PA-8500: The Continuing Evolution of the PA-8000 Family

Gregg Lesartre
Doug Hunt

Hewlett-Packard Company
3404 East Harmony Road, MS 55
Fort Collins, Colorado 80525-9599

Abstract

The PA-8500 is the newest member of the PA-RISC family of processors. The design is based on the PA-8000 and PA-8200 processors, but is implemented in a 0.25 micron process. The new process allows the large first level caches to be moved on-chip so that the frequency can be boosted without the need to add tightly-coupled second level caches. Improvements have also been made in the branch prediction hardware to allow a single branch prediction structure to take advantage of static and dynamic branch prediction techniques seamlessly. These improvements and others will allow the PA-8500 to deliver industry-leading application performance.

1. Introduction

The PA-8500 processor, now under development at Hewlett-Packard’s Engineering Systems Lab in Ft. Collins, Colorado, is the latest addition to the PA-RISC[1,2] family of processors. The design goals of the development project are to deliver a processor which will:

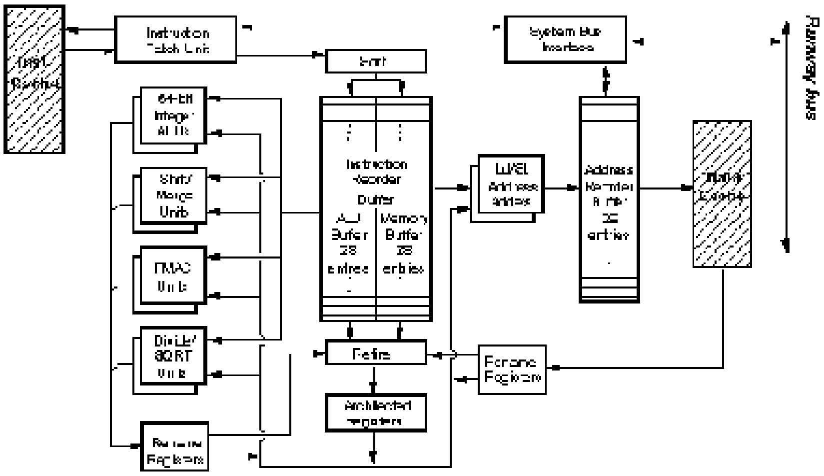
- Lead the industry in application performance.
- Provide full binary compatibility with all existing PA-RISC binaries.
- Deliver maximum performance on binaries tuned for the PA-8000 and PA-8200.
- Interface to the Runway system bus.

The PA-8500 design builds upon the foundation established in the PA-8000[3] (see figure 1). With its large complement of functional units, dual-ported data cache and sophisticated instruction reordering hardware, the PA-8000 effectively exploited instruction level parallelism to set new standards of performance. The PA-8200[4] design improved upon the PA-8000 by increasing the sizes of the caches, Branch History Table (BHT) and Translation Lookaside Buffer (TLB) as well as increasing the operating frequency. In order to deliver still higher levels of performance, the PA-8500 design emphasizes increased frequency over improvements to Cycles Per Instruction (CPI) making the most of the infrastructure established in the PA-8000. Higher frequency is achieved by implementing the PA-8500 in a 0.25 micron process, which also allows the inclusion of enough SRAM to implement effective first level on-chip caches. Although priority has been given to increasing the frequency of the PA-8500, some CPI improvements have also been incorporated. Most notable among these improvements is a change to the branch prediction hardware which allows static and dynamic branch prediction techniques to be combined seamlessly.

2. Primary Caches: The larger the better

Some benchmarks have run well using small one-cycle, on-chip caches backed by a larger level two (L2) cache. Since many on-chip caches are too small to hold the quantity of data necessary to complete complicated tasks, more demanding applications sometimes suffer from this arrangement. To support commercial applications such as transaction processing, a large instruction cache with high bandwidth is required. Implementing large, two-cycle primary caches in the PA-8x00 family of microprocessors has effectively improved performance across a wide range of workloads by significantly reducing first level cache misses.

In instances where the miss-rate is high, L2 cache latency and bandwidth become a significant impediment to performance. Level two caches cycle slower than level one (L1) caches and have lower bandwidth to move more data. The latency incurred by accessing the slower L2 cache is compounded by the need to retrieve a full cache line to fill the L1 cache rather than just the data being currently accessed.



Large L1 caches avoid the penalties of a multilevel cache structure by providing a high bandwidth instruction and data paths directly to a large store of data. Hewlett- Packard has consistently taken this approach in its high-end processor design. Now, using 0.25 micron technology we can implement these caches on-chip to improve their cycle time and to reduce the overall system cost.

2.1 The PA-8500: 1.5 MB of on-chip cache

The PA-8500 provides access to more data in two cycles from its 0.5MB instruction and 1MB data caches than many systems provide in 10 or more cycles from an L2 cache. The reordering queues inherited from the PA-8000 do an excellent job extracting useful independent work from the instruction code stream. The processor’s functional units can therefore keep busy, rather than stalling as a conventionally pipelined processor would have to do during a two state cache access.

One of the challenges in developing the PA-8500 is to create an on-chip cache that can fit into the allocated die area and still keep up with the reorder queues. This requires the data cache to support two simultaneous memory operations while maintaining a two cycle access. HP accomplished this by using the same dual bank system developed for the PA-8000’s off-chip data cache. With this system the cache can be implemented with a simple single port RAM design, conserving area. Since each lookup needs to concern itself with only half the cache, the physical dimensions traversed during a cache access are reduced, supporting a quicker cache access time.

Access to the two banks of the cache are controlled by the address reorder buffer (ARB) portion of the PA-8500 reorder queue. The 28 entry ARB receives addresses that have been calculated by the PA-8500's address adder units. The ARB prioritizes memory accesses that have had their addresses calculated, and then picks one even and one odd access to use the cache each cycle. A bypass path is provided to route an address directly from the address units if there is no outstanding access held in the ARB.

This arrangement enables the cache ports to be efficiently utilized even when simultaneously calculated addresses happen to access the same half of the cache. Any delay in getting data in this scenario is hidden by the out-of-order queues.

Each 0.5MB data cache bank is implemented as four one-eighth megabyte arrays, each providing a double-word of data plus error correction bits. Data is organized within the arrays such that a full cache line can be addressed at one time, or four ways of associativity can be addressed together. The cache line tags are held in four smaller separate and independently addressable RAM arrays. In this way the data and tag can either be accessed together for a data read, or independently to effect a data store at the same time another store is accessing its cache line status.

The PA-8500 instruction cache is a 0.5MB four-way set associative pipelined cache that provides 128 bits of instruction plus predecode bits per cycle to the instruction fetch engine.

For most workloads, a set associative cache is more likely to hit than an equivalently sized direct mapped cache. As a direct mapped cache fills up with useful data, it becomes increasing unlikely that the next piece of requested data will find an unoccupied location in cache to use. Set associative caches increase hit ratios because there are multiple locations that can be used to store a given piece of data.

The PA-8000 has more than 500 I/O's dedicated to the off-chip caches. These are needed to provide the bandwidth required to feed the processor's computation units. Increasing the number of I/O's to support an off-chip associative lookup was not a cost effective option for the PA-8000. Since the PA-8500 cache is put directly on-chip, however, it requires only a few wires to add set associativity to the cache, and still support the required bandwidth. Once set associativity is added, the cache operates more efficiently which effectively increases the cache size as it is measured by cache hit rate.

2.2 Data Integrity: Single bit error correction

All data stored in the PA-8500 caches are protected from single bit errors. Since there are 1.5MB of instruction and data cache on the PA-8500, covering three-quarters of the die, it is imperative that safeguards be taken to maintain data integrity.

For the instruction cache, simple parity is sufficient since its contents are always clean. Any time an instruction access encounters an error, the access is treated as a cache miss. Cache lines with corrupted data are invalidated, and the data is re-accessed from memory.

It takes more effort to protect the data cache since error correction is required when a dirty line in cache becomes corrupted. The PA-8500 provides six extra bits per word to enable single bit error correction to protect the cache data. The correction, however, is not directly in the cache access path where it would lengthen the critical cache access latency. Errors are instead detected by parallel error checking logic. If an error is detected, the corrupted data is forced out of the cache. The data is corrected in the copy out path if the line is dirty. If the line is clean, it is invalidated. The access is then re-executed, causing the line to be brought back into cache with the corrected data.

The data cache's tag RAMs are also correctable, but in a different way. The PA-8500 maintains two copies of each cache line's tag to allow two accesses to be serviced simultaneously. Each is protected with parity. If an error is detected in one of the tags, a copy out of the line is started. During the copy out, the tag array that does not have the parity error provides the physical address for memory and the correct status. If the status indicates the line was dirty, the line is copied back to memory. If the line was clean, the cache line is invalidated. Once the cache has been scrubbed in this fashion, the access is executed again to bring the error-free line back into cache.

2.3 On-Chip Caches: No RAMs, lower cost

With on-chip primary caches larger than most L2 caches, combined with the scheduling capabilities of its 56 entry reorder queue, the PA-8500 doesn't need to support a tightly coupled L2 cache directly from the CPU. This eliminates the need to include an integrated L2 cache controller. In addition, there is no need to dedicate a large number of I/Os to connect external RAMs.

A minimum configuration PA-8000 processor requires the CPU and eleven RAMs. A PA-8500 with more cache requires just the CPU in a low pin-count package. This enables the PA-8500 to serve the low end of the PA-RISC product space as well as the high.

3. Branch Prediction: Static and dynamic

Branch prediction is a hot topic these days, and for good reason. Today’s high frequency, superscalar processors pay a high cost every time they guess wrong about the actual path program execution takes. This penalty is high due to increased pipeline depth, but just as importantly, mispredicted branches prevent the processor from effectively exploiting instruction level parallelism across branches. Considering the importance of the topic, it is no surprise that many approaches to the problem of reducing branch penalties have been proposed and implemented [5,6,7,8,9,10,11]. These approaches may be roughly categorized into static methods and dynamic methods, and each has its own strengths and weaknesses[12].

Static methods[6,13] rely on the compiler to optimize branch performance. Since the optimizations are performed when the program is compiled, the optimizations do not change as the program runs. These optimizations may be based on execution profile information gathered during a training run (Profile Based Optimization or PBO) or they may be based on the structure of the source code by using heuristics[14]. Examples of compiler branch optimization include eliminating certain branches altogether using predication [15,16,17], and encoding hints in branch instructions which tell the hardware the likely direction each branch will follow. For branches which are strongly biased, that is, which are either almost always taken or almost always not-taken, compiler hints can be very accurate and produce excellent results. The principal advantage of compiler branch optimizations is that they do not consume chip area. Since there is no physical structure on the chip, compiler-based optimizations do not have the capacity limits that dynamic branch prediction encounters. For example, if the compiler can hint each branch with its expected direction, then most branches will be predicted fairly well, no matter how many branches are in the program. Of course, branches which the compiler eliminates cannot ever be mispredicted.

In contrast with static branch prediction, dynamic branch prediction methods[5,8,9,10,11] predict the future behavior of a branch based on the past behavior of that branch and/or other branches during program execution. Branch history information is collected using special hardware, and the processor bases its prediction on the history it has collected. Dynamic branch prediction can perform better than static branch prediction under many circumstances. For example, if a given branch tests whether a certain command line option was given, then that branch will go the same way each time it is executed in a given run of the program, but the compiler cannot specify that direction since it is not known until the program is run. Another stumbling block for static prediction is when a branch changes its characteristics over time. A branch in a sort routine might be mostly not-taken on an unsorted list but become mostly taken as the list becomes more sorted. Even the simplest dynamic branch prediction methods perform better than static branch prediction on such "mode-dependent" branches.

One can argue that if the only help the compiler can provide (outside of branch elimination) is to hint a branch to be taken or not-taken forever, a simple dynamic scheme will be able to perform just as well, and it will also be able to handle the mode-dependent branches which static branch prediction does not address. The Achilles' Heel of dynamic branch prediction, however, is a limited resource for collecting branch statistics. If two branches are strongly biased in opposite directions and map to the same location in the branch prediction hardware, they will interfere with one another and the prediction accuracy of each will suffer[18,19].

Since both static and dynamic branch prediction methods have their advantages under different circumstances, the design of the PA-8000 accommodated both static and dynamic prediction modes. At compile time, the application developer can mark a binary to be run in one mode or the other depending on which performs better for that program. The design of the PA-8500 increases this flexibility by providing a means to combine the advantages of static and dynamic branch prediction.

The Branch History Table (BHT) in the PA-8500 is a standard array of two bit counters [5,7], but the information stored in the counters is not the direction of the branch (taken or not-taken). Rather, the counters record whether or not the branch went in the direction indicated by the static hint supplied by the compiler. If the static hint disagrees with the actual direction the branch followed, the counter is incremented; if it agrees, the counter is decremented. Each time a branch is fetched, the BHT is consulted and, if the counter is zero or one, the static hint encoded in the instruction is followed. If the counter is two or three, the hardware predicts that the branch will go in the direction opposite the static hint.

One can see that, if the compiler has done a good job of setting the static hints and most of the branches are strongly biased, most of the counters in the prediction cache will end up with counts of zero or one. For branches which are incorrectly hinted, the counts will tend to be two or three and the hardware will override the static hint provided by the compiler. With this algorithm, two branches of opposite bias which map to the same location in the BHT do not interfere with one another if each is hinted correctly.

The modified BHT enables the PA-8500 to combine the advantages of static and dynamic branch prediction methods. The combination of static and dynamic information is achieved with a single hardware structure, rather than two or three hardware arrays as some other prediction methods require. Of course, this method of combining static and dynamic information could be combined with many other more elaborate dynamic branch prediction schemes. At this design point, however, HP believes maximizing the L1 cache available on-chip is more beneficial than adding more hardware to provide an incremental improvement in branch prediction accuracy. Finally note that the enhancements made to branch prediction for the PA-8500 are backward compatible with the PA-8200 and PA-8000.

4. PBO: Performance path to the future

The radical new design of the PA-8000 opened up many new opportunities for compiler optimizations. In addition, the PA-8000 was the first processor to implement the new PA-2.0[2] architecture. This meant that much effort went into the development of the compilers and into tuning applications. By leveraging the core micro-architecture of the PA-8000, the PA-8500 ensures that all of the investment in compilers and application tuning is protected.

It is also important to note that the employment of Profile Based Optimization (PBO) has come of age with the PA-8000 family of processors. This technique has demonstrated its value by enabling the compiler to perform optimizations which are not possible without knowledge of the dynamic behavior of the program. PBO also enables the compiler to more accurately encode static branch prediction hints. The investments which have been made by software developers in moving to the PBO paradigm will continue to pay off more and more as compilers improve in their ability to exploit the ever increasing potential of the processors of the future.

5. Other features

In addition to increasing the frequency, adding cache associativity and increasing branch prediction flexibility, the following performance enhancements have been incorporated into the PA-8500:

- The size of the TLB has been increased from 120 to 160 entries.
- Modifications have been made to support a higher performance system bus.
- Support for higher graphics bandwidth has been added.
- Various performance penalties have been removed or reduced.

6. Conclusion

The PA-8500 is the next member of the PA-RISC processor family. By migrating to a 0.25 micron process, implementing large, associative on-chip caches, and enhancing the branch prediction hardware the PA-8500 will deliver industry leading performance on binaries tuned for the PA-8000 microarchitecture.

Acknowledgments

The authors would like to acknowledge all of the individuals who have worked so hard to make the PA-8500 possible, which includes everyone who worked on the PA-8000 and PA-8200. Special thanks are also in order for the language operation as well as individuals in the performance community and the field who have worked and continue to work to bring the full potential of the PA-8000 family of processors to customers.

References

[1] R. B. Lee, "Precision Architecture", IEEE Computer, Vol. 22, No. 1, January 1989, pp. 78-91.
[2] G. Kane, PA-RISC 2.0 Architecture, Upper Saddle River, NJ: Prentice Hall PTR, 1996.
[3] D. Hunt, "Advanced Performance Features of the 64-bit

PA-8000", Compcon Digest of Papers, (March 1995), pp. 123-128.
[4] L. Gwennap, "HP Pumps Up PA-8x00 Family", Microprocessor Report, Vol. 10, No. 14, October 28, 1996.
[5] J. E. Smith, "A Study of Branch Prediction Strategies", 8th International Symposium on Computer Architecture, (1981), pp. 135-148.
[6] S. McFarling and J. Hennessy, "Reducing the Cost of Branches", 13th International Symposium on Computer Architecture, (June, 1986), pp. 396-403.
[7] J. Lee and A. Smith, "Branch Prediction Strategies and Branch Target Buffer Design", IEEE Computer, Vol. 17, No. 1, (January, 1984), pp. 6-22.
[8] L. Gwennap, "New Algorithm Improves Branch Prediction", Microprocessor Report, Vol. 9, No. 4, March 27, 1995, pp. 18-19.
[9] T. Yeh and Y. Patt, "Two-Level Adaptive Branch Prediction," 24th ACM/IEEE International Symposium on Microarchitecture (Nov. 1991), pp. 51-61.
[10] M. Evers, P. Chang and Y. Patt, "Using Hybrid Branch Predictors to Improve Branch Prediction Accuracy in the Presence of Context Switches", 23rd Annual International Symposium on Computer Architecture, (May, 1996), pp. 3-11.
[11] S. T. Pan, K. So and J. Rahmeh, "Improving the Accuracy of Dynamic Branch Prediction Using Branch Correlation, 5th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS), (October, 1992), pp. 76-84
[12] W. Hwu, T. Conte and P. Chang, "Comparing Software and Hardware Schemes For Reducing the Cost of Branches", 16th International Symposium on Computer Architecture, (May, 1989), pp. 224-231.
[13] J. Fisher and S Freudenberger, "Predicting Conditional Branch Directions From Previous Runs of a Program", 5th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS), (October, 1992), pp. 85-95.
[14] T. Ball and J. Larus, "Branch Prediction For Free", Proceedings of the SIGPLAN Conference on Programming Language Design and Implementation, (June, 1993), pp. 300-313.
[15] P. Vogel and B. Holmer, "Analysis of the Conditional Skip Instructions of the HP Precision Architecture", 27th International Symposium on Microarchitecture, (November, 1994), pp. 207-216.
[16] G. Tyson, "The Effects of Predicated Execution on Branch Prediction", 27th International Symposium on Microarchitecture, (November, 1994), pp. 196-206.

[17] S. Mahlke, et. al., "Characterizing the Impact of Predicated Execution on Branch Prediction", 27th International Symposium on Microarchitecture, (November, 1994), pp. 217-227.
[18] A. Talcott, et. al., "Influence of Branch Prediction Table Interference on Branch Prediction Scheme Performance", Proceedings of the Conference on Parallel Architectures and Compilation Techniques, (1995), pp. 89-98.
[19] S. Sechrest, C. C. Lee, and T. Mudge, "Correlation and Aliasing in Dynamic Branch Predictors", 23rd International Symposium on Computer Architecture, (May, 1996), pp. 22-32.