engineering from Syracuse University. He is currently an Advisory Engineer working in the PowerPC processor development group at Somerset. Prior to that he worked on the POWER2 processor design. He started with IBM in Endicott, New York, in the System/370 midrange processor development group. Mr. Shippy has received two IBM Outstanding Technical Achievement Awards and one IBM Division Award, as well as three IBM Invention Achievement Awards. He holds several patents and has numerous technical publications.

T. W. Griffith, Jr. *IBM RISC System/6000 Division, 11400 Burnet Road, Austin, Texas 78758 (GRIFFITH at AUSVM6).* Mr. Griffith received a B.S. in electrical engineering from the University of Texas in 1979 and an M.S. in electrical engineering from Florida Atlantic University in 1982. He joined IBM in 1979 in Boca Raton, Florida, where he worked on Series/1 test engineering. He transferred to Austin in 1986 to work on the RS/6000 I/O channel controller.