Functional Design of the HP PA 7300LC Processor

Microarchitecture design, with attention to optimizing specific areas of the CPU and memory and I/O subsystems, is key to meeting the cost and performance goals of a processor targeted for midrange and low-end computer systems.

by Leith Johnson and Stephen R. Undy

The PA 7300LC microprocessor is the latest in a series of 32-bit PA-RISC processors designed by Hewlett-Packard. Like its predecessor, the PA 7100LC,[1,2] the PA 7300LC design focused on optimizing price and performance. We worked toward achieving the best performance possible within the cost structures consistent with midrange and low-end systems. This paper describes the microarchitecture of the two main components of the PA 7300LC: the CPU core and the memory and I/O controller (MIOC).

CPU Core Microarchitecture Design

Approximately one-half of the engineering effort on the PA 7300LC processor was dedicated to the design of the CPU core. The CPU core includes integer execution units, floating-point execution units, register files, a translation lookaside buffer (TLB), and instruction and data caches. Fig. 1 shows a block diagram of the CPU core.

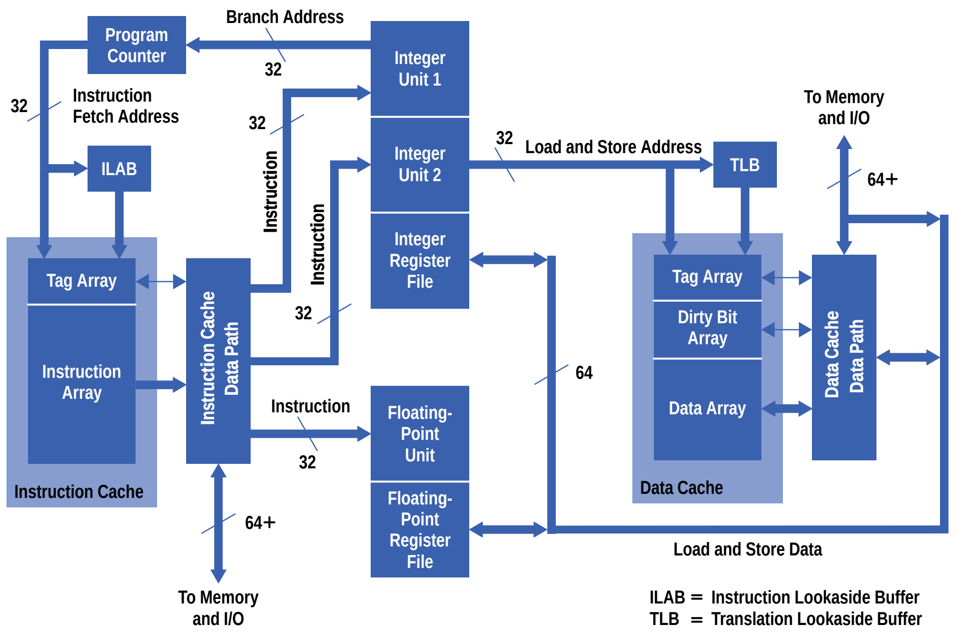


***Fig. 1.** The PA 7300LC CPU core block diagram.*

Core Design Objectives

The design objectives for the PA 7300LC processor were to provide the best possible performance while choosing the proper set of features that would enable a system cost appropriate for entry-level and high-volume workstation products. To reach this goal, we integrated large primary caches on the processor chip and developed a tight coupling between the CPU core and the memory and I/O subsystems. The design objectives for the PA 7300LC are discussed in detail in ***Article 6***.

CPU Core Differences

The PA 7300LC CPU core is derived from the PA 7100LC CPU design.[1,2] Although the PA 7300LC has many similarities with its predecessor, there are some key differences in the design that allowed us to meet our performance objectives. The first difference is that the PA 7300LC runs at 160 MHz compared to only 100 MHz for the PA 7100LC. The most obvious difference is the large primary instruction and data caches integrated directly onto the PA 7300LC chip. The PA 7100LC only has a small (1K-byte) instruction cache on the chip. Also, the organization of the caches was changed to avoid many of the stall cycles that occur on the PA 7100LC. The cache organization is discussed later in this article. The PA 7300LC has a 96-entry TLB, compared to 64 entries on the PA 7100LC. Finally, the PA 7300LC has a four-entry instruction lookaside buffer (ILAB) while the PA 7100LC's ILAB contains one entry.

Pipeline and Execution Units

Like all high-performance microprocessors, the PA 7300LC is pipelined. What is notable about the PA 7300LC pipeline is that it is relatively short at six stages, while running at 160 MHz.

Fig. 2 shows a diagram of the PA 7300LC pipeline. It does not differ greatly from the pipelines used in the PA 7200, PA 7100LC, or PA 7100 processors.[1,2,3,4] The following operations are performed in each stage of the pipeline shown in Fig. 2:

1. Instruction addresses are generated in the P stage of the pipeline.

2. The instruction cache is accessed during the F stage.

3. The instructions fetched are distributed to the execution units during the first half of the I stage. During the second half of the I stage, the instructions are decoded and the appropriate general registers are read.

4. The integer units generate their results on the first half of the B stage. Memory references, such as load and store instructions, also generate their target address during the first half of the B stage.

5. Load and store data is transferred between the execution units and the data cache on the second half of the A stage.

6. The general registers are set on the second half of the R stage.

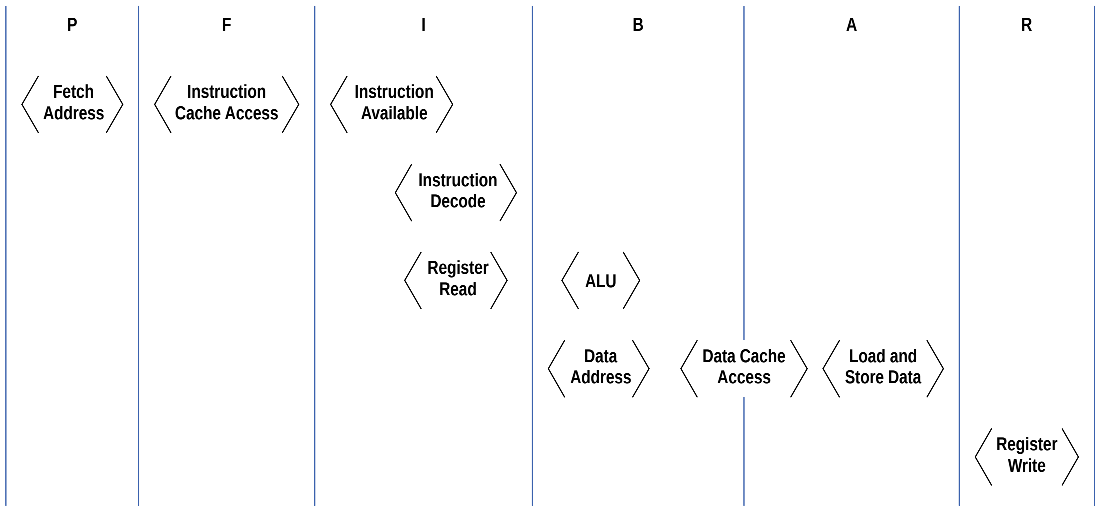
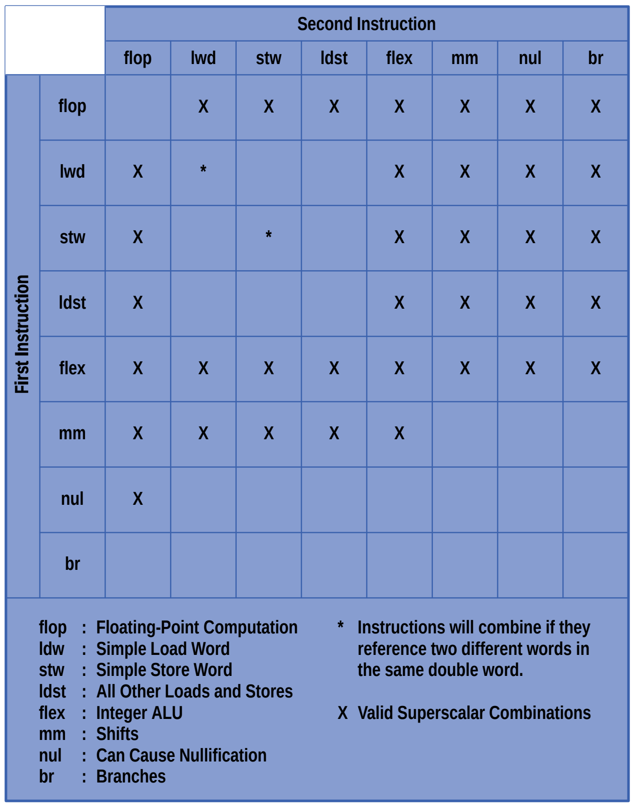


Fig. 2. *The PA 7300LC pipeline diagram.*

Superscalar Processor. The PA 7300LC is a superscalar processor, capable of executing two instructions per pipeline stage. This allows it, at 160 MHz, to execute at a maximum rate of 320 million instructions per second. This, however, is a peak rate that is rarely achieved on real applications. The actual average value varies with the application run. The theoretical maximum assumes the proper mix of instructions, but not every pair of instructions can be bundled together for execution in a single cycle. Fig. 3 shows which pairs of instructions can be bundled for execution in a single pipeline stage.

Delayed Branching. The PA-RISC architecture includes delayed branching.[5] That is, a branch instruction will not cause the program counter to change to the branch address until after the following instruction is fetched. Because of this, branches predicted correctly with a simple branch prediction scheme execute without any pipeline stalls. The majority of the remaining branches execute with only a single stall (see Fig. 4).

Two Integer Execution Units. The PA 7300LC contains two integer execution units. Each contains an ALU (arithmetic logic unit) that handles adds, subtracts, and bitwise logic operations. Only one unit, however, contains a shifter for handling the bit extract and deposit instructions defined in the PA-RISC architecture. Since only one adder is used to calculate branch targets, only one execution unit can process branch instructions. This same unit also contains the logic necessary to



First Instruction	Second Instruction: flop	lwd	stw	ldst	flex	mm	nul	br
flop		X	X	X	X	X	X	X
lwd	X	*			X	X	X	X
stw	X		*		X	X	X	X
ldst	X				X	X	X	X
flex	X	X	X	X	X	X	X	X
mm	X	X	X	X	X			
nul	X							
br								

flop	:	Floating-Point Computation
ldw	:	Simple Load Word
stw	:	Simple Store Word
ldst	:	All Other Loads and Stores
flex	:	Integer ALU
mm	:	Shifts
nul	:	Can Cause Nullification
br	:	Branches

* Instructions will combine if they reference two different words in the same double word.

X Valid Superscalar Combinations

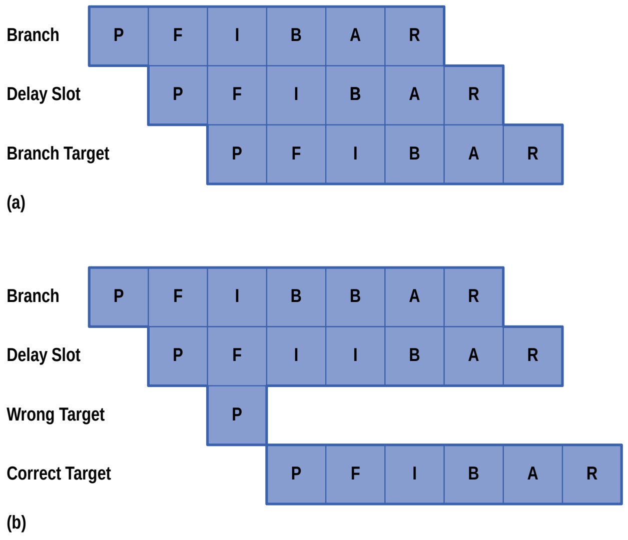


Fig. 3. *Valid superscalar instruction combinations for PA 7300LC.*

Fig. 4. *Branch behavior. (a) Correctly predicted branch. (b) Incorrectly predicted branch.*

calculate nullification conditions.* By limiting execution to only one branch or nullifying instruction per pipeline stage, we avoided a great deal of functional complexity. Finally, only one unit contains the logic to generate memory addresses. Since the data cache is single-ported, there is no need to have two memory addresses generated per cycle. In special cases, however, two integer load or store instructions may be bundled together, provided they use the same double-word address. As mentioned before, these asymmetries between the integer units prevent any two arbitrary integer instructions from bundling together. However, even with this limitation, compilers are able to take advantage of the integer superscalar capabilities of the PA 7300LC.

Multimedia Instructions. The PA 7300LC integer units implement a set of instructions first introduced on the PA 7100LC that accelerate multimedia applications.[1,2,6] These instructions allow each integer unit to perform two 16-bit adds, subtracts, averages, or shift-and-adds each cycle. Because of superscalar execution, the PA 7300LC can execute four of these operations per cycle for a peak rate of 640 million operations per second.

Floating-Point Unit. The PA 7300LC contains one floating-point unit. Contained in this unit is a floating-point adder and a floating-point multiplier. The adder takes two cycles to calculate a single- or double-precision result. It is pipelined so that it can begin a new add every cycle. The multiplier takes two cycles to produce a single-precision result and three cycles for a double-precision result. It can begin a new single-precision multiply every cycle and a new double-precision multiply every other cycle. Divides and square roots stall the CPU until a result is produced. It takes eight cycles for single-precision and 15 cycles for double-precision operations.

Instruction Cache and ILAB

Integrating a large primary instruction cache onto the processor chip broke new ground for PA-RISC microprocessors. In the past, our processor designs relied on large external primary caches. With the PA 7300LC, we felt that we could finally integrate enough cache memory on the processor chip to allow fast execution of real-world applications. Indeed, we have integrated twice as much cache on-chip as the PA 7100LC used externally in the HP 9000 Model 712/60 workstation (i.e., 128K bytes versus 64K bytes). The integrated cache not only improves performance but also reduces system cost, since an external cache is no longer mandatory.

See Subarticle 7a: ***Timing Flexibility***.

* The PA-RISC architecture enables certain instructions to conditionally nullify or cancel the operation of the following instruction based on the results of the current calculation or comparison.

Primary Instruction Cache. The PA 7300LC primary instruction cache holds 64K bytes of data and has a *two-way set associative* organization. A set associative cache configuration is difficult to achieve with an external cache, but much more practical with an integrated cache. When compared to a similarly sized directly mapped cache, it performs better because of higher use and fewer collisions. We chose a two-way associative cache over other ways to save overhead caused by the replication of comparators and to reduce the propagation delay through the way multiplexer.

The primary instruction cache is virtually indexed and physically tagged. Because the PA-RISC architecture restricts aliasing** to 1M-byte boundaries, we could use a portion of the virtual address (in this case, three bits) to form the index used to address the cache. To avoid using virtual address bits would have required us either to place the virtual-to-physical translation in series with cache access (increasing the cache latency) or to implement a large number of ways of associativity (in the case of a 64K-byte cache, this would have required a 16-way set associative organization).

Data Array Requirements. The instruction cache is composed of a tag array and a data array, each containing addresses and instructions. Without using more wires or sense amplifiers than those found in a conventional cache organization, we organized the data arrays in an unusual fashion in the primary caches on the PA 7300LC to meet two requirements.

The first requirement is for the instruction cache to supply two instructions per cycle to the execution units. Because the cache is two-way set associative, each location, or *set*, contains instructions corresponding to two distinct physical addresses. Thus, for any given set (determined by the instruction fetch address), there are two possible choices for the instructions being read. Each of these choices is called a *group* (see Fig. 5a). For speed reasons, both groups are read from the instruction data arrays simultaneously. Logic that compares the physical addresses in the tag arrays (one per group) with the physical address being fetched from the data array determines which group is selected and sent to the rest of the CPU. Since this is the normal instruction fetch operation, it must be completed in a single processor cycle.

The second requirement is to be able to write eight instructions simultaneously, all to the same group. Because a write occurs as part of the cache miss sequence, it is important that the write take only a single cycle to interrupt instruction fetches as little as possible.

Fig. 5a shows the conventional method of addressing data arrays. Because of electrical and layout considerations, the upper four instructions of each eight-instruction-long cache line are kept in a separate array from the lower four instructions. Both the upper and lower arrays are addressed and read concurrently. There are four arrays in total: group 0 upper, group 0 lower, group 1 upper, and group 1 lower. The instruction fetch address sent to the instruction cache, Address[0:11], contains twelve bits. One address bit, Address[10], selects between the upper and lower arrays. The rest of the address bits, Address[0:9] and Address[11], go to all four arrays and determine which set (Set[x]) is read out of the arrays. This is accomplished with the 11-to-2048 decoders. In reality, four decoders, one for each array, would be needed, but they all connect to the same address. As discussed above, there are two possible pairs of instructions to choose from with a given address. A signal from logic called *hit compare* selects between the two possibilities. In the example shown in Fig. 5a, instructions 0 and 1 from group 0 are selected from the instruction cache.

This conventional approach meets our first requirement. However, it does not meet our second requirement. It cannot access all eight instructions as a single group simultaneously. This is because a cache line is located in two adjacent sets and only half of the line can be read (or more important, written) at any one time. For example, if the group 0 upper array is supplying instructions 0 and 1, it obviously cannot supply 2 and 3. The only way to solve this problem with the conventional approach is to split each array into two halves. This, however, would require twice as many wires and possibly sense amplifiers producing a sizable increase in area cost. By making a slight modification to the way the data arrays are organized and addressed, we found we could avoid this pitfall and meet both of our requirements.

Our addressing approach on the PA 7300LC is called *checkerboarding*. Fig. 5b shows how instructions are fetched from the instruction cache on the PA 7300LC. There are, again, four arrays: left upper, left lower, right upper, and right lower. The most significant address lines, Address[0:9], go to all four arrays, while Left[11] goes only to the two left arrays and Right[11] goes only to the the two right arrays. A single address bit, Address[10], selects between the upper and lower arrays, as before.

When an instruction is fetched, both Left[11] and Right[11] are set to the value of Address[11]. Because of this, the operation is virtually identical to the conventional approach described above, except for one key difference: a cache line for a given group is spread across all four arrays, rather than just two. This can be seen in Fig. 5b, where the instructions corresponding to group 1 have been shaded. Each array contains pieces of cache lines from both groups in a checkerboard fashion.

Fig. 5c illustrates how checkerboarding allows simultaneous access to an entire cache line. By setting Left[11] to the group desired and Right[11] to the opposite value, all eight instructions from one group can be read or written. In the example shown in the figure, an entire cache line from group 1 is read out. Left[11] is set high, while Right[11] is set low. Address[0:9] selects which pair of sets, Set[x] and Set[x+1], are accessed. Fig. 6 lists the results of addressing the arrays with the various combinations of values on Left[11] and Right[11].

** Aliasing refers to intentionally allowing two different virtual addresses to map to the same physical address. The PA-RISC architecture restricts the number and location of bits that may differ between two virtual addresses.

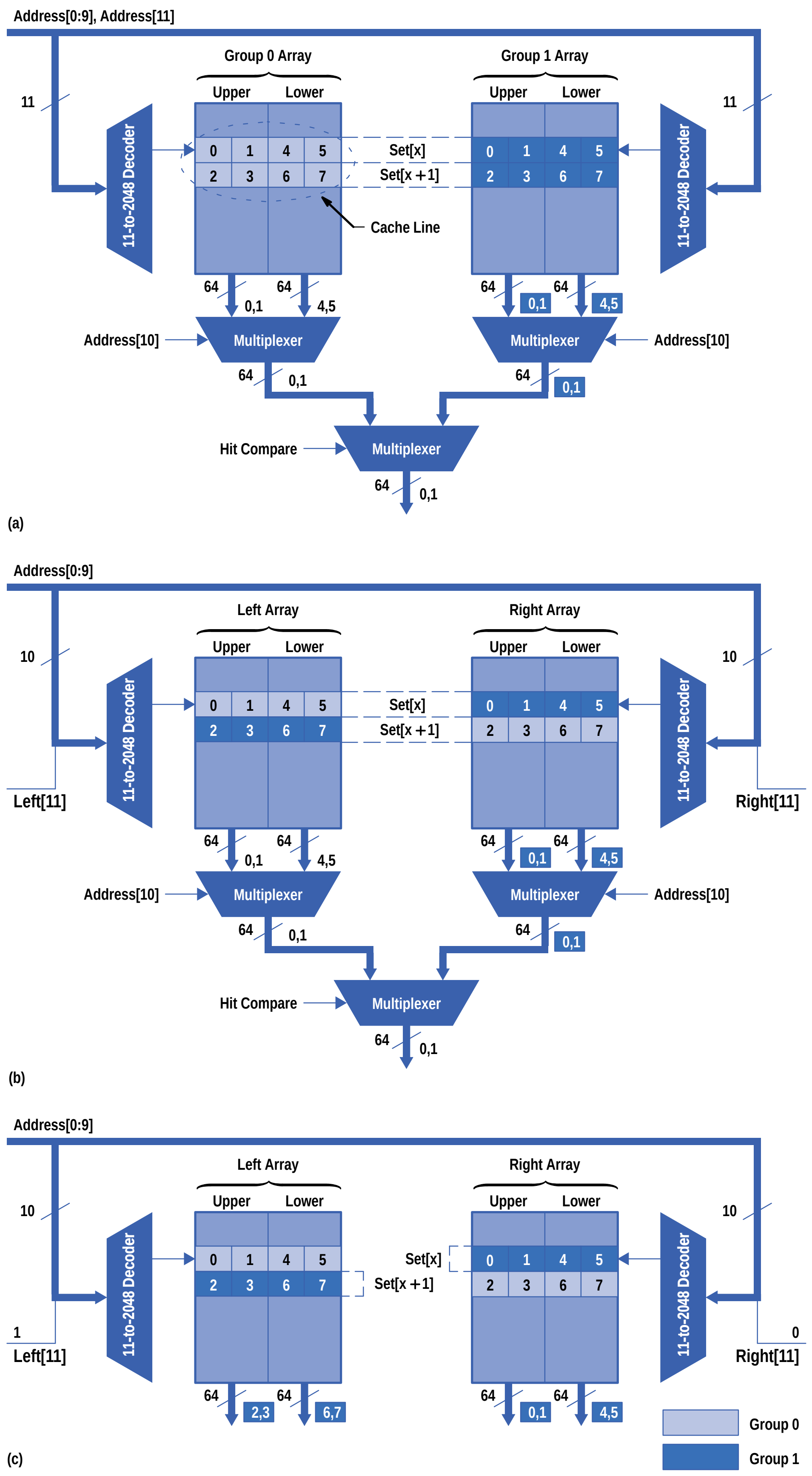


Fig. 5. *(a) Conventional two-cache organization. (b) Checkerboard instruction fetch. (c) Checkerboard full line access.*

Left[11]	Right[11]	Left Upper Array Output	Left Lower Array Output	Right Upper Array Output	Right Lower Array Output
0	0	Set[x] Group 0 Instructions 0,1	Set[x] Group 0 Instructions 4,5	Set[x] Group 1 Instructions 0,1	Set[x] Group 1 Instructions 4,5
0	1	Set[x] Group 0 Instructions 0,1	Set[x] Group 0 Instructions 4,5	Set[x +1] Group 0 Instructions 2,3	Set[x +1] Group 0 Instructions 6,7
1	0	Set[x +1] Group 1 Instructions 2,3	Set[x +1] Group 1 Instructions 6,7	Set[x] Group 1 Instructions 0,1	Set[x] Group 1 Instructions 4,5
1	1	Set[x +1] Group 1 Instructions 2,3	Set[x +1] Group 1 Instructions 6,7	Set[x +1] Group 0 Instructions 2,3	Set[x +1] Group 0 Instructions 6,7

Note: x is determined by Address[0:9]

Fig. 6. *The meaning of checkerboard address lines Left[11] and Right[11].*

Instruction Cache Hit Stages. The CPU core will attempt to fetch a pair of instructions from the instruction cache every cycle during which it is not stalled. For example:

- The instruction fetch address arrives at the instruction cache at the end of the P stage of the pipeline.
- On the first half of the F stage, the word line decoders fire one word line to each array.
- On the second half of the F stage, the array is read, driving its value onto the bit lines to the sense amplifiers. The way multiplexer then selects the proper pair of instructions from the sense amplifier outputs.
- On the first half of the I stage, the instructions are driven to the execution units for decoding and execution.

Instruction Cache Miss Stages. In the case of an instruction cache miss, which is known by the end of the F stage of the pipeline, the entire pipeline will stall. A read request for an entire cache line will then be sent to the memory controller. This request is called a *copyin request*. A 64-bit data path between the memory controller and the instruction cache requires a minimum of four cycles to transfer the entire cache line to the instruction cache. Four cycles are required because the memory controller can only deliver 64 bits per cycle and a cache line contains 256 bits. The memory controller will return the pair of instructions originally intended to be fetched first, regardless of the pair's position within the cache line. As each pair of instructions is returned from memory, it is written into a write buffer. The instructions can be fetched directly from this buffer before they are written to the cache, with the first pair's arrival causing the pipeline to resume execution. This capability is commonly referred to as *streaming*. In effect, the write buffer forms a third way of associativity. After the last pair of instructions arrive from memory, the write buffer contents are written to the cache in one cycle.

Unified Translation Lookup Table. Since the instruction fetch address is a virtual address, it must be mapped into a corresponding physical address at the same time the instruction cache arrays are being accessed. Normally, a full instruction translation lookaside buffer, or ITLB, is used to perform this function. On the PA 7300LC, as on all recent PA-RISC processors, we felt that the performance improvements achieved with a separate ITLB and DTLB (for data accesses) did not warrant the increased chip area costs. Instead, we opted for a unified TLB that performs both instruction and data translations.

Instruction Lookaside Buffer (ILAB). Because both an instruction and a data translation are required on many cycles, a smaller structure called an instruction lookaside buffer, or ILAB, is used to translate instruction addresses, while the larger unified TLB is free to translate data addresses. The four-entry ILAB is a subset of the unified TLB and contains the most recently used translations. This strategy is quite effective because instruction addresses, unlike data addresses, tend to be highly correlated in space in that they generally access the same page, a previous page, or the next page.

When an instruction address does miss the ILAB, normally because of a branch, the pipeline will stall to transfer the desired translation from the unified TLB to the ILAB. We designed in two features to mitigate these ILAB stalls. On branch instructions that are not bundled with a memory access instruction (such as a load or store), the unified TLB will be accessed in parallel with the ILAB, in anticipation of an ILAB miss. If the ILAB misses, the normal ILAB stall penalty will be reduced. The second feature we added was ILAB prefetching. Every time the CPU begins executing on a new instruction page, the TLB will take the opportunity to transfer the translation for the next page into the ILAB. This can completely avoid the ILAB misses associated with sequential code execution.

Data Cache and TLB

We designed the data cache array to be very similar to the instruction cache arrays. Like the instruction cache, the data cache is two-way set associative, virtually indexed, and physically tagged. It is composed of three arrays:

- A data array, which has the same checkerboard organization as the instruction cache data array
- A tag array, which is almost identical to its instruction cache counterpart

- A dirty bit array, which has no counterpart in the instruction cache. This array keeps track of whether a data cache line has been modified by the instruction stream.

Although organized in a way similar to the instruction cache, the data cache's internal operation and effect on the CPU pipeline are quite different. The data cache and TLB operate in the A and B stages of the pipeline. A load instruction causes a data address to be generated in the first half of the B stage. The data cache word line decoders operate on the second half of the B stage. On the first half of the A stage, the arrays drive their values out. Based on the comparison between the physical address and the output of the tag arrays, the way multiplexer then selects the proper data value. This word or double-word value is then driven to the integer and floating-point units during the second half of the A stage.

A store instruction generates a data address in the same manner as a load instruction. That address is used to read from the tag array as described above. Instead of using the store address to read from the data array, however, the address from the head of a two-entry store queue (Fig. 7) is used to index into the data array on the second half of the B stage. The data from the head of the store queue is written into the data array on the first half of the A stage. The data from the store instruction is driven from the integer or floating-point units to the data cache on the second half of the A stage where it is written into the tail of the store queue.

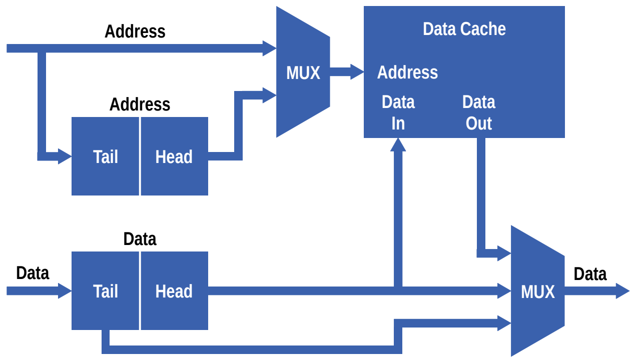


Fig. 7. *The store queue.*

Store Queue. A load can retrieve data directly out of the store queue if it is to the same address as the store. The necessity of the store queue is twofold:

- The floating-point unit cannot drive store data in time to write the data array during the proper pipeline stage. The store queue, therefore, provides the time to transfer the data from the execution units to the data cache.
- Memory cannot be modified until it is known that the store instruction will properly finish execution. If the store instruction is going to trap, say, because of a TLB fault, any architected state, such as memory, must not be changed.

The disposition of the store instruction is not known until the R stage of the pipeline, well after the data array is to be written. The store queue serves as a temporary buffer to hold pending store data. If a store that writes into the store queue subsequently traps, that store queue entry is merely invalidated. Also, by using a store queue, we are able to use a single bidirectional bus to transfer data between the execution units and the data cache. The store queue allows data to be transferred on the second half of the A pipeline stage for both load and store instructions, preventing conflicts between adjacent loads and stores in the instruction stream.

Semaphore Instructions. The data cache performs other memory operations besides load and store instructions. It handles semaphore instructions, which in the PA-RISC architecture require a memory location to be read while that location is simultaneously zeroed. In operation, a semaphore is quite similar to a store instruction with zeroed data, except that the semaphore read data is transferred on the second half of the A stage. In cases in which the semaphore is not present or modified in the data cache, the load and clear operation must be performed by the memory controller.

Flush and Purge Instructions. We must also execute flush instructions, which cause a given memory location to be cast out of the data cache. Related is the purge instruction, which at the most privileged level causes a memory location to be invalidated in the data cache with no cast out, even if the line is modified.

Reducing Miss Latency. Data cache misses are detected on the first half of the A stage of the pipeline. To reduce miss latency, the physical address being read from the data cache is forwarded to the memory controller before the data cache hit-or-miss disposition is known. This address is driven to the memory controller on the first half of the A stage. A "use address" signal is driven to the memory controller on the first half of the R stage if a cache miss occurs.

Copyin Transaction. A number of transaction types are supported between the CPU core and the memory controller. The most common type is a copyin transaction.

After receiving a copyin request, the memory controller returns the requested cache line, one double word at a time. As with instruction misses, the memory controller returns the data double word that was originally intended to be fetched first.

On load misses, when the critical double word arrives, it is sent directly to the execution units for posting into the register files. On integer load misses, the critical data is bypassed before error correction to reduce latency even further.

In the extremely rare event that the data contains an error, the CPU is prevented from using the bad data and forced to wait for corrected data. As each double word arrives from the memory controller, it is placed into a copyin buffer.

When all the data has arrived, the contents of the copyin buffer are written to the data cache data array in a single cycle. There are actually two copyin buffers to ensure that two data cache misses can be handled simultaneously.

Fig. 8 shows a block diagram of the copyin and copyout buffers.

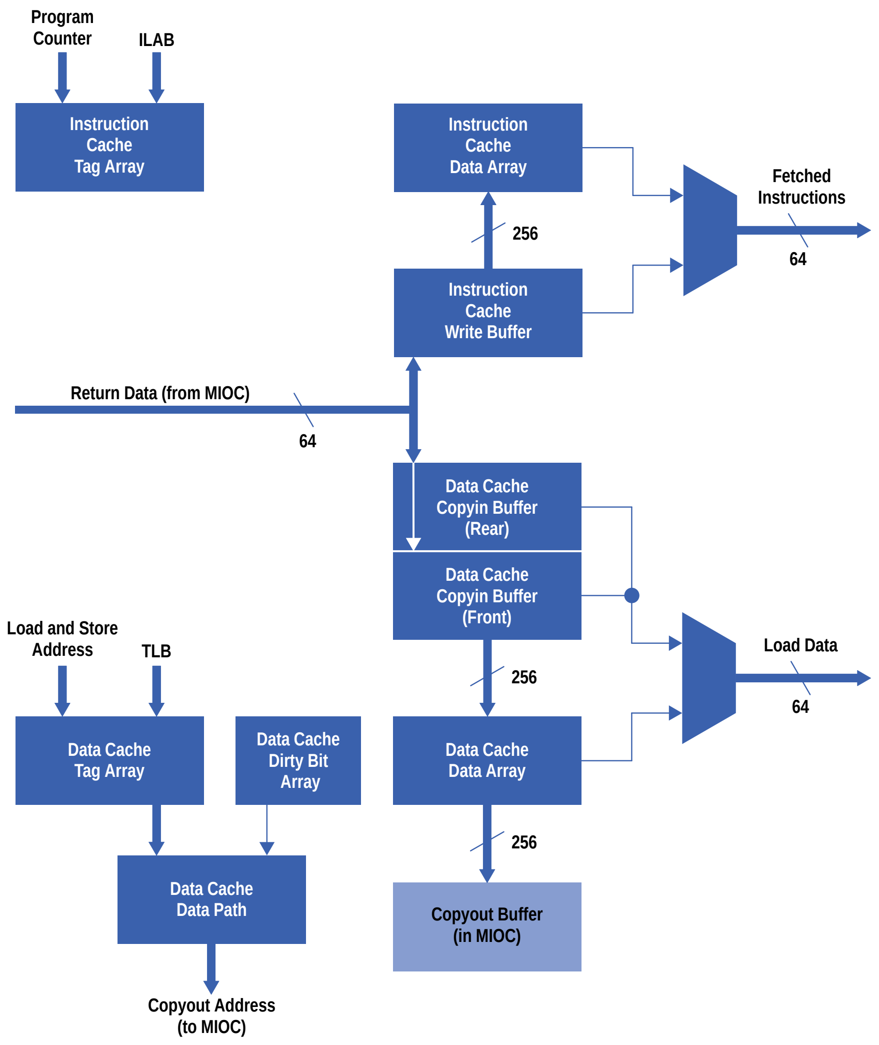

Program Counter
ILAB
Instruction Cache Tag Array
Instruction Cache Data Array
256
Instruction Cache Write Buffer
Fetched Instructions
64
Return Data (from MIOC)
64
Data Cache Copyin Buffer (Rear)
Data Cache Copyin Buffer (Front)
Load and Store Address
TLB
256
Load Data
64
Data Cache Tag Array
Data Cache Dirty Bit Array
Data Cache Data Array
256
Data Cache Data Path
Copyout Buffer (in MIOC)
Copyout Address (to MIOC)

Fig. 8. *A block diagram of the copyin and copyout buffers.*

Copyout Transaction. A data cache line can contain modified data requiring posting or writing back to memory when cast out. To this end, another transaction type is implemented—a copyout transaction. A copyout is necessary under two circumstances. The first case is when a data cache miss is detected and the existing cache line selected for replacement has been modified. This is the most common case.

The second case is when a flush instruction is executed and hits a modified line in the data cache. The data cache supplies both a physical address and 32 bytes of data on a copyout. The data cache uses the checkerboard organization, so the full cache line read for the copyout takes only one cycle.

Reducing Cache Miss Penalties. In the PA 7300LC, we have taken a number of steps to reduce the penalty caused by cache misses. As mentioned above, we have reduced cache miss latencies. We have also continued to adopt a “stall-on-use” load miss policy pioneered on earlier PA-RISC designs.[4] In this policy a load miss stalls the CPU pipeline only long enough to issue the copyin transaction and possibly a copyout transaction. In many cases, the delay lasts for only one cycle. The CPU will then only stall when the target register of the load instruction is subsequently referenced. If the critical data returns from memory fast enough, the pipeline will not stall at all.

Because memory data is not needed by the CPU on a store miss, the CPU only stalls once, again for only one cycle in many cases, to issue the copyin and copyout transactions.

A scoreboard keeps track of which words have been stored so that the copyin write will not overwrite more recent data. Since high-bandwidth writes to I/O space can be critical to graphics performance, under most circumstances the PA 7300LC will not stall on a store to I/O space. This optimization is possible because an I/O space access is guaranteed to miss the data cache, so there is no need to stall the CPU to perform a copyout read.

Cache Hints. The PA-RISC architecture defines cache hints to allow the programmer or compiler to communicate information that can be used by the hardware to improve performance.[5] We have implemented two of these hints on the PA 7300LC:

- Block copy store. Hints are used to indicate that software intends to write an entire cache line. In this case, there is no need to perform a main memory read on a cache miss. With this hint specified, upon detecting a store miss, the PA 7300LC simply zeros out a copyin buffer and continues without issuing a copyin transaction.
- Coherent operation semaphore hint. This optimization improves semaphore performance by not forcing the load and clear operation to the memory unit if the data is present in the cache.

TLB Access. All memory reference instructions are guaranteed access to the unified TLB containing both instruction and data translations, during the B and A stages of the pipeline. The TLB is fully associative and contains 96 page translations. The TLB receives a virtual data address on the first half of the B stage and drives a translated physical address on the first half of the A stage. This physical address goes to the data cache to perform hit comparison and to the memory controller in anticipation of a data cache miss.

In addition to containing 96 page entries, each of which maps to a 4K-byte page, the TLB also contains eight block entries used to map larger memory ranges. These block entries are managed by the operating system.

CPU Summary

Although the CPU core of the PA 7300LC is not dramatically different from its predecessors, several noteworthy features that improve performance and allow more cost-effective system designs include:

- A simple pipeline and a capable superscalar core that increased our operating frequency to 160 MHz.
- Substantial primary caches integrated directly onto the processor chip
- Most important, cache controllers that take advantage of integrated caches, resulting in features designed into the CPU core to increase the competitiveness of PA 7300LC-based systems.

Memory and I/O Controller Design

The memory and I/O controller (MIOC) is responsible for interfacing the CPU core to the memory and I/O subsystems. Integrating the MIOC on the same chip as the CPU core provides a tight coupling that results in outstanding memory and I/O performance. The memory controller includes a main memory controller and a controller for an optional second-level cache. The I/O controller interfaces the CPU core to HP's general system connect (GSC) I/O bus and handles direct memory access (DMA) requests from I/O devices.

CPU to MIOC Interface

The CPU core transmits four basic types of request to the MIOC:

- Copyins. These requests occur during first-level cache misses and are used by the CPU core to read a cache line from the memory subsystem.
- Copyouts. A copyout is a cache line from the CPU core that must be written to the memory subsystem because it was modified in the first-level cache by a store instruction. Copyouts are only issued when a modified cache line is replaced or flushed from the first-level cache.
- Uncached loads and stores. An uncached load or store request is a read or write to either memory or I/O for an amount of data that is less than a cache line.
- Load-and-clears. This request is an indivisible request to read a location and then clear it. This operation is needed to implement PA-RISC's semaphore mechanism. Requests that have addresses located in memory address space are processed by the memory controller, and all others are sent to the I/O controller.

The PA 7300LC has a four-entry copyout buffer. Copyouts are posted to memory as a background operation, allowing copyins to be processed before copyouts. New copyin requests are checked for conflict within the copyout buffer. If there is no conflict, the copyin is processed before all copyouts to help minimize load use stalls.

Second-Level Cache Control

Even though first-level caches on the PA 7300LC are relatively large for integrated caches, many applications have data sets that are too big to fit into them. The second-level cache (SLC) implemented for the PA 7300LC helps solve this problem. Logically, the SLC appears as a high-speed memory buffer; other than its performance improvement, it is transparent to software. The SLC is physically indexed, is write-through, has unified instructions and data, and is direct mapped.

The SLC becomes active after an access misses the first-level cache. The first-level cache miss indication becomes available after the TLB delivers the real address. As a result, there is little advantage to virtually indexing the SLC and real indexing avoids the aliasing problems associated with virtual caches.

Multiway Associative Cache Comparison. Multiway associative caches enjoy better hit rates because of fewer collisions. However, multiway caches are slower because of way selection, and for a given cache size, are much more expensive to implement with industry-standard components. For most applications, it is more advantageous to trade off size for ways of associativity.

Write-Back Cache Comparison. Write-back caches* generally have better performance than write-through caches. However, sharing the data bus with main memory alters this situation. If the SLC were write-back, lines copied out of the SLC would have to be read into the PA 7300LC, the error-correcting code (ECC) would have to be computed, and the line would have to be written back to main memory. This operation would be quite expensive in terms of bus bandwidth. Instead, dirty lines cast out by the first-level cache are written to the SLC and to main memory simultaneously.

Any valid line in the SLC always has the same data as its corresponding location in main memory. Writing simultaneously to main memory and to the SLC is slightly slower than simply writing to the SLC SRAM, but produces a good performance and complexity trade-off when compared to a write-back design.

DMA Interface. DMA reads and writes from I/O devices are typically sequential and do not exhibit the access locality patterns typical of CPU traffic. Entering DMA traffic into the SLC tends to pollute the SLC with ineffective entries. Instead, buffering and prefetching inside the DMA interface are better ways of improving DMA performance. To maintain consistency, an SLC check cycle is run for DMA writes, and if it hits, the line is marked invalid. DMA write data is always written to main memory and DMA reads are always satisfied from main memory. Because of the write-through design of the SLC described above, data in the SLC never becomes stale.

SRAM Components. The PA 7300LC is optimized for both price and performance. Relatively early in the design process, it became necessary to select the static random access memory (SRAM) components used to build the SLC. SRAM components are frequently used in cache construction because they offer high speed with moderate cost and capacities. Given the relatively long design cycles necessary to produce a complex microprocessor and the uncertainties of the semiconductor marketplace, it was impossible to predict which components would be most attractive from a price and performance perspective when the PA 7300LC entered full production. Instead of selecting a single component, the decision was made to support a broad range of SRAM types. This allowed component selection to be made late in the development cycle and even upgraded at some point during the production phase.

Second-Level Cache Size. Most popular computer benchmark programs have relatively small working sets and are not particularly sensitive to the performance of the memory system beyond the first-level cache. On the other hand, application programs have widely variable working set sizes. Some are small and fit well in the first-level cache and some exhibit little reference locality and don't fit in any reasonably sized cache. Hence, no single SLC size is appropriate. The PA7300LC SLC controller supports cache sizes ranging from 256K bytes up to 64M bytes. Although a 64M-byte SLC is expensive, it might be cost-effective for some applications.

The SLC data array width can be programmed to either 64 or 128 bits plus optional ECC. However, the width must match the width of main memory.

Memory Arrays. The SLC consists of two memory arrays: the data array and the tag array. The data array shares the data bus with main memory. As an option, ECC bits can be added to the data array, and the full single-bit correct and double-bit detect error control invoked for SLC reads. The tag array includes a single optional parity bit. If parity is enabled and bad parity is detected on a tag access, an SLC miss is signaled, the failing tag and address are logged, and a machine check is signaled.

Main Memory Control

DRAMs. Dynamic random access memory (DRAM) technology is used to construct main memory because of its high density, low cost, and reasonable performance levels. The main memory controller supports industry-standard DRAMs from 4M-bit to 256M-bit capacities. Systems can have up to 16 slots and total memory can be up to 3.75G bytes, the maximum possible with the PA-RISC 1.1 architecture.

Data Bus Width. Data bus width can be either 64 or 128 bits plus optional ECC. The 128-bit data bus width significantly improves memory performance. The 64-bit option supports lower-cost systems.

* In a write-back cache design (also called copy-back), data is written only to the cache on a write, and is not written to main memory until the cache line is invalidated. In a write-through cache design, data is written to both the cache and main memory on a write.

Main Memory Controller. The PA 7300LC main memory controller is very flexible and is able to support most types of asynchronous DRAMs. The controller is intentionally not SIMM/DIMM (single or double inline memory module) specific. This allows use of the PA 7300LC in a wide variety of system configurations. The main memory can support extended data out (EDO) DRAMs, which are similar to other DRAMs but use a slightly modified protocol that pipelines the column access.

Fig. 9 shows the timing diagrams of read accesses, emphasizing the improved data bandwidth of EDO DRAMs compared to standard page-mode DRAMs.

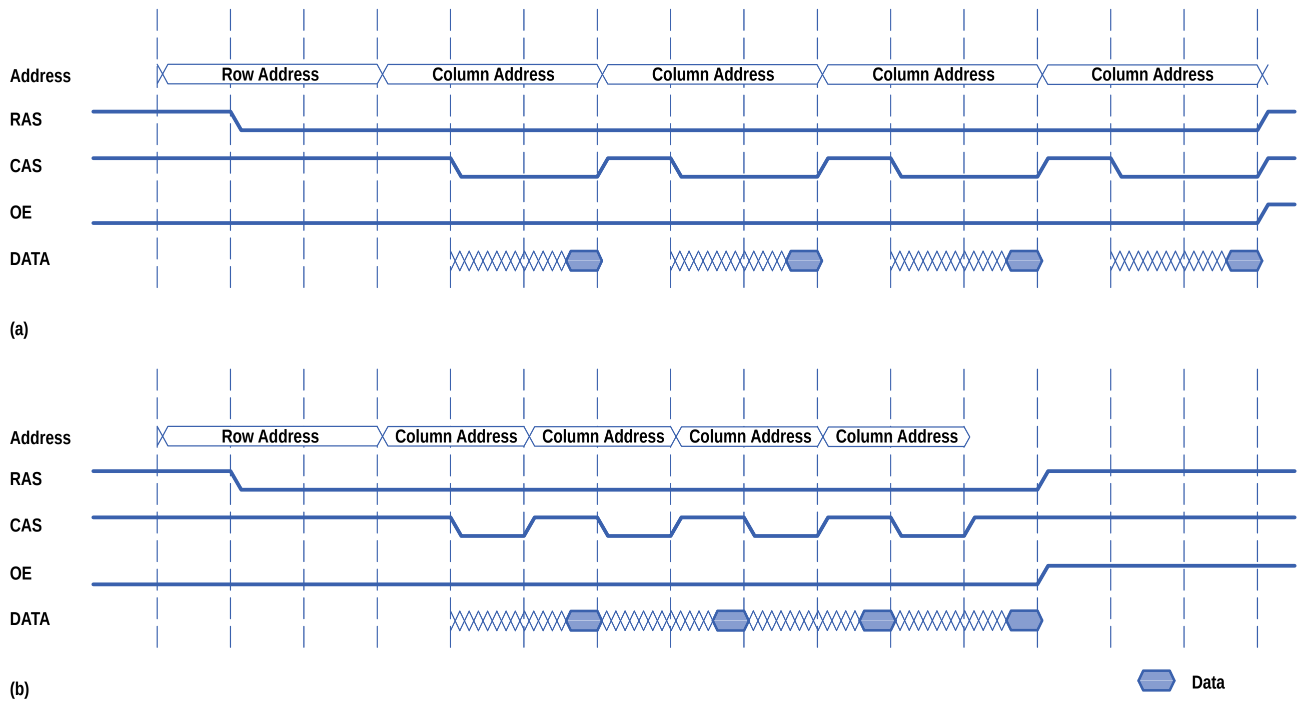


Fig. 9. *DRAM timing diagrams. (a) Page mode read. (b) Extended data out (EDO) mode read.*

Error-Correcting Code. The state of DRAM memory cells is susceptible to corruption from incident energetic atomic particles. Because of this, the PA 7300LC main memory controller optionally generates and checks an error-correcting code. The code is generated over a 64-bit data word. Any single-bit error within the 64-bit data word can be corrected. All double-bit errors and all three- or four-bit errors within an aligned nibble can be detected. The aligned nibble capability is useful since memory systems are typically built with four-bit-wide DRAMs. The nibble mode capability allows detection of the catastrophic failure of a single four-bit-wide DRAM. Whenever an error is detected, data and address logging registers are activated to support efficient fault isolation and rapid field repair.

Shared SLC and Main Memory Data Bus

From a cost perspective, it was desirable to share the large data buses needed for the SLC and main memory, thereby lowering the pin count of the PA 7300LC. However, sharing the large load from main memory DRAM cards would have significantly impacted the speed of SLC operations. The solution to this problem resulted in using an FET switch to isolate the main memory load from the SLC bus when the SLC is driving the bus, but to allow the bus to be shared when main memory is being accessed (see Fig. 10). The FET switch is a relatively inexpensive industry-standard part, which has a propagation delay of less than 1 ns when in the on state.

FET Switch. The FET switch also enabled us to connect the PA 7300LC to legacy 5-volt DRAM cards. The PA 7300LC operates at 3.3 volts and is not tolerant of the 5-volt logic swing of many existing DRAM cards. Biasing the gate of the FET switch to a voltage lower than 5 volts effectively limits the voltage swing from DRAM cards to 3.3 volts when seen by the PA 7300LC.

Chip Layout Challenges

Although the MIOC is a small part of the PA 7300LC, it controls nearly all of the I/O pins. Because the pins are located at the chip perimeter, long signal routes from the MIOC to some pins are unavoidable. Separating the MIOC into several blocks that could be placed near the chip perimeter and controlled remotely helped manage this problem. In particular, the data flow across the shared SLC and main memory data bus is completely predictable (because there are no slave handshakes from the memories), making the memory data interface the ideal block to be controlled from the other side of the chip.

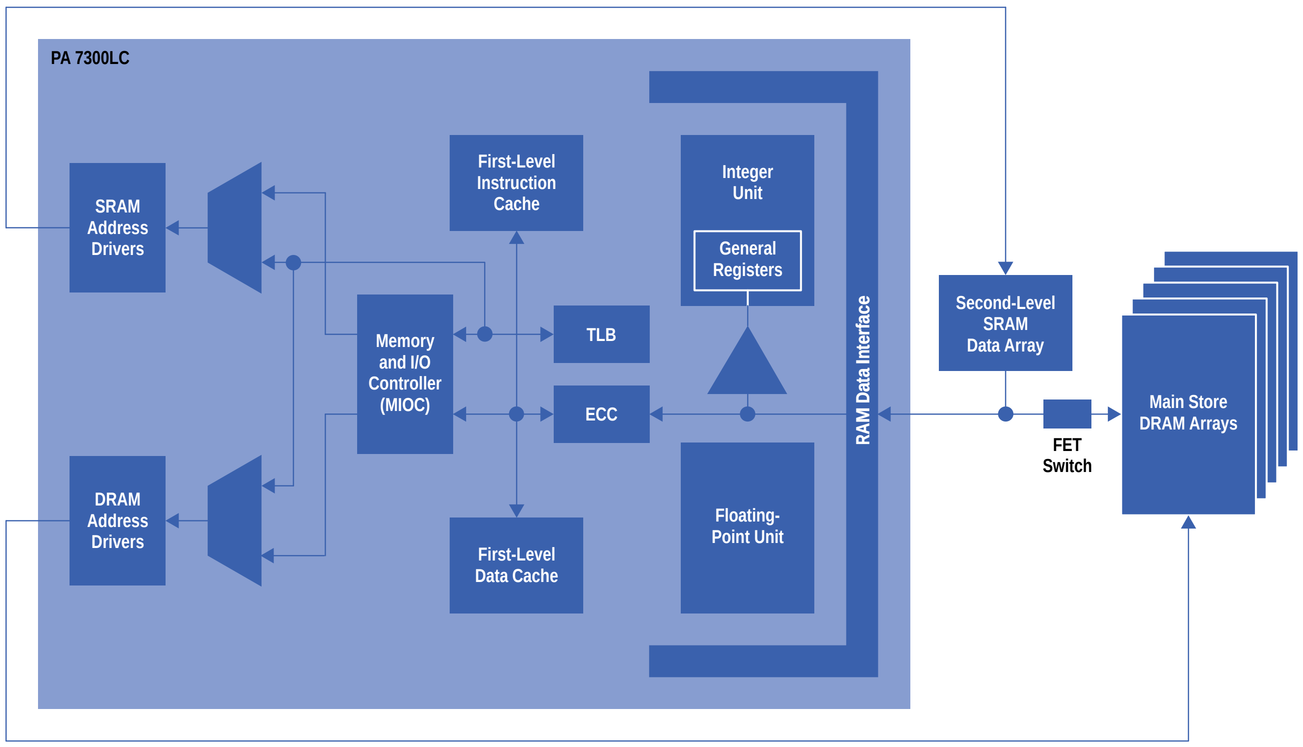


Fig. 10. *PA 7300LC block diagram showing the position of the FET switch.*

Cache Miss Data Flow

The MIOC is highly optimized for satisfying CPU cache misses. Although DMA transaction processing is handled efficiently, system performance is more sensitive to CPU cache miss performance than DMA performance.

When idle, the SLC and main memory controllers pass through physical addresses that are coming directly from the TLB and going to the SLC and main memory address pads. On the cycle following each address, the CPU core indicates whether that address resulted in a miss in the first-level cache. If a miss occurred, then an access is initiated and a cycle is saved by having passed along the physical addresses to the SLC and main memory.

For copyins, the SLC begins an access. The tag and data array are accessed in parallel. If there is an SLC hit, then data is returned to the processor.

On an SLC miss, the SLC data array data drivers are disabled, the FET switch is closed, and control is transferred to the main memory controller.

When a transaction is received by the main memory controller, it endeavors to activate the correct DRAM page. This may be as simple as issuing a row address strobe (RAS) with the proper row address, or may require deasserting RAS, precharge, and a new RAS. The memory controller sequences up to the point at which it is ready to issue a column address strobe (CAS) command, waits there until the SLC misses, and switches control over to complete the CAS command. However, if the SLC hits, it will wait for the next transaction and start the cycle again. Performance is improved by starting the DRAM access in parallel with the SLC access.

In the case of an SLC miss, once the main memory controller has control, it issues the proper number of CAS cycles to read the data. As the data passes the SLC, it is latched into the SLC data array. At the end of the cycle, the FET switch is opened, the SLC drivers are enabled, and the next transaction is processed.

Reducing Low-Miss Latencies. Much of the work described above concerns reducing miss latencies. This is important because even though the PA 7300LC CPU core has a non-blocking cache, load use stalls still develop quickly for many instruction sequences. Low-miss latencies minimize the impact of these stalls, which results in better overall performance. At CPU clock rates of 160 MHz, the PA 7300LC, as seen by the CPU pipeline, is capable of SLC hit latencies of three cycles with industry-standard 6-ns asynchronous SRAM. Main memory latencies can be as low as 13 cycles with 50-ns DRAM. Many single-cycle latency reductions have been implemented in the PA 7300LC; each by itself would not have much impact on overall memory access latency, but taken together, they make a significant difference.

I/O Interface

The PA 7300LC contains interface logic that allows direct connection to HP's high-speed GSC I/O bus. This interface processes I/O requests from the CPU core and DMA requests from GSC I/O bus devices.

Programmed I/O. Programmed I/O allows load and store instructions from the CPU core to communicate with the I/O subsystem. From a performance perspective, programmed I/O writes to graphics devices are important for many workstation applications. The improvements made for graphics performance in the PA 7300LC are described later in this article.

DMA Interface Controller. The DMA interface controller is designed to minimize main memory controller traffic and to reduce DMA read latency. The DMA interface controller employs three 32-byte line buffers. When servicing any DMA read, the controller requests 32 bytes from main memory and puts the data into one of the buffers. DMA requests on the GSC bus may be 4, 8, 16, or 32 bytes long. Since most DMA requests are to sequential addresses, requests less than 32 bytes can probably be satisfied from data contained in the buffer without issuing another request to the main memory controller. The DMA controller is also able to prefetch the next sequential line of information to increase the chances that DMA read requests are serviced from the DMA buffers.

GSC Write Requests. Writes are collected by the DMA hardware and passed on to the main memory controller. GSC write requests of 32 bytes are sent directly to the controller, but when possible, smaller-sized writes are collected into 32-byte chunks by the DMA controller to allow the main memory controller to access memory more efficiently.

Improvements for Graphics Applications

Graphics performance depends on many aspects of the system design. In addition, graphics workloads are sensitive to the system architecture. For the PA 7300LC, we chose to optimize the design for engineering graphics, where the typical workload involves rendering an object to the display device.

From a high-level point of view, the process of rendering an object can be divided into three steps:

1. Traversing the display list that describes the object
2. Clipping, scaling, and rotating the object with the current viewpoint
3. Transforming the object from primitive elements, such as polygons, into pixels on the screen.

This process can be partitioned in different ways. With today's powerful CPUs, the most cost-effective method is to store the display list in the computer system's main memory. The host CPU performs the display list traversal and the clipping, scaling, and rotation steps, and then passes primitives to dedicated graphics hardware for conversion into onscreen pixels.

Graphics Requirements. Several different models, including specialized CPU instructions and DMA engines, have been used to extract data to be rendered from main memory. While these approaches work, they incur the undesirable cost of specialized driver software that doesn't port well between processor generations. Starting with the PA 7100LC, the philosophy has been to support the graphics requirements within the existing architecture as much as possible. For example, the PA-RISC architecture defines a set of 32-bit integer unit general registers and another set of 64-bit floating-point unit general registers. Loads and stores from either set can be made to memory space, but only integer register loads and stores were architecturally defined to I/O space.

Starting with the PA 7100LC, floating-point register loads and stores to I/O space have been implemented. This has yielded improved performance because a single load or store can now move 64 bits and because more registers are available for operations that communicate with I/O space.

In contrast with specialized operations, extensions within the architecture are generally applicable and carry forward into future generations. These optimizations can also be used to benefit workloads other than graphics.

Graphics Optimizations. Several of the optimizations made in the PA 7300LC to further improve graphics performance include:

- A large I/O store buffer
- A relaxation of the load and store ordering rules
- The elimination of a CPU hang cycle previously needed for I/O stores
- Improvements to the GSC I/O bus protocol.

The structure of industry-standard graphics libraries leads to bursty graphics I/O traffic. The bursts are of many different sizes, but the most common burst is a write of 26 words. The PA 7300LC CPU core-to-I/O interface implements a large write buffer and can accept up to 19 double-word writes without stalling the CPU pipeline. This allows the CPU core to burst up to 19 double-word writes to the I/O subsystem, and then continue with its next task while the I/O interface is sending this data out to the graphics hardware.

Graphics Ordering. PA-RISC is a strongly ordered architecture. Strongly ordered means that all elements of the system must have a consistent view of system operations. In the case of graphics performance, this means that all buffered I/O stores must be observed by the graphics device before the CPU can access a subsequent piece of data in main memory. Hence, an I/O store and a following memory read are serialized. A loophole to the ordering requirement was created for graphics. I/O stores within a programmable address range are allowed to be out-of-order with respect to the memory accesses. The graphics software takes responsibility for ordering when necessary.

Hang Cycle. Previous PA-RISC processors always incurred a minimum of one hang cycle for an I/O store. Extra logic was added to the data cache controller on the CPU core to eliminate this hang cycle.

Graphics Transfer Size. HP's high-speed GSC bus is used to connect graphics adapters to the PA 7300LC. The CPU sends data to the graphics device with I/O stores. In the PA-RISC architecture, I/O stores are 64 bits or less. The GSC is a 32-bit multiplexed address and data bus. Stores of 64 bits turn into an address cycle followed by two data cycles. At best the payload can be only two thirds of the maximum bus bandwidth. As mentioned above, the average transfer size to graphics is 26 words. Since these transfers are sequential, sending an address with every two words is unnecessary. Some form of address suppression or clustering of sequential writes was desired. Thus, the write-variable transaction was created.

Write-Variable Transactions. A new write-variable transaction type was created for the GSC bus. Write-variable transactions consist of an address and from one to eight data cycles. Since the PA 7300LC must be compatible with existing cards that do not implement the write-variable cycle type, the PA 7300LC only generates them in configurable address spaces.

With this protocol, the I/O controller blindly issues write-variable transactions for enabled I/O address regions. Starting with the initial write, as each write is retired from the I/O write queue, the I/O controller performs a sequentiality check on the next transaction in the queue. The process repeats for up to eight GSC data cycles. Maximum performance is achieved by allowing the I/O controller to begin issuing the write when the first piece of data becomes available.

The length of the transaction is limited to eight data cycles. Choosing eight data cycles is a good compromise between flow control issues and amortizing address cycle overhead with payload. The write-variable enhancement increased maximum CPU-to-graphics bandwidth from two thirds of the GSC raw bandwidth to 8/9 of the raw bandwidth. The PA 7300LC can easily saturate the GSC bus at 142 Mbytes per second compared with the 50 Mbytes per second achieved by the PA 7100LC with careful coding.

MIOC Summary. The MIOC implemented a number of features that improve system performance while keeping costs low, including:

- The second-level cache and main memory controllers are optimized to reduce the latency of copyin requests from the CPU core.
- The I/O controller improves graphics bandwidth and supports efficient DMA accesses through the use of buffers and prefetching.
- The MIOC is designed to be flexible, supporting a range of second-level cache sizes, a variety of industry-standard memory components, two different memory widths, and an optional error correction scheme.

Conclusion

The PA 7300LC design builds on the success of past processor designs and offers significant improvements in key areas. It features a superscalar CPU core, a large, efficient on-chip cache organization, tightly coupled second-level cache and main memory controllers, and bandwidth improvements for graphics. These features combined with frequency increases, extensive configurability, and high chip quality make the PA 7300LC attractive for a wide range of computer systems.

Acknowledgments

A large number of people were responsible for the successful design of the PA 7300LC, especially the design teams from the Engineering Systems Lab in Fort Collins and from the Integrated Circuits Business Division's Fort Collins Design Center. Many important contributions were also made by individuals from the Fort Collins Systems Lab, the Systems Performance Lab in Cupertino, the Computer Technology Lab in Cupertino, and other organizations within HP.

References

1. P. Knebel, et al, "HP's PA 7100LC: A Low-Cost Superscalar PA-RISC Processor," *Proceedings of IEEE Compcon*, February 1993, pp. 441-447.
2. S. Undy, et al, "A VLSI Chipset for Graphics and Multimedia Workstations," *IEEE Micro*, Vol. 14, no. 2, April 1994, pp. 10-22.
3. G. Kurpanek, et al, "PA 7200: A PA-RISC Processor with Integrated High-Performance MP Bus Interface," *Proceedings of IEEE Compcon*, February 1994, pp. 375-382.
4. E. DeLano, et al, "A High-Speed Superscalar PA-RISC Processor," *Proceedings of IEEE Compcon*, February 1992, pp. 116-121.
5. R. Lee, "Precision Architecture," *IEEE Computer*, Vol. 22, no. 1, January 1989, pp 78-91.

6. R. Lee, J. Beck, L. Lamb, and K. Severson, “Real-Time Software MPEG Video Decoder on Multimedia-Enhanced PA 7100LC Processors,” *Hewlett-Packard Journal*, Vol. 46, no. 2, April 1995, pp. 60-68.
7. M. Bass, T. Blanchard, D. Josephson, D. Weir, and D. Halperin, “Design Methodologies for the PA 7100LC Microprocessor,” *Hewlett-Packard Journal*, Vol. 46, no. 2, April 1995, pp. 23-35.

▶ Go to Subarticle 7a
▶ Go to Next Article
▶ Go to Journal Home Page