PA-RISC 2.0 Architecture

This document is an excerpt from PA-RISC 2.0 ARCHITECTURE, by Gerry Kane, published by Prentice Hall PTR, isbn 0-13-182734-0. Print copies of the book can be ordered at www.hp.com/hpbooks.

This document contains:
Appendix F: TLB and Cache Control
This appendix provides detailed information relating to operation of the TLB (Translation Lookaside Buffer) and cache in a PA-RISC system. It describes how software can control the contents of the TLB under various conditions such as TLB miss handling and also specifies the operation requirements of the TLB.

The appendix also describes the responsibilities of system software in regards to handling address aliasing as well as cache move-in rules. Finally, a brief summary of the guidelines relating to coherency in multiprocessor systems is provided.

TLB Control

TLBs function as buffers for the most frequently used address translations. Terms used to describe TLBs are given below.

Entry
- The term entry refers to a translation which is present in the TLB. Entries are visible to software through references (such as load, store, and semaphore instructions, access rights probes, and the LOAD PHYSICAL ADDRESS instruction).

Slot
- Hardware resources in the TLB which hold entries are referred to as slots.

Remove
- An entry is removed when some action causes it to be inaccessible to software. Insertion of translations into the TLB, for example, causes other entries to be removed. Entries can be explicitly removed by purging them.

NOTE
- From a hardware perspective, a translation is removed when it is displaced by the insertion of another translation. If each slot has an associated valid bit which causes that slot not to participate in TLB lookup, then a translation is removed if the slot that contains it is marked invalid. This would be one way of implementing TLB purges.

Several mechanisms can be used by software to remove a specific translation from the ITLB or DTLB. First, when a new translation is inserted into the TLB, any old translations which overlap the virtual address range of the new translation are removed. Second, a specific virtual address and page size may be used to purge (remove) the associated translations from the TLB. The PURGE INSTRUCTION TLB and PURGE DATA TLB instructions perform this function. These instructions also optionally cause the translation to be removed from the TLBs of other processors in a multiprocessor system.

Translations may also be removed from the TLB using the PURGE INSTRUCTION TLB ENTRY and the PURGE DATA TLB ENTRY instructions. These purge some machine-specific number of entries in the TLB without regard for the translation. These instructions are used by system software to clear the entire instruction or data TLB.

Because the TLB is managed by a mixture of hardware and software mechanisms, software may not, in
general, rely on the existence of translations in the TLB and hardware may, in general, remove TLB entries at any time, provided that forward progress is assured. There are limited situations, however, in which software may rely on a translation existing in the TLB. This means that software may make virtual accesses using this relied-upon translation, and no TLB miss fault will occur. Hardware is required to retain this TLB entry as long as the constraints of the limited situation are met. These situations are described in “TLB Operation Requirements” on page F-3.

NOTE
As a result, the following hardware actions are allowed, except in the defined limited situations.

- A TLB miss fault may be taken even though the translation exists in the TLB.
- In the event of a TLB error, one or more entries may be removed.

Note also that software may not rely on the existence of any translations in the TLB immediately after any group 1, 2, or 4 interruption.

Software TLB Miss Handling
In order to insure forward progress, some restrictions are placed on software which performs TLB miss handling.

For instruction TLB miss handling, the following restrictions apply:
- Software can insert multiple instruction address translations into the ITLB, provided that the translation which caused the trap is inserted last.
- Software must not execute a purge TLB instruction using the virtual address corresponding to the data address translation needed for the execution of the trapping instruction.
- Software must not insert translations into the DTLB.

For non-access instruction TLB miss handling, the following restrictions apply:
- Software can only insert into the ITLB up to all of the eight translations for the page group. The translation which caused the trap must be inserted last.
- Software must not execute a purge TLB instruction using the virtual address corresponding to the data address translation needed for the execution of the trapping instruction.
- Software must not insert translations into the DTLB.

For data TLB miss handling and non-access data TLB miss handling, the following restrictions apply:
- Software can only insert into the DTLB up to all of the eight translations for the page group. The translation which caused the trap must be inserted last.
- Software must not execute a purge TLB instruction using the virtual address corresponding to the instruction address translation needed for the execution of the trapping instruction.

The following restrictions apply to all four TLB miss handlers:
• Software must not make any virtual references.
• Software must not execute any PURGE DATA TLB ENTRY or PURGE INSTRUCTION TLB ENTRY instructions.

Hardware TLB Miss Handling

The default endian bit (see “Byte Ordering (Big Endian/Little Endian)” on page 2-19) determines how data from the hardware-visible page table is interpreted by the hardware TLB miss handler, if implemented. If the default endian bit is 0, the hardware-visible page table entries are loaded as doublewords in big-endian format; if the default endian bit is 1, the entries are loaded as doublewords in little-endian format.

If a data prefetch instruction (described in “Data Prefetch Instructions” on page 6-11) references an address which misses in the data TLB, the hardware TLB miss handler should not be invoked.

TLB Operation Requirements

Software may rely on the existence of particular translations in the TLB only in certain situations. The following describes the situations in which software may rely upon the fact that a specific translation will continue to exist in the TLB. In these situations, software may make virtual accesses using the relied-upon translation, and no TLB miss fault will occur (including non-access TLB miss faults).

1. When an instruction takes one of the following interruptions, the associated data address translation will remain in the DTLB, and is termed the relied-upon translation.

<table>
<thead>
<tr>
<th>Intr. No.</th>
<th>Interruption</th>
</tr>
</thead>
<tbody>
<tr>
<td>18</td>
<td>data memory protection/unaligned data reference trap</td>
</tr>
<tr>
<td>19</td>
<td>data memory break trap</td>
</tr>
<tr>
<td>20</td>
<td>TLB dirty bit trap</td>
</tr>
<tr>
<td>21</td>
<td>page reference trap</td>
</tr>
<tr>
<td>22</td>
<td>assist emulation trap</td>
</tr>
<tr>
<td>26</td>
<td>data memory access rights trap</td>
</tr>
<tr>
<td>27</td>
<td>data memory protection ID trap</td>
</tr>
<tr>
<td>28</td>
<td>unaligned data reference trap</td>
</tr>
</tbody>
</table>

The translation will continue to remain in the DTLB, meaning no data TLB miss fault will occur on virtual data accesses which use this translation, for as long as software meets the following constraints:

• No virtual data references are made to pages other than the page corresponding to the relied-upon translation.
• The execution stream does not contain nullified instructions which, had they not been nullified, would have made virtual data references to pages other than the page corresponding to the relied-upon translation.
• No memory management instructions other than LPA are executed. (See “Memory
Management Instructions (Mem_Mgmt)” on page C-5 for a list of memory management instructions.)

- No purge TLB instructions which would purge the relied-upon translation are executed by other processors in a multiprocessor system.
- No virtual instruction references are made.
- No DIAGNOSE instructions are executed.
- No attempt is made to execute undefined instructions.

---

**Programming Note**

Software may rely upon this translation in order to improve performance in handling the above-mentioned traps. For example, the absolute address which corresponds to the virtual address used in the trapping instruction can be determined by using this code sequence:

```
LPA x(s,b),t
```

Because no TLB miss fault can occur, the interruption handler need not incur the overhead of making itself interruptible.

---

2. If the PSW Q-bit is 1, and is set to 0 by a RESET SYSTEM MASK or MOVE TO SYSTEM MASK instruction, the instruction address translation used to fetch the RSM or MTSM instruction will continue to remain in the ITLB, and is termed the relied-upon translation. No instruction TLB miss fault will occur on virtual instruction accesses which use this translation for as long as software meets the following constraints:

- The RSM or MTSM instruction which sets the PSW Q-bit to 0 (the clearing RSM or MTSM) is preceded by another RSM, SSM, or MTSM instruction which does not affect the PSW Q-bit, and which appears at least 8 instructions prior.
- The instructions between the initial RSM, SSM, or MTSM instruction and the clearing RSM or MTSM do not include any memory management instructions, virtual data references, or instruction references to pages other than the code page containing the clearing RSM or MTSM instruction.
- The clearing RSM or MTSM instruction is not within 8 instructions of a page boundary.
- No virtual data references are made.
- The execution stream does not contain nullified instructions which would have made virtual data references had they not been nullified.
- No FLUSH INSTRUCTION CACHE instructions are executed with the PSW D-bit equal to 1.
- No memory management instructions are executed. (See Appendix C, “Operation Codes” for a list of memory management instructions.)
- No purge TLB instructions which would purge the relied-upon translation are executed by other processors in a multiprocessor system.
• No instruction references are made to pages other than the code page containing the clearing RSM or MTSM instruction.
• No DIAGNOSE instructions are executed.
• No undefined instructions are attempted to be executed.
• No instructions are executed which are followed, within 8 words, by a branch instruction, a memory management instruction, a DIAGNOSE instruction, an undefined instruction, or a page boundary.

Programming Note
Software may rely upon this instruction translation in order to improve performance in process dispatch. For example, in this code sequence:

```
SSM 0,gr0 ; initial RSM, SSM or MTSM
LDW ; set up process state
. ; must be at least 7 instructions
. ; between the system mask instructions
.
LDW RSM 8,gr0 ; set PSW Q-bit to 0
MTCTL reg1,cr20; set up IIASQ
MTCTL reg2,cr20
MTCTL reg3,cr21; set up IIAOQ
MTCTL reg4,cr21
LDW ; set up last of process state
.
.
LDW ; dispatch process
```

Because no TLB miss fault can occur, the interruption handler need not incur the overhead of disabling code translation just prior to process dispatch. Note that the LDW instructions in this sequence must use absolute addresses. (Use absolute loads, or do these with the PSW D-bit equal to 0.)

Address Aliasing
Normally, a virtual address does not translate to two different absolute addresses. It is the responsibility of memory management software to avoid the ambiguity such occurrences would create.

Caches are required to permit a physical memory location to be accessed by both an absolute and a virtual address when the virtual address is equal to the absolute address. Such a virtual address is said to be equivalently-mapped. For equivalently-mapped addresses in the memory address space, note that since the upper 2 bits of the offset are not used in forming the absolute address, these bits need not
match the corresponding bits in the virtual address for the two addresses to be equivalently-mapped.

The instruction and data caches are required to detect that the same physical memory location is being accessed by two virtual addresses that satisfy all the following requirements:

1. The two virtual addresses map to the same absolute address.
2. Offset bits 44 through 63 are the same in both virtual addresses.
3. If the use of space bits in generating the cache index is enabled, the two virtual addresses have the same values for the following space identifier bits: 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, and 47.

Processors must provide an implementation-dependent mechanism to enable/disable the use of space bits in generating the cache index.

**NOTE**

Implementations are encouraged to provide a mechanism to enable/disable the three contiguous groups of 4 space bits (bits 36 through 39, bits 40 through 43, and bits 44 through 47) independently.

When space bits are enabled for use in generating the cache index, for each space bit that participates, the corresponding offset bit which gets ORed with it in the generation of virtual addresses is always forced by hardware to 0 for address generation purposes.

These rules provide offset aliasing on 16 Mbyte boundaries, with optional support for offset aliasing on smaller power of two sized boundaries, and either restricted or unlimited space aliasing.

Two virtual addresses that satisfy all of the above requirements are called equivalent aliases of each other. Virtual addresses that satisfy rule 1 but violate rule 2 or 3 are non-equivalent aliases, and are more restricted in their use. For non-equivalent aliases, read-only aliasing is supported with minimal restrictions, and many-reader/one-writer aliasing is supported with more significant restrictions.

Generally, if system software must use multiple addresses for the same data, these addresses are equivalent aliases (or are an absolute address and an equivalently-mapped virtual address). This description of non-equivalent aliasing, and the restrictions on software only apply in rare situations when non-equivalent aliasing is necessary.

For the purposes of supporting non-equivalent aliasing, a read-only translation is defined as one where the TLB and page table both meet at least one of the following conditions:

- The page type in the access rights field is 0, 2, 4, 5, 6, or 7. (See “Access Control” on page 3-11.)
- The D-bit (dirty bit) is 0.

A translation not meeting this requirement is termed a write-capable translation.

Software is allowed to have any number of read-only non-equivalently aliased translations to a physical page, as long as there are no other translations to the page. This is referred to as read-only non-equivalent aliasing.

Before a write-capable translation is enabled, all non-equivalently-aliased translations must be removed.
from the page table and purged from the TLB. (Note that the caches are not required to be flushed at this time.) The write-capable translation may then be used to read and/or modify data on that page. Before any non-equivalent aliased translation is re-enabled, the virtual address range for the writable page (the entire page) must be flushed from the cache, and the write-capable translation removed from the page table and purged from the TLB. If an old read-only translation is re-enabled, or a translation is enabled that is equivalently-aliased to an old translation, the virtual address range for the re-enabled translation must be flushed from the cache before accesses are made to the page. (This flushing is only required if the re-enabled virtual page has not been flushed since it was last accessed.) This is referred to as many-reader/one-writer non-equivalent aliasing.

Absolute read accesses can be made to a page which is mapped with a non-equivalently-mapped read-only translation, as long as the absolute address range accessed is flushed before enabling any write-capable translation. Since absolute accesses do not cause prefetching, it is not necessary to flush the entire page - only the accessed range need be flushed.

All other uses of non-equivalent aliasing (including simultaneously enabling multiple non-equivalently aliased translations where one or more allow for write access) are prohibited, and can cause machine checks or silent data corruption, including data corruption of unrelated memory on unrelated pages. It is the responsibility of privileged software to avoid non-equivalent aliasing, except as described above. This requires flushing the affected address range from the caches prior to any of the following:

- Changing the address mapping in the TLBs.
- Making an absolute access to a location which might reside in the caches as a result of an access by a virtual address that was not equivalently mapped.
- Making a virtual access to a location which might reside in the caches as a result of an access by its absolute address that was not equivalently mapped.
- Making a virtual access to a location which may reside in the caches as a result of an access by another virtual address that was not equivalently aliased.

NOTE

The restrictions on non-equivalent aliases are necessary to allow the design of high-performance caches and memory interconnect, including multi-level caches (including victim or miss caches) and directory-based coherency structures. Coherency schemes are greatly simplified by allowing the assumption that there is at most a single private copy of a physical line at any time. The read-only translation informs hardware to request a shared or public copy of the line.

Cache Move-in Restrictions

Data and instructions from memory can be brought into cache only under certain circumstances. Two different schemes are used for controlling cacheability – one for virtual accesses and one for absolute accesses.
Virtual Accesses

For virtual accesses, cache move-in is controlled by a mapping-based approach. Generally, if a translation exists in the TLB that permits access, the memory on that page may be brought into the cache. The access information in the TLB entry must meet a simple check, however, to permit move-in. This is required in order to allow for translations which can be used for cache flushing purposes, but which do not enable move-in. This is described in detail in following sections.

Since TLB miss handling may insert into the TLB any entry in the page table for which the R-bit (Reference bit) is 1, system software should consider the page table as the main tool for controlling cacheability of memory.

Absolute Accesses

For absolute accesses, cache move-in is controlled by a reference-based approach. Generally, only instructions and data referenced by executed instructions may be brought into the cache. The definition of “reference” is somewhat loose to permit instruction prefetching and instruction pipelining.

No data reference may cause a move-in to the instruction cache and no instruction reference may cause a move-in to the data cache. This means that the execution of a FLUSH DATA CACHE or PURGE DATA CACHE instruction guarantees that the addressed line, if it has been referenced as data but not as instructions, is no longer present in the cache system. Similarly, only a FLUSH INSTRUCTION CACHE instruction is required to guarantee that a line which has been referenced as instructions but not as data, is no longer present in the cache system. The actions which constitute a reference are described in “Data Cache Move-In” on page F-8 and “Instruction Cache Move-In” on page F-9.

If implemented, the U (Uncacheable) bit is found in the data TLB entry associated with a page. Whether or not the U-bit is implemented, whether the memory reference is virtual or absolute, and whether the reference is made from a page in the memory or I/O address spaces, determine if the reference may be moved into the data cache. The detailed rules for moving references into the data cache are specified in “Data Cache Move-In” on page F-8.

Data Cache Move-In

For virtual accesses, cache move-in is permitted only if there is a translation for the virtual address which meets both of these conditions:

- The page type field in the access rights for the entry contains a value in the range 0 to 3 (access rights allow read access).
- The T-bit in the entry is 0.

For absolute accesses, data lines are brought into the cache only as a result of references. Except where noted, a data reference may move in all of the lines on the page containing the reference. The following actions constitute a data reference, and may cause move-in to the data cache:

- Execution of a load, store, or semaphore instruction
- Interruption of a load, store, or semaphore instruction by any interruption except the ones listed
Because protection is checked (interruption 10), the reference cannot bring in any data which could not have been accessed.

- A load or store instruction which is left at the front of the interruption queues because of a prior instruction which took a group 4 interruption, provided that the load or store would not have taken any of the above interruptions (6, 7, 8, 10).

Data items which would have been referenced by a nullified load, store, or semaphore instruction are not moved in.

The instructions LDWA, LDDA, STWA, and STDA are exceptions to the general rule that a data reference may cause all of the lines in the page containing the reference to be moved in. These instructions can cause only the referenced line to be moved into the data cache.

### Instruction Cache Move-In

For virtual accesses, cache move-in is permitted only if there is a translation for the virtual address which meets both of these conditions:

- The page type field in the access rights for the entry contains a value in the range 2 to 7 (access rights allow execute access).
- The access rights for the entry does not match this binary pattern: “111 0X 1X”, where each X stands for either a 1 or a 0 (execute-only page where PL2 ≤ PL ≤ PL1 but PL2 > PL1).

For absolute accesses, instructions are brought into the instruction cache, or combined data and instruction cache, only as a result of references. Except where noted, an instruction reference may move in all of the lines on the page containing the reference, as well as all of the lines on the next sequential page. The following actions constitute an instruction reference, and may cause move-in to the instruction cache:

- Execution of an instruction
- Execution of a nullified instruction. This action can cause only those lines on the page containing the instruction to be moved in
- Execution of a branch can cause all of the lines on the page containing the target of the branch to be moved in.
- Execution of a branch to a target instruction which is the last instruction on a page, followed by an
instruction which traps (in the branch delay slot), can cause all of the lines on the page containing
the target instruction, as well as all of the lines on the next sequential page to be moved in.

• Interruption of an instruction by any interruption
• A branch instruction which takes a group 4 interruption can cause all of the lines on the page
containing the instruction which would have been branched to, to be moved in.

Instructions which would have been branched to by nullified or untaken branches are not moved in.

Programing Note
If a data page immediately follows an instruction page, it is possible that the entire data page
may have been moved into the instruction cache because of these move-in rules. Software must
be aware of this fact and flush both the instruction and the data caches in order to remove the
data page from the cache.

I/O Addresses and Uncacheable Memory
Accesses to the I/O address space, whether through absolute accesses or through virtual accesses that
map to the I/O address space, are never cached.

Virtual accesses for which the U-bit in the TLB entry is 1 are not cached.

Cache Flushing
Cache control instructions have two effects on caching. One is that they force particular lines to be
removed from the cache. The other is that, for absolute accesses, they disable further cache move-in
from the affected memory range until further references to that range are made.

For virtual accesses, a purge TLB instruction stops (disables) any subsequent move-in operations to that
page. Subsequent accesses must trigger a TLB miss, and then may move into the cache only if the new
translation from the page table allows it.

For absolute accesses, a flush cache or purge cache instruction to a page stops (disables) any subsequent
move-in operations to that page until another reference to that page is made. In a multiprocessor system,
these instructions stop any subsequent move-in operations to that page on all processors until another
reference to that page is made.

Once a line could have been brought into a cache, the only way software can insure that the line has
been removed from the cache is to

• Purge the translation from the TLB and insure that the corresponding page table entry does not
  allow cache move-in (if the cached memory was brought in due to a virtual access), and
• Flush or purge the line and execute a SYNC instruction, or flush the entire cache with flush-entry
  instructions and execute a SYNC instruction.

Once a line has been made cacheable, even if it is subsequently forced out of the cache by other
accesses, the cache system can move it in again at will, until the enabling translation is removed (for
virtual accesses) or until the line is flushed (for absolute accesses).
System software can create translations which allow a virtual address range to be flushed from the caches, but which do not enable cache move-in. There are two ways of creating such a translation:

- Create a translation in which the page type field in the access rights has the value 0 or 1 (this prevents I-cache move-in), and has the T-bit 1 (this prevents D-cache move-in), or
- Create a translation in which the page type field in the access rights has the value 7 (this prevents D-cache move-in), and has the special access rights pattern “111 0X 1X” (this prevents I-cache move-in).

**Cache Coherence with I/O**

Accesses to memory by I/O modules may be either coherent or non-coherent with processor data caches.

**Coherent I/O**

Processors in systems with coherent I/O modules must implement the LOAD COHERENCE INDEX instruction, which loads the coherence index corresponding to a given virtual address into a general Register. Coherent I/O modules provide the coherence index along with the absolute address of data it is reading from or writing to memory. The coherence index must provide enough information such that, together with the absolute address, the processor can find data that was brought into its data cache by the original virtual address.

Software need not flush or purge data from the data cache when sharing the data with a coherent I/O module. For I/O output (e.g., memory to secondary storage), the coherent I/O module performs coherent read operations which will read the data from memory or a processor's data cache depending on where the most up-to-date copy is located. For I/O input, the coherent I/O module performs coherent write operations which will write the data to memory and also update or invalidate matching lines in processor data caches.

Coherent I/O operations are not coherent with instruction caches. Software is responsible for flushing the appropriate instruction cache lines before or after the I/O operation.

**Non-coherent I/O**

Non-coherent I/O modules process data in memory; this data can be non-coherent with processor caches. Software is required to ensure that:

1. The contents of the appropriate caches are flushed to main memory prior to an I/O output (e.g., memory to secondary storage) operation.
2. The contents of the appropriate caches are purged or flushed prior to an I/O input (e.g., secondary storage to memory) operation.
3. The contents of the appropriate caches are purged following an I/O input operation, if the cache move-in rules would have allowed the processor to move the data into the cache during the I/O operation.
Operations Defined for I/O Address Space

Semaphore instructions are undefined if the address maps to the I/O address space. Additionally, semaphore instructions are undefined if the address maps to a page in the memory address space for which the TLB U-bit is 1.

Cache flush and purge instructions execute as NOPs if the address maps to the I/O address space.

Data prefetch instructions directed to the I/O address space or to a page in the memory address space for which the TLB U-bit is 1 behave as described in “Data Prefetch Instructions” on page 6-11 except that they may not affect the cache state and may optionally generate a transaction to the addressed device.

All accesses other than those listed above are defined to the I/O address space.

Cache and TLB Coherence in Multiprocessor Systems

Multiprocessor systems may include PA-RISC processors as well as other processors.

The cache-coherent part of a multiprocessor system is required to behave as if there were logically a single D-cache and a single I-cache. If there are multiple physical D-caches, they must cross-interrogate for current data and must broadcast purge and flush operations except for FDCE and FICE. Purge and flush operations do not cause TLB faults on other processors. Multiple I-caches require only that flushes be broadcast. The I-cache is read-only, and software is responsible for coherence when modifications are made to the instruction stream.

The non-cache-coherent part of a multiprocessor system (if any) may either cross-interrogate with the caches in the cache-coherent part of the system, or may have an independent cache system. This design decision is generally based on the frequency of data sharing.

In the cache-coherent part of a multiprocessor system, all data references to cacheable pages must be satisfied by data that was obtained using cache coherence checks, and has remained coherent since the data was moved in. Data references to uncacheable pages do not need to be satisfied by data that was obtained using cache coherence checks. Data from an uncacheable page could be in a cache if it was moved in when that page was marked cacheable, but the page is now marked uncacheable.

Implementations with write buffers must also check buffer contents on cache coherence checks, in order to insure proper ordering of storage accesses.

Instruction references need not be satisfied by data that was obtained using cache coherence checks.

Instruction caches are read-only. In the case of a separate instruction cache implementation, instruction cache lines must never be written back to main memory.

Each processor in a multiprocessor system must have its own TLB system. All TLBs in a multiprocessor system are required to broadcast global purges to all other TLBs. The originating processor’s purge instruction suspends until all target processors complete the purge. TLB inserts, local purges and the PDTLBE and PITLBE instructions are not broadcast and do not affect translation on other processors.